Django -Lab 1

Django – Djongo –MongoDB

# Environment Setup

1. Type “Anaconda Navigator” in search bar (Windows Button).
2. Till redirect you to Anaconda Navigator and there “Launch Cmd.exe Prompt”
3. Go to the folder where you want the Application to be
4. In the terminal, to create a virtual environment with conda, run the following command:

**conda create --name newEnv django**

**(base) D:\ML\_DL\_NLP\Django>conda create --name newEnv django**

Here, we created an environment called “newEnv” with the latest version of Django. Proceed with the default packages when prompted (i.e. select y)

1. To activate the environmen, run:

**activate newEnv**

**(base) D:\ML\_DL\_NLP\Django>activate newEnv**

Note: When an environment is activated, anything installed with pip or conda will only be installed for the given environment

1. If you get a django not intalled error, run the following command to install Django in this new environment

**pip install django**

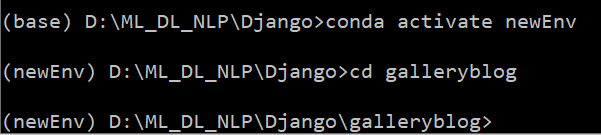
**(newEnv) D:\ML\_DL\_NLP\Django>pip install django**

# Create project

1. Create a folder called “django”. Use this folder to store all your Django projects.
2. Navigate to the “django” folder [Where you have created] in command-line using cd command and open the folder in atom’s project panel.

This is an example shown below:

Use your folder location using cd command

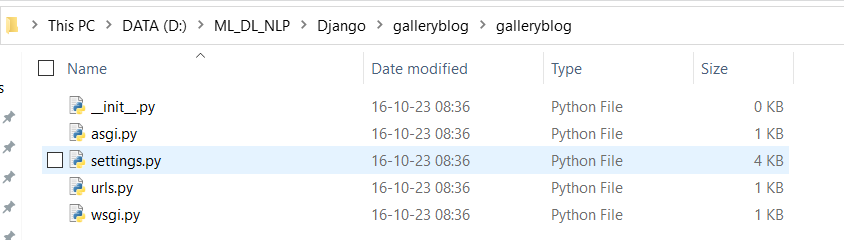


1. Using Django-admin, create your first project:

**django-admin startproject <app-name>**

**(newEnv) D:\ML\_DL\_NLP\Django>django-admin startproject galleryblog**

1. You will notice a folder “first\_project” created with the following files:



You have now successfully created a django project.

# Test Run

1. Now **navigate into the galleryblog** folder using cd command and run the following command:

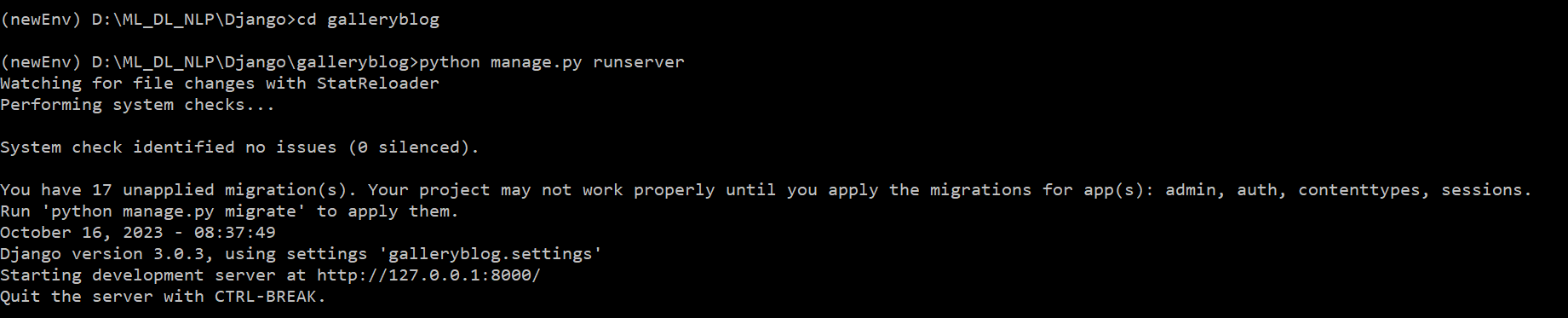
(newEnv) D:\ML\_DL\_NLP\Django>cd galleryblog

**To run the server: Python manage.py runserver**

(newEnv) D:\ML\_DL\_NLP\Django\galleryblog>**python manage.py runserver**

Starting development server at http://127.0.0.1:8000/

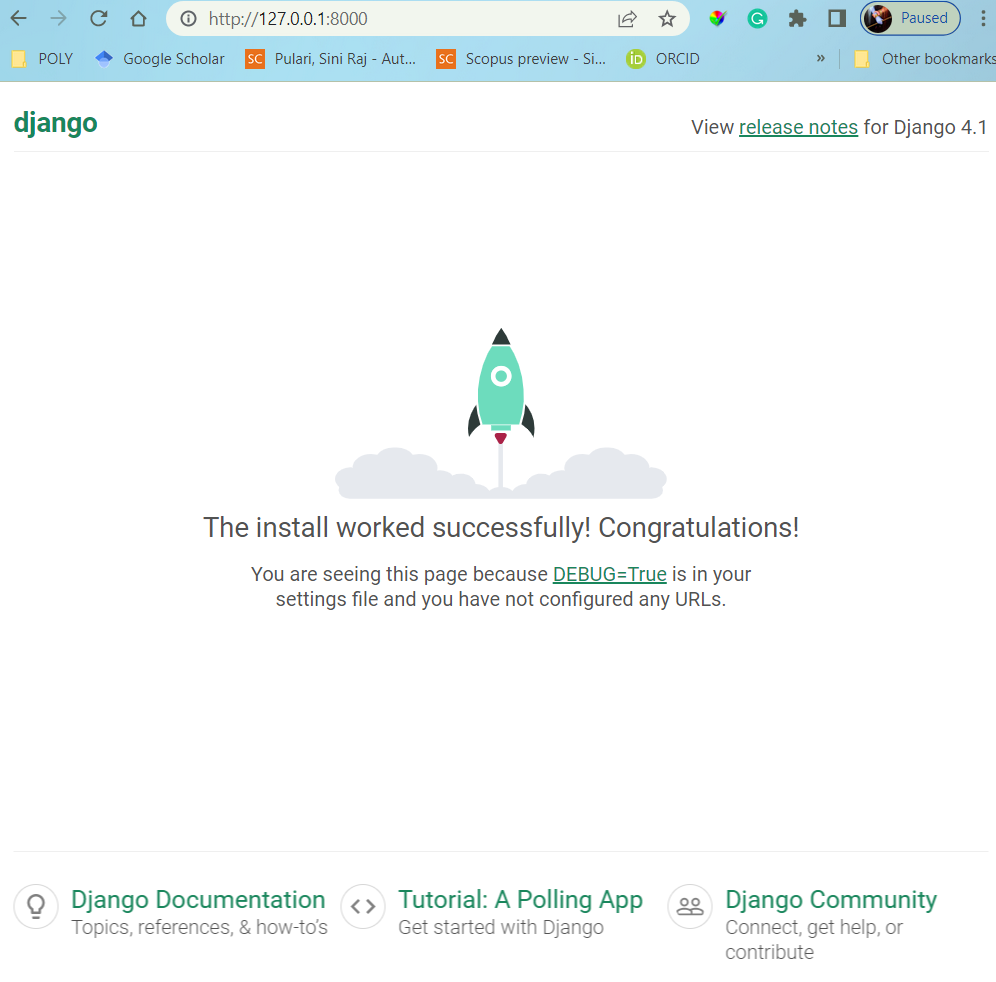
Output:



(Ignore migration setting error messages)

1. To run the project, copy the URL and paste it in your web browser:

**http://127.0.0.1:8000/**



Create a superuser – Django Admin

1. To create a superuser to access the Admin application, run the createsuperuser command in CMD:

**python manage.py createsuperuser**

1. You will be prompted to enter a username and password: enter **admin** and **abcd1234** for username and password respectively.

**(newEnv) D:\ML\_DL\_NLP\Django\galleryblog>python manage.py createsuperuser**

**Username (leave blank to use 'sinik'): admin**

**Email address:** [**admin@gmail.com**](mailto:admin@gmail.com)

**Password:**

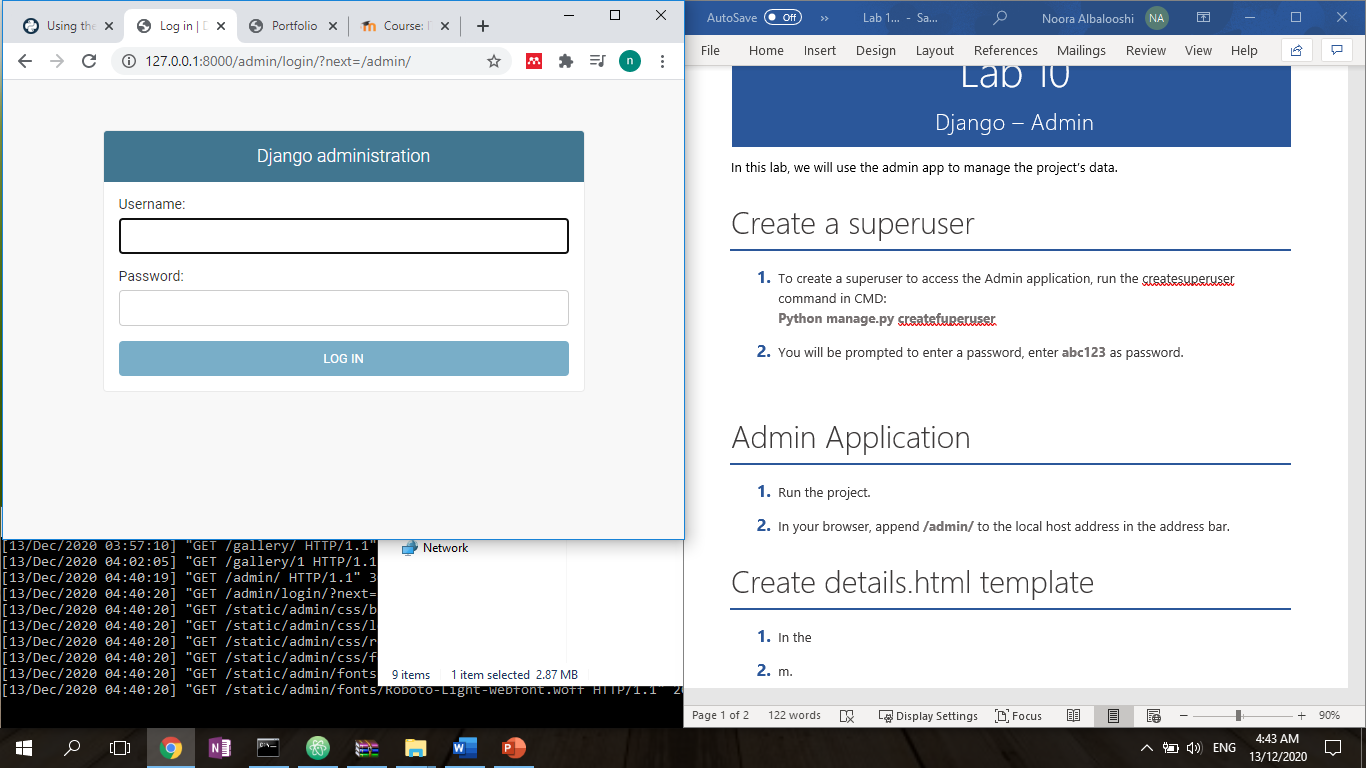
**Password (again):**

**Superuser created successfully.**

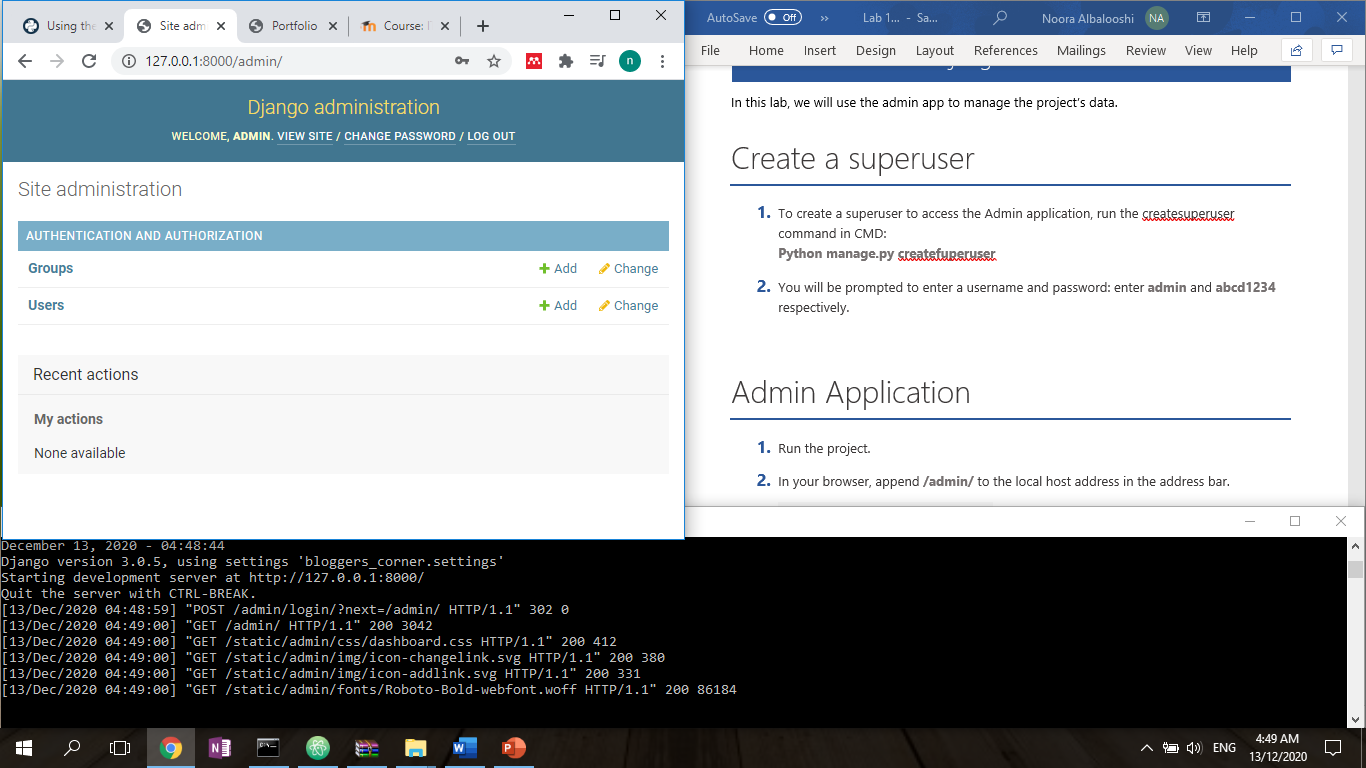
Admin Application

1. Run the project.
2. In your browser, append **/admin/** to the local host address in the address bar.

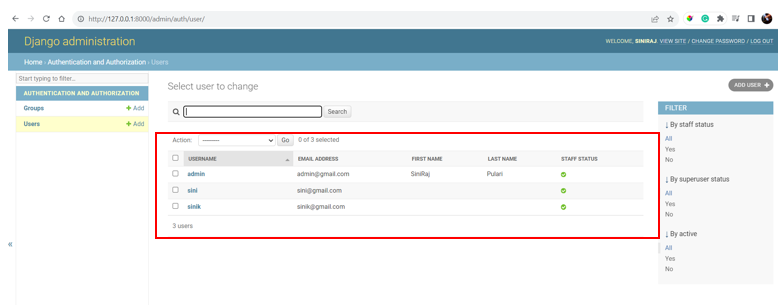
<http://127.0.0.1:8000/admin/>



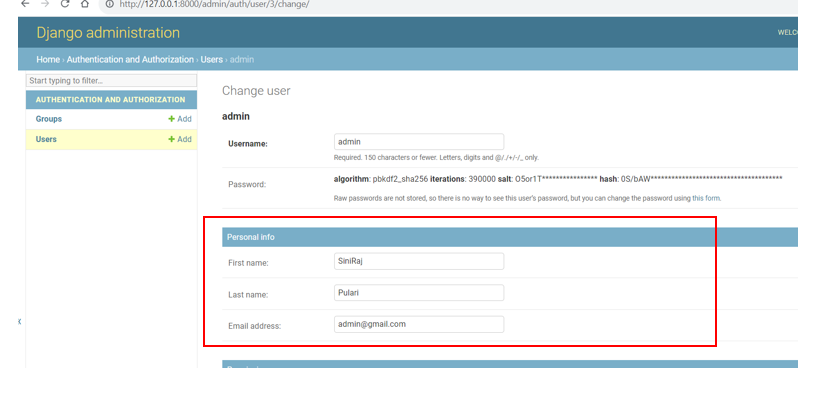
1. Enter the username and passowrd as defined in the previous section.
2. Once logged in, the following details are displayed:



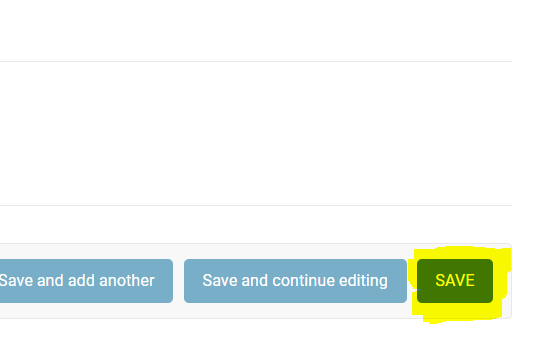
1. Click on users to see a list of registered users.



1. Click on the Admin User



1. Then scroll down and click save

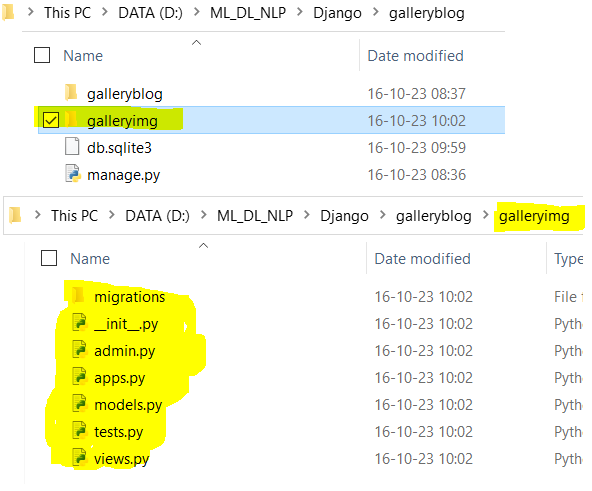


Creating the Applications in Django

1. Now, to create an application within this project, run:

**(newEnv) D:\ML\_DL\_NLP\Django\galleryblog>python manage.py startapp galleryimg**

“galleryimg” folder will be created within your project. These are your application files.



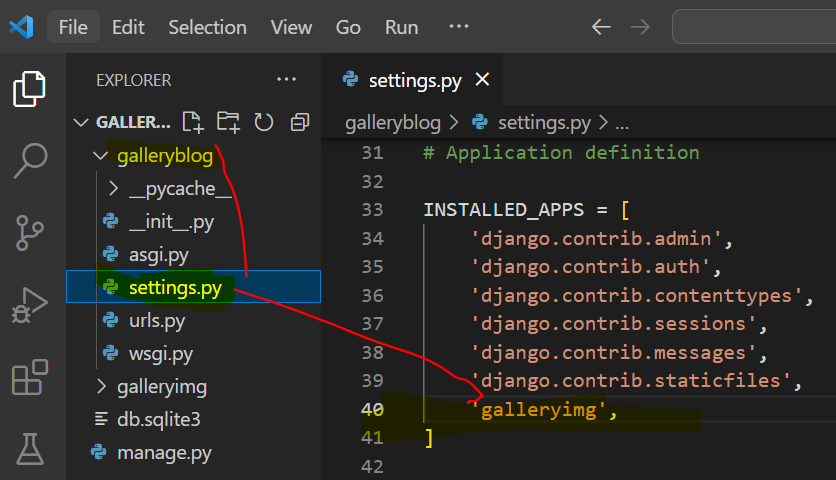
Please note!!!

**Please understand galleryblog is the project! And galleryimg is the application in Django!**

**Hence whichever name you use doesn’t matter , however, please use the names appropriately while programming the code!**

GalleryBlog –settings.py

1. Add the application ‘galleryimg’ to the project **galleryblog/settings.py** installed\_app list in settings.py



Settings.py Installed Apps section looks like as follows:

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

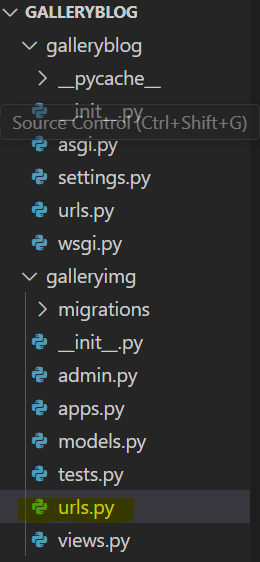
    'galleryimg',

]

Urls.py in galleryimg folder

1. To create a list of urls for the gallery application, create a urls.py file inside the **galleryimg** application folder.

**galleryimg/urls.py**



1. In galleryimg/urls.py file, add the following code:

Galleryimg/urls.py

from django.urls import path

from galleryimg import views

urlpatterns=[

    path('',views.gallery\_list)

]

In the code above, we import path to be able to create paths for our urls and import views from gallery.

Then we add a path that directs us to gallery\_list view

[**Please note we learn more about list in another exercise**]

**List view** helps us to view multiple items, **detailed View** help us to get the information of one data, **create view** helps us to create new views!

Note we will create this view in the coming steps.

1. We need to update about the same in the galleryblog/urls.py too

Since the project directly accesses the projects URLs, we need to map galleryimg application’s URLs in galleryblog/urls.py as shown below:

Add the code

path('gallery/', include('gallery.urls'))

galleryblog/urls.py

from django.contrib import admin

from django.urls import path,include

from galleryimg import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('galleryimg.urls')),

]

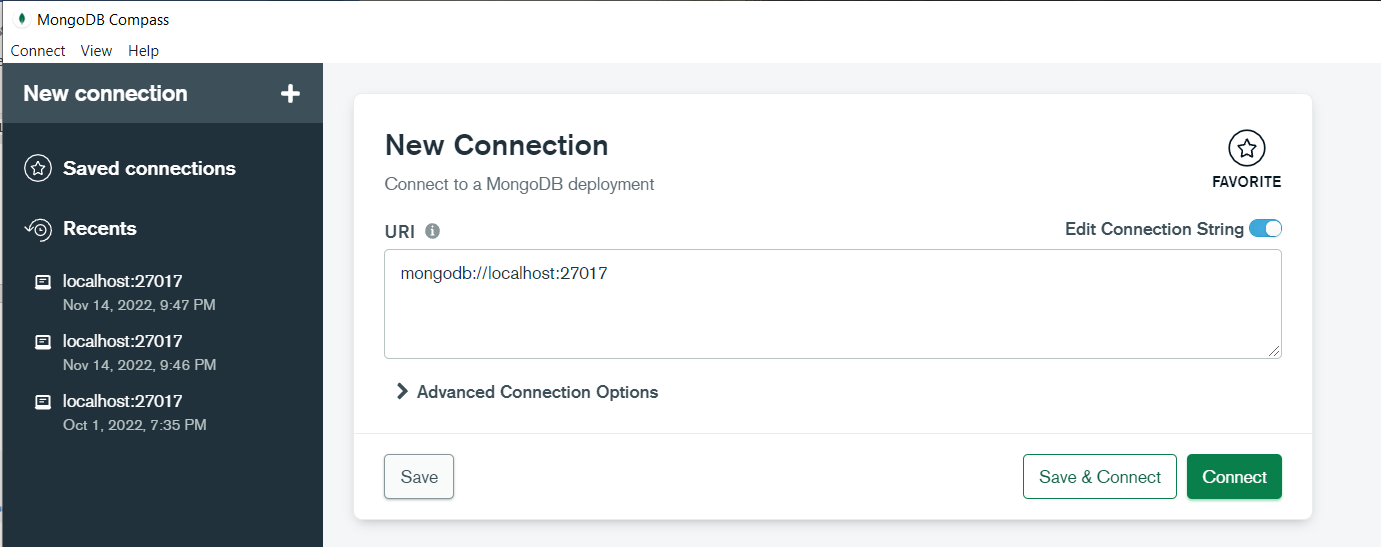
Here, we are importing include to be able to include other urls files.

We are identifying the application folder in which the new file is stored ‘gallery/’ and including the file.

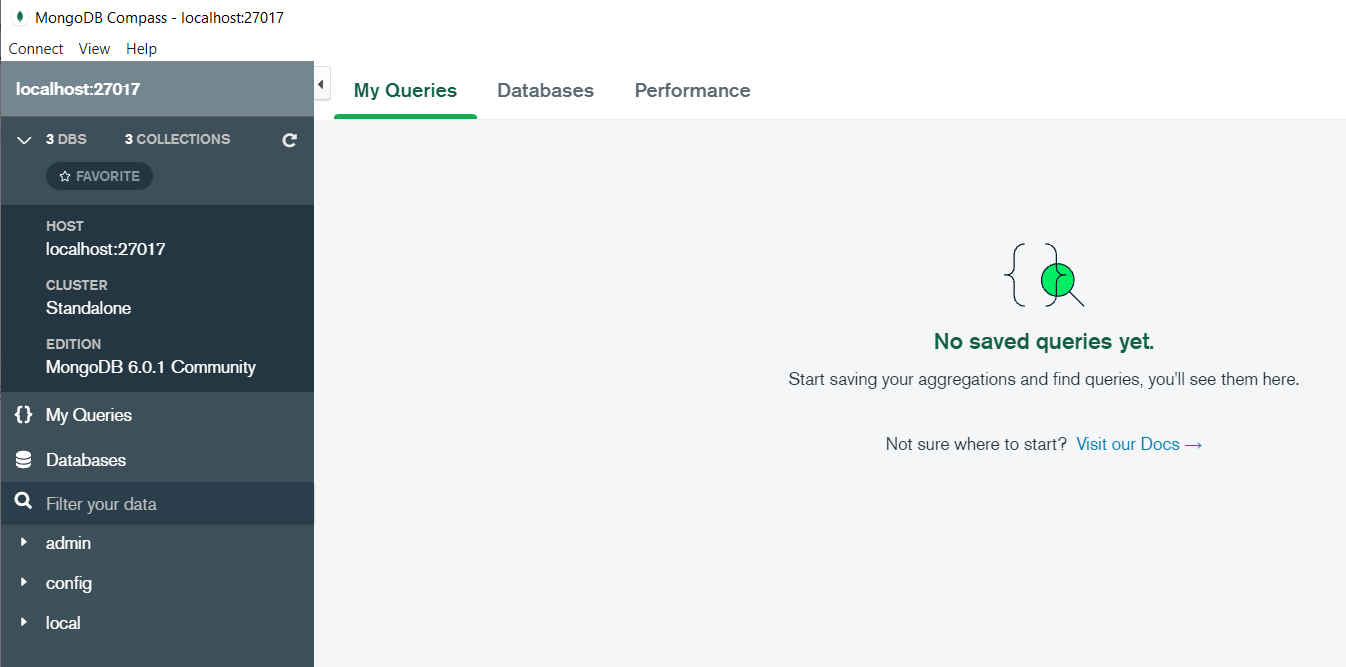
# Database setup - Compass

Skip step 1, 2 and 3 if you already installed MongoDB Compass

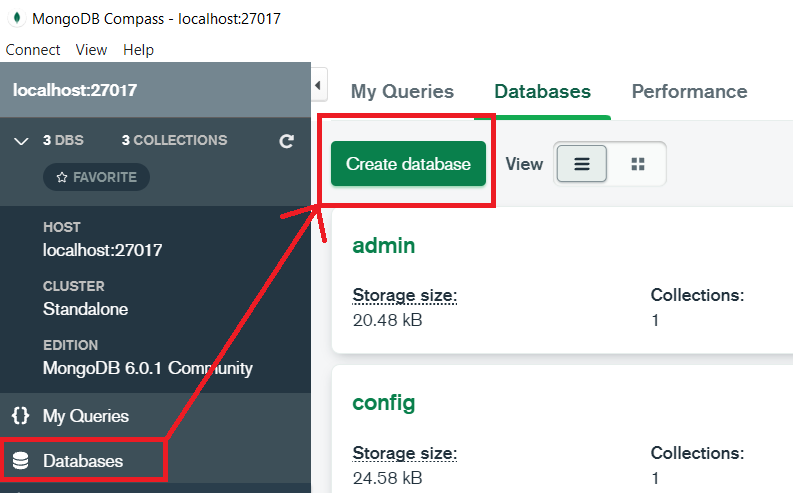
1. Download and install MongoDB Compass. MongoDB Compass provides a graphical user interface to manage your databases. <https://www.mongodb.com/try/download/compass?tck=docs_compass>
2. Run MongoDB Compass application.
3. In the first window, select connect; this allows you to connect to the local mongoDB server.



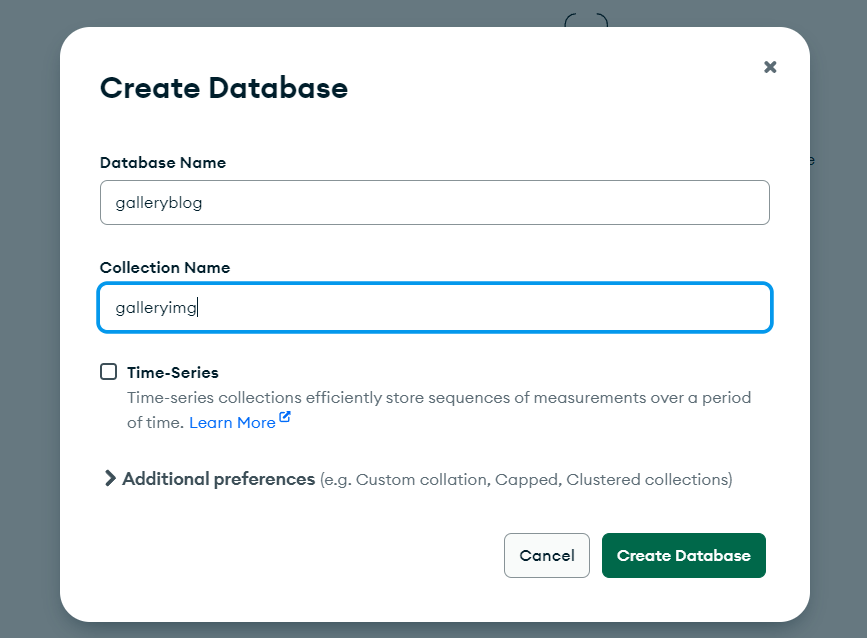
Click Connect and you will see something like given below:



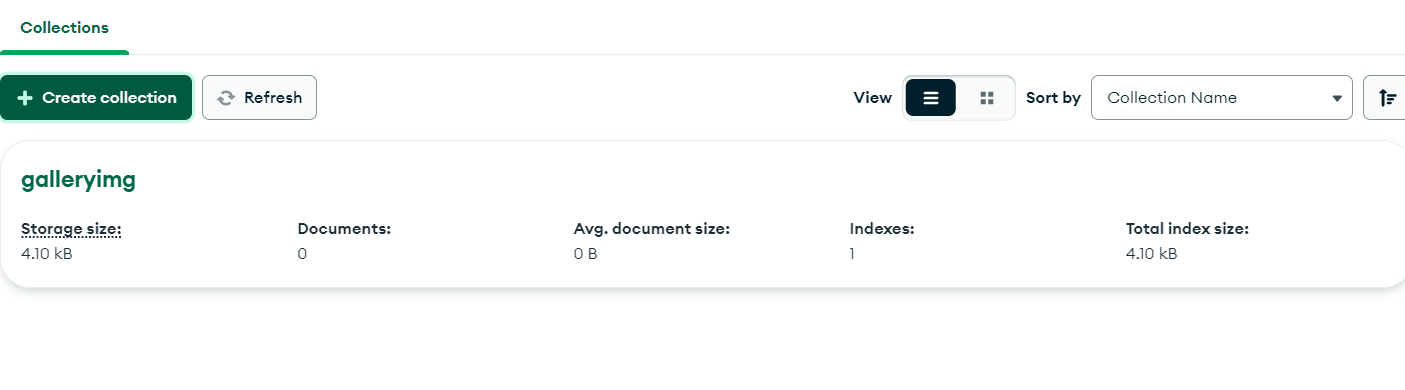
1. Click on Database and it will show Create database



1. Create database called “blog” by clicking on CREATE DATABASE. We will store our applications data in this database.

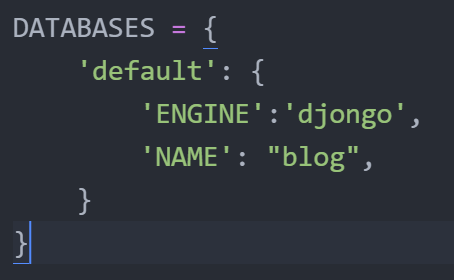


1. Click on galleryblog database that appears in the list of databases. The following window displays the list of collections in the database. (currently only galleryimg)



# Database setup - Django

1. Now update the database information in the galleryblog/settings.py file in your project folder as below:



Code in galleryblog/settings.py

DATABASES = {

    'default': {

        'ENGINE': 'djongo',

        'NAME': 'galleryblog',

    }

}

# Create view

1. **In “galleryimg/views.py”,** create a view function

Code as follows:

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def gallery\_list(request):

    return HttpResponse("Are we on the right track!!!")

We imported HttpResponse object.

We created a view function that returns a HttpResponse object to display a plain text message.

# Create model

Now we will build our model class for the galleryimg application. In **galleryimg/models.py** file, include the following code:

In this class, we have 3 attributes: title, description and image.

* Title is of type CharField and has a maximum length of 100 characters
* Description is of type TextField
* Image is of type CharField, where the path and name of the file will be stored

Next, we will see how Django automatically creates collections for our model class.

from django.db import models

# Create your models here.

class Gallery(models.Model):

    title=models.CharField(max\_length=100)

    description=models.TextField()

    image=models.CharField(max\_length=100)

1. Run the following commands

As you have done changes in the models.py

1. (newEnv) D:\ML\_DL\_NLP\Django\galleryblog>**python manage.py makemigrations**

Migrations for 'galleryimg':

galleryimg\migrations\0001\_initial.py

- Create model Gallery

1. (newEnv) D:\ML\_DL\_NLP\Django\galleryblog>**python manage.py migrate**

Operations to perform:

Apply all migrations: admin, auth, contenttypes, galleryimg, sessions

Running migrations:

No migrations to apply.

1. (newEnv) D:\ML\_DL\_NLP\Django\galleryblog>**pip install Djongo**

**[ Yes , it is not a mistake, it is Djongo and not django…]**

Djongo is an improvement over PyMongo in that developers need not write lengthy queries. It maps Python objects to MongoDB documents, i.e., Object Document Mapping (ODM). Djongo ensures that only clean data enters the database. By performing integrity checks, applying validations, etc. with Djongo, there is no need to modify the existing Django ORM.

1. (newEnv) D:\ML\_DL\_NLP\Django\galleryblog>**python manage.py runserver**

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

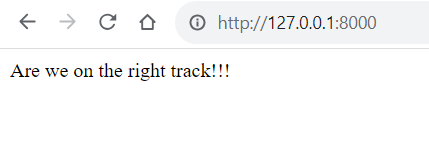
October 16, 2023 - 13:33:54

Django version 3.0.3, using settings 'galleryblog.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.

1. Observe the output

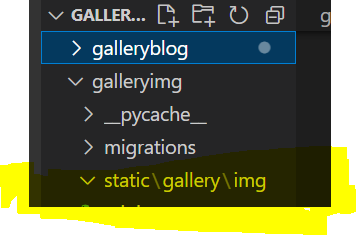


# Create a galleryimg object

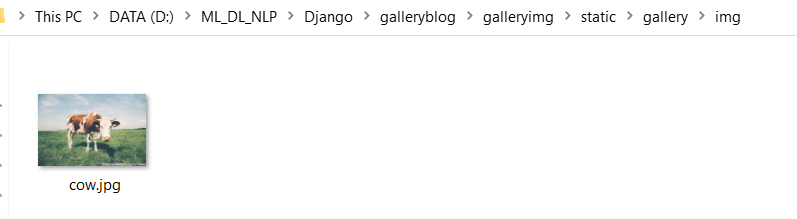
1. To store the images of our GalleryImg objects, create the following folder structure in Gallery Img applications folder:

**static/gallery/img**

i.e. create a folder “static” with a sub-folder “gallery” with a sub-folder “img”



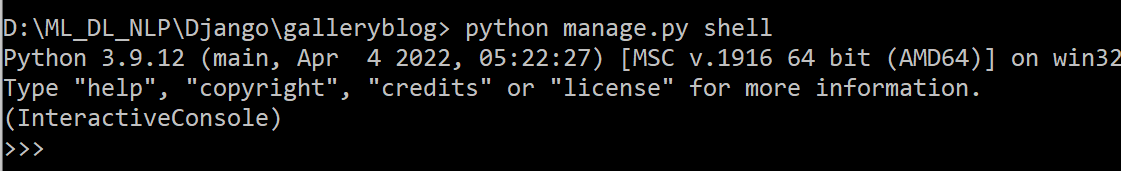
1. Download 1 image of your choice from google and store them in img folder.



1. **To test that we can create gallery objects,**

**in the normal cmd prompt, go the folder where the project galleryblog ,not the anaconda shell, run:**

D:\ML\_DL\_NLP\Django\galleryblog>python manage.py shell



**python manage.py shell**

1. Now run the following commands to create a galleryimg object, then refresh your database to see if the document entry is made.
2. In the normal command prompt itself,

import the Gallery class:

**from galleryimg.models import Gallery**

1. In the normal command prompt itself,

Create a Gallery object:

**destination1 = Gallery(title = "Cow", description="This is the image of a grazing Cow ", image="\gallery\img\cow.jpg")**

1. In the normal command prompt itself,

Run the save() command to store the data into the database:

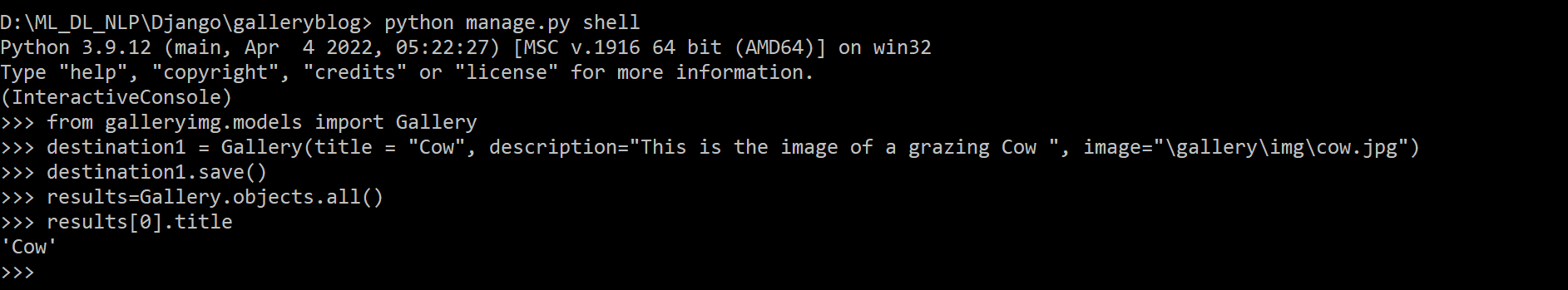
**destination1.save()**

1. In the normal command prompt itself,

To test that the data has been successfully stored, we will retrieve the data using the following commands:

**results=Gallery.objects.all()** #returns all objects

**results[0].title** # returns title of first object



**If you want you can add more images!**

# MongoDB Compass - Updations

1. Refresh mongoDB Compass to see the updates.



1. Keep Adding 2 more images in the static/gallery/img folder and access them in the same way!
2. Repeat the steps and see whether they are updated in database!
3. In the normal command prompt itself,

**Destination2 = Gallery(title = "Niagara", description="Niagara is an Amazing waterfalls ", image="\gallery\img\niagara.jpg")**

**Destination3 = Gallery(title = "Rivers", description="Rivers act as the life line of any country!", image="\gallery\img\river.jpg")**

**>>> Destination2 = Gallery(title = "Niagara", description="Niagara is an Amazing waterfalls ", image="\gallery\img\niagara.jpg")**

**>>> Destination3 = Gallery(title = "Rivers", description="Rivers act as the life line of any country!", image="\gallery\img\river.jpg")**

**>>> Destination2.save()**

**>>> Destination3.save()**

**>>> results[1].title**

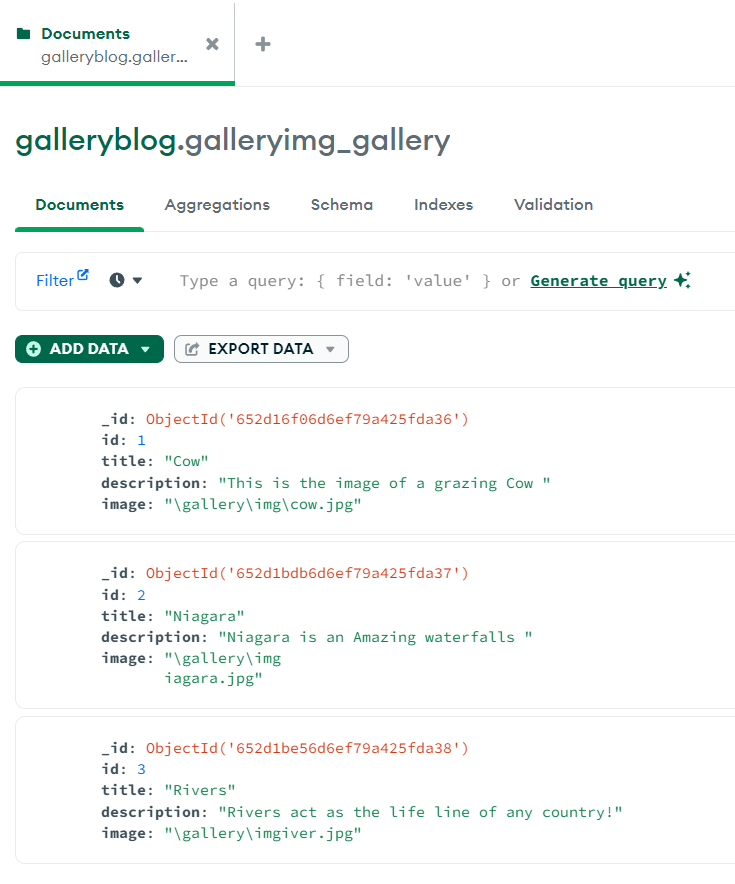
**'Niagara'**

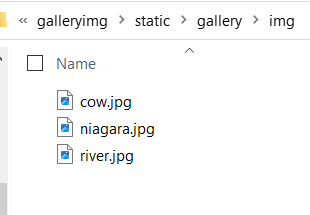
**>>> results[2].title**

**'Rivers'**

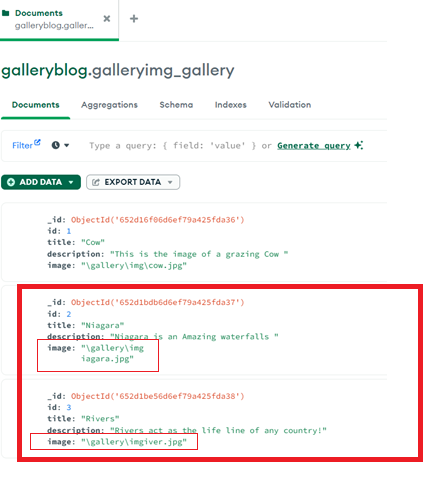
**>>>**

1. Observe the MongoDBCompass and see the updates

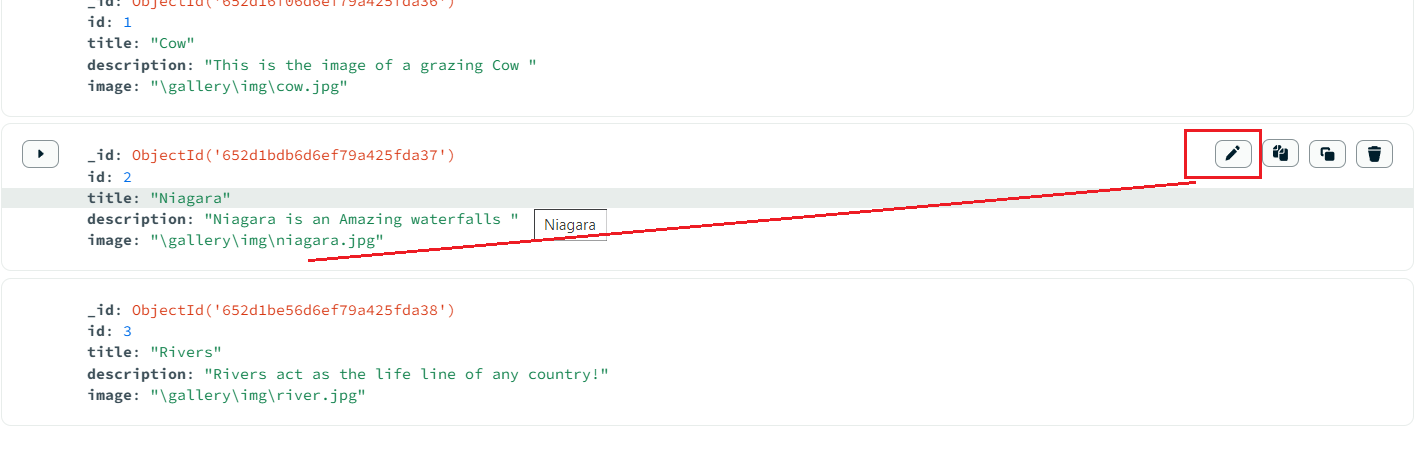


**Please note here! In the database, the image path needs to be \gallery\img\cow.jpg**

**When you note the names in the database,**

****

**It is not stored properly. So correct this immediately by clicking on the edit button against each of the documents stored.**

****

**And change it, Otherwise you will have lots of issues related to display of images in the website.**

# Create Template in settings.py

1. In the galleryimg application folder, **create a folder called “templates” with a subfolder “gallery”.** The html files will be stored in this folder.
2. To **register the templates folder in the galleryblog/settings.py file**, edit the ‘DIRS’ key’s value in the TEMPLATES variable as follows

**os.path.join(BASE\_DIR,'galleryimg/templates')**

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [os.path.join(BASE\_DIR,'galleryimg/templates')],

        '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',

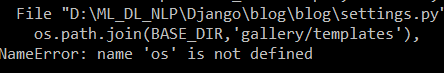
            ],

        },

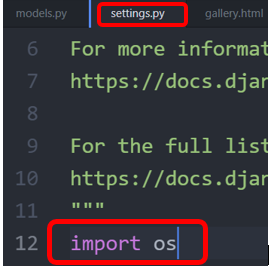
    },

]

**Please note : if you get an error:**



Add **import os** in settings.py file



# Galleryblog/settings.py static - Updations

1. Update the settings.py as below, add the static information, static in Django, helps to store images, css and js etc.

STATIC\_URL = '/static/'

MEDIA\_URL = '/media/'

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

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'static/media')

STATIC\_ROOT = 'static/gallery/img/'

STATICFILES\_DIRS =(

    os.path.join(BASE\_DIR,'/static'),

)

1. Update the galleryblog/urls.py as follows:

from django.contrib import admin

from django.urls import path,include

from galleryimg import views

from django.conf import settings

from django.conf.urls.static import static

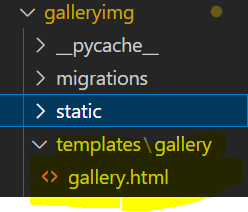
urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('galleryimg.urls')),

] + static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

# gallery.html in templates folder



1. In the templates/gallery, create a file gallery.html and include the HTML structure and display a plain text message “Making a trial with this page!”

<!DOCTYPE html>

<html lang="en" dir="ltr">

    <head>

        <meta charset="utf-8"

        <title></title>

    </head>

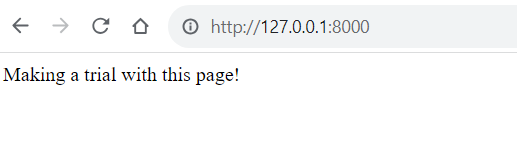
    <body>

        Making a trial with this page!

    </body>

</html>

Run the server and see the changes in the display!



1. Now, edit **the galleryimg/views.py** 🡪 gallery\_list view function to render the gallery.html page (instead of HttpResponse):

**return render(request, 'gallery/gallery.html')**

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def gallery\_list(request):

    #return HttpResponse("Are we on the right track!!!")

    return render(request,'gallery/gallery.html')

# Retrieve data from Gallery class

1. In galleryimg/views.py, gallery\_list view function, retrieve all Gallery objects and store them into gallery\_objects variable.

Import Gallery class in views module to be able to access class objects

**from galleryimg.models import Gallery**

Store gallery objects is a variable named “gallery\_objects” in the gallery\_list view

**gallery\_objects = Gallery.objects.all()**

galleryimg/views.py

from django.shortcuts import render

from django.http import HttpResponse

from galleryimg.models import Gallery

# Create your views here.

def gallery\_list(request):

    #return HttpResponse("Are we on the right track!!!")

    gallery\_objects = Gallery.objects.all()

    return render(request,'gallery/gallery.html')

1. Now, we will pass the gallery\_objects to the html page to display the details. (line 9)

Add, {‘gallery\_objects’: gallery\_objects}

NB: [Please check out the quotes while typing in atom, if u copy paste, make sure to use the right one]

Galleryimg/views.py

from django.shortcuts import render

from django.http import HttpResponse

from galleryimg.models import Gallery

# Create your views here.

def gallery\_list(request):

    #return HttpResponse("Are we on the right track!!!")

    gallery\_objects = Gallery.objects.all()

    return render(request,'gallery/gallery.html', {'gallery\_objects': gallery\_objects})

# Edit gallery.html in templates folder in gallery

1. In this section, we will use Django’s templating language:

Note: {% code logic goes here %} and {{ to print variables }}

In the gallery.html file, include the following code:

<!DOCTYPE html>

<html lang="en" dir="ltr">

    <head>

        <meta charset="utf-8">

    <title>

    </title>

    </head>

    <body>

        Making a trial with this page!

        {% for item in gallery\_objects %}

        <h4>{{item.title}}</h4>

        <p>{{item.description}}</p>

        {% endfor %}

    </body>

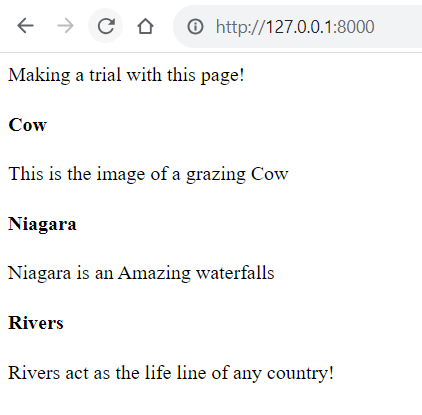
</html>

Code includes the code logic which includes a for loop, looping through the gallery\_objects and the title and description are printed.

1. Run the server to test the output:

python manage.py runserver

Observe the output



Title and description are printed!

# Adding image

1. To include images, we need to load the static files (where images are stored) of the application first. Include the following code at the top of your gallery.html page:

{% load static %}

<!DOCTYPE html>

<html lang="en" dir="ltr">

    <head>

        <meta charset="utf-8">

    <title>

    </title>

    </head>

    <body>

        Making a trial with this page!

        {% for item in gallery\_objects.all %}

        <h4>{{item.title}}</h4>

        <p>{{item.description}}</p>

        <img src="{% static item.image %}" alt="{{item.title}}" >

        {% endfor %}

    </body>

</html>

1. Run the server, python manage.py runserver

