**WORKING WITH DATABASES IN DJANGO**

**Topic 1: Creating rows (classes that we create inside models.py files) in the table (inbuilt modles class in django). Adding the rows into the databse.**

**Migration:**

1.You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.

The above message we see on running the server is because of the pre-existing several python apps in our project which can be seen in settings file.

Run 'python manage.py migrate' to apply them.

Run ‘python manage.py runserver’

2. Note: Whenever we make a new model or do some change in our existing model then it is called migration. Those changes needs to be migrated into the database. So, Migration is basically telling the database that there is a new table (class in the model) or a change in an existing table (an existing class in the model).

After making any change in the model file, we need to first inform the system about the changes

Run ‘python manage.py makemigrations’

Now, we can check about number of unapplied changes due to migration of changed model file

Run ‘python manage.py runserver’

Now, we should apply the migration of model file

Run 'python manage.py migrate'

3. migration files are created ex 0001\_initial.py

**Models:**

Inside each app that we created there is a file called models.py

Actually any design that we want to create on our page that is linked with database we create models for them. These models that we create are created in the form of a class inside the models.py file.

Inside the model.py file we can find that there is an inherited class ‘model’. This inherited model class can be thought of as a **Table** and now inside this file we are going to create classes(models) (these classes/models can be thought of as **Rows** of the table or object of the inherited model class).

The classes/models will have properties(fields). These fields can be customized.

**Inbuilt models class= TABLE**

**Classes that we create by inheriting the inbuilt model class= ROWS of the table**

Or

**Inbuilt models class = CLASS**

**Classes that we create by inheriting the inbuilt model class = Objects of the inbuilt model class**

Example:

from django.db import models

# Create your models here.

class Project(models.Model):

# here i have created a class by the name Project and inherited the models class that django has inbuilt for us. this will help us to interact with the database create table and bring things in and out

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

# we have to be very specific about the use of field (property) in django class, that is if we are going to have string as a field then we must specify what type of string ex charfield or textfield etc. Whenever we want to add a field (property) to our class then refer to the page https://docs.djangoproject.com/en/3.0/ref/models/fields/#field-types on internet and choose the field you want to use. Inside the () of a field we can have its customization

description=models.charfield(max\_length=250)

image=models.imagefield(upload\_to='portfolio/images/')

# the value of upload to should be the path of the folder where your images you want to be uploaded. Now if we will not specify a path for media files via editing the settings.py file then by default an images folder will be created automatically inside the app portfolio folder. So, if we want that the portfolio/images folder is created in a different folder (say inside a top level folder inside the base directory of our project by the name of the folder media ) then we need to specify that in the settings.py file as MEDIA\_ROOT

url=models.urlfield(blank=True)

# blank is a property which can be used with any field. It is actually used to make the use of field optional for the user. Default value of blank is False.

Note: if we are using ImageField and we have not installed then on saving the models.py file we will get this error:

ERRORS:

portfolio.Project.image: (fields.E210) Cannot use ImageField because Pillow is not installed.

HINT: Get Pillow at https://pypi.org/project/Pillow/ or run command "python -m pip install Pillow".

Therefore we need to install pypi

run command "python -m pip install Pillow".

**Now that we have created our table and rows now we want to add this to our database. So how can we add the model to database?**

Ans: Migration

Note: Whenever we make a new model or do some change in our existing model then it is called migration. Those changes needs to be migrated into the database. So, Migration is basically telling the database that there is a new table (class in the model) or a change in an existing table (an existing class in the model).

After making any change in the model file, we need to first inform the system about the changes

Run ‘python manage.py makemigrations’

Now, we can check about number of unapplied changes due to migration of changed model file

Run ‘python manage.py runserver’

We will see something like this

You have 1 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): portfolio.

Run 'python manage.py migrate' to apply them.

Now, we should apply the migration of model file

Run 'python manage.py migrate'

Now your model has been migrated to the database successfully

3. A new folder by the name migrations is created inside your app and migration files are created ex 0001\_initial.py

--------------------------------------------------------------------------------------------------------------------------------------

**CREATING ADMIN USER LOGIN AND PASSWORD**

In the **url.py** file in our project folder we have a default url path already created for admin.

urlpatterns = [

path('admin/', admin.site.urls),

]

So if we open the page localhost://8000/admin we shall land on to the admin user login page

Now we need to create the login id and password for admin(**superuser**)

Type the following command:

Python manage.py createsuperuser

Now you can login with the userid and password created

If you want to change the password type this command:

Python manage.py changepassword <username>

--------------------------------------------------------------------------------------------------------------------------------------

**ADDING PROJECT STUFF INTO THE ADMIN/SUPERUSER PROFILE**

We can do this by adding the models(classes) that we created in models.py file of the app into the admin.py file of that app

So, open the admin.py file in the project folder and edit it.

from django.contrib import admin

# Register your models here.

from .models import Project # first we need to import the model (class) that we want to add to admin login page

admin.site.register(Project) # then we add the model (class) to the admin page

Save and Reload the page.

In our example the name of the model is Project. Then we can see that the Project is added to the admin login page.

Now, when we open the Project and inside when we click on Add Project, then we can see all the fields that we created for the model (that is Project in this case). In this way we can add the models.

Things which are bold are compulsory field and the non-bold are non-compulsory fields (created by blank=True).

Provide the values of all the fields, and click SAVE. You will see that an object of the class has been created. When you will click on the object then you can see the fields and their value. However, if you will click on the image link that you provided as value to the image field then the image will not be displayed.

**How to add image if we have an image field in our model in admin page?**

A folder by the name Image was created inside our app folder portfolio and the image whose path we provided as input to the image field in above step that image is located here. It hapeend so because in our models.py file we have provided this path where the image should be uploaded.

image=models.ImageField(upload\_to='portfolio/images/')

I don’t want the image to be uploaded in this location that is inside my app folder a folder by the name image is created and the image is uploaded inside that folder.

So I delete this Image folder which is inside my app portfolio.

I want to have a common location for all my media files. I want to keep all my media files inside a folder called as media under our base directory

I will go to the settings.py file of my project and specify the root location for all the media files

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

Save the changes

You can see that a new folder by the name media has been added to our project

Go back to the admin page and the object and again choose the path of the image from your computer.

Now you can see that the image is now located inside the folder media and inside it a folder portfolio and inside it a folder images and inside it the image is located.

However the image is still not visible inside the admin page when we click on the link which is the value of the image field inside the object created. **Thus to do this we need to make some changes in our url.py file.**

We need to add a url pattern which is a list. In this url list first thing that we are going to add is static.

Static is a functionality provided by Django. We will first import this functionality inside our url.py file

Import static functionality in url.py file

from django.conf.urls.static import static

Now adding urlpattern

urlpatterns += static(<here we need to specify here first what is going to be the media url and then the media root)

go to settings.py fikle and add the url for the media files

MEDIA\_URL = '/media/' note: here instead of media we can give any name. It should be the name that while typing the url how we want the final directory in the path should look like. Whatever the name may be that name will refer to the MEDIA\_ROOT value for the location of the image.

Now, let us go back to our url.py file

Now we need to import settings.py file also into url.py file.

from django.conf import settings

Now provide the values inside the parentheses of urlpattern

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

--------------------------------------------------------------------------------------------------------------------------------------

**How to get the objects that we create in our database on to our home page?**

Inside our views.py of portfolio app we will grab the database objects and then send them forward into the template which we want to use for their display.

Step 1: open views.py of portfolio app and Since the model/class Project was created in the model.py file therefore import the class/model Project from models file

from .models import Project

Step2: Now all we need to grab the database object that we created in the database is

<Name of the class/model>.objects.all() This will grab all the objects created in the database of the model/class

Ex: Project.objects.all()

Now let us put the above into some variable say projects. We can give any name to the variable

projects=Project.objects.all()

So, now all the objects crated in the database for the class Project will be collected in the variable projects. This variable is then we are going to pass in our template home.html as dictionary.

Step3: Now I am going to pass into my template home.html a dictionary

return render (request, 'portfolio/home.html',{'key':value})

ex:

return render (request, 'portfolio/home.html',{'projects':projects})

Step4: Now inside our template home.html if I collect the value passed by the key value pair of the dictionary in this way

{{ projects }} dictionary value are captured inside {{}}

Then I wont get the output display as needed because projects is basically a list of objects and not just one value.

Therefore we need to loop through the value of projects variable. To do that we change the curly bracket to %. Loops in django {% for xyj in abc%} {%endfor%}

Note: In Django whenever you have to do a logic like iteration, etc., then you should use % inside {}

{% for project in projects %} here project is just a name we can use any name

{{ project }}

{% endfor %}

But again we find that we are not able to see the output as we wanted. That is the fields/properties of our object are still not displayed

Step5: Displaying object with its fields/properties

<h2>{{ project.title }}</h2>

<p>{{ project.description }}</p>

{{ project.image }}

{{ project.url }}

However, we find that the image is not displayed but its path is displayed.

If I make , {{ project.image.url }} then the entire path of the image is displayed. But still the image is not displayed.

Thus to display an image we need to use image tag and the source value in the image tag is to be the path of the image url.

<img src="{{ project.image.url }}""

We can also specify height and width of the image

<img src="{{ project.image.url }}"height=200 width=200>

Let us put our url field inside href tag

<a href="{{ project.url }}"> Link</a>

Since in our class/model we have decided that the objects that will be created from this model/class

May or may not have a url link. Therefore we have to incorporate this logic into our code in home.html template.

{% if project.url %}

<a href="{{ project.url }}"> Link</a>

{% endif %}

------------------------------------------------------------------------------------------------------------------------------------

**How to connect different apps to app where your home page is?**

When you have lot of different apps in a project then your home page will be in one of the app.

The landing path of the home page is provided in the urls.py file of the project sub folder.

For ex: in this project, we have our home page in portfolio app. Therefore, the path for the home page is provided in the project level file urls.py like this

from django.urls import path

from portfolio import views

urlpatterns = [

path('', views.home, name='home'),

]

However, we can forward all the url requests to a particular app if some name is provided.

For ex: we have another app in our project which has a name blog. Now any url request related to the app blog should land to a file say urls.py inside the blog folder of blog app. This we will do like this

Create a file inside the folder blog by the name urls.py. Then come back to the urls.py file at the project level sub folder.

from django.urls import path, include

from portfolio import views

urlpatterns = [

path('', views.home, name='home'),

path('blog/', include('blog.urls')), #this line means anything that comes for our website that has /blog/ and doesn’t matter what is after that it will be forwarded to our blog app.

]

Now go back to blog/urls.py file and edit it

from django.urls import path

from . import views # dot since we are inside the blog directory itself

urlpatterns = [

path('', views.all\_blogs, name='all\_blogs'),

]

Now do as you do usually, define the function all\_blogs inside the views.py file of blog app.

from django.shortcuts import render

# Create your views here.

def all\_blogs(request):

return render (request, 'blog/all\_blogs.html')

Now create a file all\_blogs.html inside blog/templates/blog folder and edit that all\_blogs.html file

Hello Blog

--------------------------------------------------------------------------------------------------------------------------------------

**Now let us create our blog app with models**

Step 1. Go to the models.py file of blog app and create your model/class

from django.db import models

# Create your models here.

class Blog(models.Model):

title=models.CharField(max\_length=200)

description=models.TextField(max\_length=250)

date=models.DateField()

Step 2. Do Migration

Python manage.py make migrations

Python manage.py migrate

Step3. Go to the admin.py file of blog folder and edit

from django.contrib import admin

# Register your models here.

from .models import Blog # first we need to import the class that we want to add to admin login page

admin.site.register(Blog) #and then we will add to the admin page

Step4. Create object/s of the model/class Blog by loging into admin page

Step5. Now in order to display the object/s that we created in the database on to the blog page we need to pull the objects. This we do inside the function that we have written for the page inside the views.py file of the blog app. The name of the function for blog page is all\_blog.

So, edit views.py fie of the blog app

from django.shortcuts import render

from .models import Blog #import your class/model

# Create your views here.

def all\_blogs(request):

blogs=Blog.objects.all()

return render (request, 'blog/all\_blogs.html', {'blogs':blogs})

Step6: Now go to the html page all\_blogs.html and write the code to display the output

**Step7: How to limit the amount of thing you want to display at a time**

We can do this inside views.py inside our function definition

1. if we want to display the most recent object

blogs=Blog.objects.order\_by(‘-date’)

2. to display the latest 5 objects

blogs=Blog.objects.order\_by(‘-date’)[:5]

------------------------------------------------------------------------------------------------------------------------------------

**Know more about databases**

There is a file called db.sqlite3 where all the information about the objects created in the database.

To read this file download a SqLiteViewer or can also read it online by uploading the file into any SqLiteViewer.

Let us use this one to upload and read our db.sqlite3 file online <https://inloop.github.io/sqlite-viewer/>

There are different type of databases which can be used with Django. The sqLite3 is the default database used in Django. In order to use any other database go to settings.py and go to the following code and make changes here accordingly. Refer online Django tutorials for more detail.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

--------------------------------------------------------------------------------------------------------------------------------------

**STATIC FILES**

Static files are different from the media files. Media files are something which can be uploaded either by the admin or user from admin panel of database or by users from webpage. But the static files are those media, document files like pdfs, and custom css files, etc., which the administrator of the website does not want to be easily accessible and changeable (like media files from database), rather he wants it to be fixed at one place.

**How to add any static file into our project?**

Step1: go to settings.py file

We have in settings.py file,

STATIC\_URL = '/static/'

Step2: create a folder with name static inside the relevant app, in this example we are going to create the static folder inside the portfolio app. And then create a folder by the name portfolio inside the static folder. The path should be like this:

portfolio\static\portfolio

Step3: add your file that is any image or pdf or etc., inside the folder portfolio created above. You can even drag and drop the file to add.

Step4: Now open the html page in the app portfolio app in this ex where you want to show this file. In this case open home.html

<img src="{% static 'portfolio/my image.jpg' %}" height=200 width=200>

Step5: whenever we use static inside a template then we have to add this line of code

{% load static %}

Step6: How to share pdf/doc, etc.?

Add the file inside static/portfolio folder. (Drag and drop)

Go to you home.html page (the page where you want to share pdf, doc, etc). Add the following code.

<a href="{% static 'portfolio/Python Concepts.docx' %}">My Resume</a>

--------------------------------------------------------------------------------------------------------------------------------------

**How to have separate pages for each object that we create in database for a class/model**

Let us understand with our example.

At present we have just one home page for our Blog app. When I open this page I am able to see all the blog objects on this page. But I want that each blog object should open in a different page.

Step1: go to the urls.py file of your blog app

Now suppose if someone want to access my third blog like blog/3 or let us say blog/number

Add this line

path('<int:blog\_id>/', views.detail, name='detail'), # here we can also put string also inplace of int

what this line says is that if anyone types www.../blog/integer then represent that integer number with the name blog\_id and pass it over to the detail.

here detail is the function that we are going to create and write in views.py file of Blog app. And also detail is going to be the name of the html page for it.

So what happens now is that whenever someone type in url …/blog then the urls.py file at the project level folder directs him to urls.py file of the blog folder and if there is any number www…./blog/3 also specified then the person is directed to detail function written in views.py file of Blog app.

Step2: go to the views.py file of the Blog app

Now here write a function with the name detail. Note that here the function not only receives the request but also the integer variable blog\_id that we have passed to it from the urls.py file.

def detail(request, blog\_id ):

return render(request, 'blog/detail.html',{'id':blog\_id}) # we are passing the blog\_id to the detail.html page

Step3: create a new detail.html file.

blog\templates\blog\detail.html

Step4: now we have been able to pass the id for the blog to html page created for that id. But now how to link that html page with the blog object in the database

We need to get one database object at a time. For that we need to add the following function in views.py file of Blog app

from django.shortcuts import render, get\_object\_or\_404

def detail(request, blog\_id ):

blog=get\_object\_or\_404(Blog, pk=blog\_id) # to the get-object\_or\_404 function we shall pass the name of the class we are looking for which in this case is Blog and the primary key(pk) that is the id of the object in the database. So, now what this function does is that it tries to grab the object whose primary key is provided via blog\_id integer variable and if it is present then it stores the result in variable blog else if no object with the particular id is present then it returns 404 page not found error.

return render(request, 'blog/detail.html',{'id':blog\_id})

now let us change accordingly the name of the parameters that we are going to pass now to our detail.html page from detail function of views.py

return render(request, 'blog/detail.html',{'blog':blog})

Step5: write the html code

For ex

{{ blog.title }}

Now similarly we can create different pages for different objects of our Portfolio class

**How to link different pages of different objects that we created above to the home page(home page of blog app in this case which is www.…/blog that is all\_blogs.html**

Suppose in this case we want that on clicking on the title of the blog which is on blog home page (all\_blogs.html that is www…/blog) one is directed to the particular blog

Step1: go to the home page /landing page of Blog app which is all\_blog.html and add a <a> tag with the title field of the object.

Note: whenever we are referring to a url which has an id with it how do we specify that in href

{% for blog in blogs %}

<a href="{% url 'detail' blog.id %}"><h2>{{ blog.title }}</h2></a>

<p>{{ blog.description }}</p>

<h5>{{ blog.date }}</h5>

{% endfor %}

Note: we can stop here but follow the following practice to achieve the linking task .

Inside the urls.py file of your blog app add a variable app\_name and set it equal to the name of your app which in this case is blog.

app\_name = ‘blog’

Now go to your home page of blog app which is all\_blogs.html and do the following change

<a href="{% url 'blog:detail' blog.id %}"><h2>{{ blog.title }}</h2></a>

Why did we use this app\_name variable for our blog app ?

Suppose we have several apps in our project and many of them have a detail.html page. Then Django will pick up the first one that it will find which may not be the page that you wanted. By specifying the app name in urls.py file of an app then any time we want to refer(a tag reference) to any of the urls specified in that app’s url.py file we have to say app name: particular url in the reference. Ex

<a href="{% url 'blog:detail' blog.id %}></a>

--------------------------------------------------------------------------------------------------------------------------------------

**Decorating template pages**

**1.Add a link to your blog app home page in portfolio app home page that is home.html file of portfolio app**

<br><a href="{% url 'blog:all\_blogs' %}">Sandeep's Blog</a> #remeber that now as you have given the app\_name = blog therefore you have to use blog:page url name that is for all\_blogs page you have to refer it like blog:all\_blogs

**2.How to show on the blog app home page that is on all\_blogs page the number of blogs I have written?**

Go to your all\_blogs.html file and write the following code

<h2>Sandeep has written {{ blogs.count }} blogs</h2>

This line of code however will not difference between singular and plural that is even when you have 1 blog count then also it will display Sandeep has written 1 blogs and not 1 blog. So in order to achieve the difference in **singular and plural count modify** the above line of code as follows:

<h2>Sandeep has written {{ blogs.count }} blog{{ blogs.count|pluralize }}</h2>

If you have limited the number of blog objects that are to be displayed at a given time on the all\_blogs page

blogs=Blog.objects.order\_by('-date')[:5]

then the count will display only that number of blogs. So, if you want to display the count of all blog objects despite limiting the number of blog object displayed on the all\_blogs page we can do like this

In views.py file inside the all\_blogs function

blogs\_count= Blog.objects.count

blogs=Blog.objects.order\_by('-date')[:5]

and then pass blogs\_count to the all\_blogs.html page through the render function

**3. How to have a customized date format instead of the default?**

For different date formats refer Django template date format on internet <https://docs.djangoproject.com/en/3.0/ref/templates/builtins/>

Now in your all\_blogs.html file edit the date string as follows or any other format as you like

<h5>{{ blog.date|date:'M d Y' }}</h5>

Or for all in upper case

<h5>{{ blog.date|date:'M d Y'|upper }}</h5>

**4. how to cap the amount of words that are displayed in description (a character field) ?**

Go to your all\_blogs.html file and edit the description field as follows

<p>{{ blog.description|truncatechars:100 }}</p>

**5. if you want to display the effect of html tags that you have used in your fields whichyou have created for your objects then you have to make this change in the field. Ex:**

Suppose we want that to use html tags while writing in the description field (a char field) for our blog object then

Go to all\_blogs.html file and make the following change in the description field

<p>{{ blog.description|safe}}</p>

And if you don’t want to display the effect of use of html tags then

<p>{{ blog.description|striptags}}</p>

-------------------------------------------------------------------------------------------------------------------------------

**Proper naming of objects instead of project1, project2, blog1, blog2, etc. in database**

For Project class/model object go to models.py file of portfolio app and add this new function

This function returns the default name of the object whenever someone looks for the object

Suppose we want to return title field of the object as the default name of the object in the database whenever the object is referred by someone

from django.db import models

# Create your models here.

class Project(models.Model):

title=models.CharField(max\_length=100

description=models.CharField(max\_length=250)

image=models.ImageField(upload\_to='portfolio/images/')

url=models.URLField(blank=True)

def \_\_str\_\_(self):

return self.title

we can further customize it by adding few more things like

def \_\_str\_\_(self):

return self.title + 'Sandeep'

Now we will do a similar thing for blog app

Go to models.py file of the blog app and add the following function

from django.db import models

# Create your models here.

class Blog(models.Model):

title=models.CharField(max\_length=200)

description=models.TextField(max\_length=250)

date=models.DateField()

def \_\_str\_\_(self):

return self.title

Note: when we add a function to the models.py then we don’t need to make any migration because the functions don’t enter the database.

-------------------------------------------------------------------------------------------------------------------------------

**How to make your website look good? ---- Base Template & extending Base Template Concept**

This concept makes the bootstrap and css easier for website

Step1: go to the bootstrap webpage on internet <https://getbootstrap.com/>

And click on Get Started button.

Go to the Starter Template and copy the code

Step2: Now we can paste the Starter Template code in the html file of our project for which we want to have a good look. Then after pasting the code on the top of the file we can copy the code that we have already written in the same html file and paste it inside the body tag (where hello world is written) of the Starter Template code. Delete the already written code once pasted inside the body of the Starter template code.

But if we have several pages in our website and we want to have a common look for all of them then copying the Starter template code on every html file individually and then copying the corresponding code of the page inside the body tag of the Starter Template is very tedious and repetitive.

Now suppose I also want to have a navigation bar on my web pages then again copy the code for the navigation bar from the bootstrap webpage and paste it inside the body tag of the Starter template even before the start of your code that you copied earlier.

BUT WE WILL NOT DO LIKE THIS INSTEAD WE WILL CREATE A BASE TEMPLATE AND EXTEND IT TO ALL OTHER PAGES

Step1: create a file by the name base.html (you can have any name) inside the portfolio folder of the template folder of the portfolio app.

portfolio\templates\portfolio\base.html

Step2: Paste the Starter template code inside the base.html file

Step3: Paste the navigation bar code inside the body of the Starter template code

Step4: **Concept of block**

Now you want to include all webpage code inside the body of the starter template code but after the nav tag. Write the following block code after the end of nav tag </nav>

{% block content %} # instead of content we can give any name to our block

{% endblock %}

Delete the hello world h1 tag

Step5: Now go to your html file of the page that you want to include inside the base.html template and write the following code

Suppose we do this for home.html page

{% extends 'portfolio/base.html' %}

{% block content %}

{% load static %}

<img src="{% static 'portfolio/my image.jpg' %}" height=200 width=200>

<br><a href="{% static 'portfolio/Python Concepts.docx' %}">My Resume</a>

<h1>Sandeep's Portfolio</h1>

{% for project in projects %}

<h2>{{ project.title }}</h2>

<p>{{ project.description }}</p>

<img src="{{ project.image.url }}"height=200 width=200>

{% if project.url %}

<br><a href="{{ project.url }}"> My Blog</a>

<br><a href="{% url 'blog:all\_blogs' %}">Sandeep's Blog</a>

{% endif %}

{% endfor %}

{% endblock %}

Step6: Now we can apply this step5 to all those pages on which we want to apply the base.html template.

We can also create different base templates for different group of pages in a single website and can have different looks for different pages.