Django

Django was created in 2003 at the Lawrence journal world newspaper.

-When encountering with Django we should always create a virtual environment or an ‘venv’.virtual environment allows us to use a virtual installation of python and its packages on our computer

-Django is a open source free web framework , which is used to develop web application written in python and it follows MVT(model view template ) design pattern

The top features of Django are fast , fully loaded , security , scalability , versatile .

The command used to install django

Pip install django

To check the current version of the django

Py -m django version

I have used anaconda for my learning so to create a virtual environment the below is the command

conda create –name environment name

to activate the environment

activate environment name

to deactivate environment

deactivate environment

To create a new django project below is the command.

Django-admin startproject project name.

The package will install many python files.

There are different files which are created .

1. \_\_init\_\_.py file : This is a blank python file but due to its special name it lets the python know as it is a directory which can be treated as a package
2. Settings.py : This is the file where all the project settings are stored.
3. This is a project script that will store all the URL pattern for your project.Basically different pages of your webapplication.
4. Wsgi.py : this is python script that acts as the web server gateway interference. It will later help us to deploy our webapplication to production.
5. Manage.py : this file is where we use lot of commands, it is associated with many commands to build our web app.

A Django project is a collection of application and configuration , when combined makes up a full web application.

Django app creation : The django app is created using the following command.

python manage.py startapp name of the app(new name to be used to create that app)

when we create a new app , we get different files they are

\_\_init\_\_ , settings.py , wsgi.py manage.py , models.py , tests.py ,views.py , apps.py ,admin.py

Admin.py : you can register your models here which django will then use them with djanogs admin interface.

Apps.py ; this provides a place for the application specific configuration.

Models.py : this is the place where we store the applications data model.

Tests.py : here we can store test function to test your code.

Views.py This where you have functions that handle requests and return responses.

Migrations:this is a directory which stores database specific information as it relates to the models.

-To map views with the url to display the html content the following steps are used

Make a entry in the settings.url , in the installed apps , add the app which you are building.

Make a entry in the views.py by importing the http module using

From django.http import HttpRespone and define the function

def index(request(this variable can be anything):

return HttpResponse(‘the html content or string’)

make an entry in the urls.py , by importing the url

from appname import views

in the url patterns enter the new view created

path(‘ ‘ , views.index , name=’index’)

-To map views with the url using the include method , the following steps are used

Make a entry in the settings.url , in the installed apps , add the app which you are building.

Make a entry in the views.py by importing the http module using

Make a entry in the views.py by importing the http module using

From django.http import HttpRespone and define the function

def index(request(this variable can be anything):

return HttpResponse(‘the html content or string’)

create a new file in the app folder as urls.py

make a entry as follows

from django.urls import include,path

from first\_app import views

urlpatterns = [

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

]

In the urls.py make a entry as

from django.contrib import admin

from django.urls import include,path

urlpatterns = [

path('first\_app/', include('first\_app.urls')),

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

]

Django templates:

These templates contains the static parts of an html page .

The templates allows to create a whole html view to the users .

In order to use the templates , create a new folder templates in the project. In the settings.py create a path by joining the existing directory to the template directory. The command is below

TEMPLATES\_DIR = Os.path.join(BASE\_DIR , ‘templates’)

In the templates folder create a folder for the app , and now enter in that folder create a new html file and enter the html in that file and there are what is known as template variable which are inserted inbetween html , they are written as {{ variable name}} eg : {[insert\_me }}

In the views.py of the app folder , display the output as

Def index(request):

My\_dict = {‘insert\_me ‘ : ‘html content to be added in between the actual html’}

Return render(request , ‘index.html(this is the file name created in the template folder)’ , context = above dictionary variable name)

-Inserting static media files.

In order to create a new static images to a web application , we need to create a new folder as static and then add a folder as images in the static folder.

Add the image into that folder

In the settings.py file , create a new variable with path to the static images.

STATIC\_DIR = Os.path.join(BASE\_DIR , ‘static’)

Then in the setting at the end add a new list as

Staticfiles\_DIRS= [ (add the above variable created)STATIC\_DIR,]

We can view this page using the url 127.0.0.1/static/images/imagename.jpg

Inorder to display the static images on to the index page.

In the html insert a template tag {% load staticfiles %} after the doctype in the html

In the body add a img tag as below

<img src =’{% static ‘img/image.jpg’%} , alt= ‘any message if the image does not loads up’>

Not only the images , we can also include the static files such as CSS files etc

When we use css files we need to style the page and then link in the html as

<link rel = stylesheet , href = ‘{% static ‘css/styles.css’ %} ‘ />

Model

This is the class attribute which is used to create a database using the model class attribute which is already present in the python module

From django.db import models

We can use oops concept of python to create a database

Eg of creating tables in the database.

class Topic(models.Model):

top\_name = models.CharField(max\_length=264,unique=True)

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

return self.top\_name

The above eg creates a new table Topic ,it is similar to the create type of sql , the string method is used to print out the result. We can also use the foreignkey as well to provide the relationship

class Webpage(models.Model):

topic = models.ForeignKey(Topic, on\_delete=models.CASCADE )

name = models.CharField(max\_length= 264,unique=True)

url = models.URLField(unique=True)

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

return self.name

after creating the database we have to migrate this data base using the following command

python manage.py migrate

we have to map the databases to the app which we have built using the below command

python manage.py makemigrations appname

after the above command , we have migrate database again

python manage.py migrate.

we can also verify the database in shell using

python manage.py shell

we cannot use the shell to make entries in the database everytime

so we have to use the admin.py file to get the power of admin permissions

in the admin.py module of the appfile we have to register the tables which are created

from app.models(the app which I have created) import table name

eg : from first\_app.models import AccessRecord,Topic,Webpage

we have to register all these table to the admin separately using

admin.site.register(tablename)

eg: admin.site.register(AccessRecord)

we have to create a superuser for the admin using the command

python manage.py createsuperuser

And provide the username and passoword and run the server

If we go to the page 127.0.0.1/admim/login

Here we will have all the access to creating users and creating entries in the database as well.

-Find the example of using the faker module , which generates the fake data in the project.

-Please see the video of connecting all the three fields to form a website

Forms in Django

We can use the form similar to the which we use in the html

We have to create a new file in the app as forms.html and import the forms, it is similar to the modules .

Using the inbuilt classes of the forms to create the forms

Eg: class FormName(name of the form)(forms.Form):

Name = forms.CharField(max\_length = 265 )(form fields)

We can import the forms module to make a view of it .And mapping that to the url

We have to create a new html file for the forms.

In the forms html file we have to include the csrf token to accept the forms while using the post method.This secures the HTTP Post action that is initiated on the subsequent submission of the form

We can print out the objects which is created for the forms through validating the forms

Please verify the project for more details.

For the submitting the form there may be empty fields or there may be a bot(automated scripting programs , which automatically fills the forms ) , we can use django to validate them.

We use hidden feature from the Django to detect the bots , and it will not be displayed in the html , we have to inspect inorder to validate whether it is bot or not, so we should include another field in the forms which is not visible to the normal human and visible when we inspect the page , so the bots will this form input as well and we will get to know that the bot is filling the data.so the extra form field should written as below

Botcatcher = forms.Charield(required=False , widget = forms.HiddenIput)

If we want to check the validation of a form i.e whether it is a bot or human , we can write a method which is below

Here clean is the keyword

def clean\_botcatcher(self):

botcatcher = self.cleaned\_data['botcatcher']

if len(botcatcher) > 0:

raise forms.ValidationError('GOTTCHA BOT!')

return botcatcher

But this cannot be done while creating a big webpage , we can use django core module , which has inbuilt validators , we can insert this into any object to check for the validation.

Form-Models: we can create forms with reference to the models so that when we submit the button after filling the form, we can make a entry in the database.

It is similar form template but with additional fields

From Django import forms

Form appname import models name

Class formname(forms.Form):

We can add the validations here

Class meta:

Models = modelsname

Fields = ‘\_\_all\_\_””(to include all the fields)

Or in the fields = (‘field’ , ‘field) to include the specific fields

Or in the exclude = [‘field’ , ‘field’] to exclude the specific fields (rather than fields we should take the parameter as exclude)

We can also create our custom validators

-Relative urls with templates.

It is used to link one link in the other page. This is nothing but providing link to other page from a webpage.

This is done using a url template tagging

The procedure to do this is

In the app url page , create a app\_name variable and assign a string as the appname , and it should be app name

Eg ; app\_name = ‘app name created’

And do the url mapping for the variable

And in the html page which I want to link

Use a anchor tag like

<a href="{% url 'appname : link name ' %}"></a>

Eg: <a href="{% url 'secondapp:other' %}">This is the Other page</a>

-Template inheritance:

If more than one page uses the same html content, we can inherit from one html page to the other

In the master html page , write the common html content that is required for many other html pages

After the common html content , we need to write a django template as

{% block body\_block %}

{% end block %}

In the other html page we need to extend from the master html page as

{% extends ‘path of the master html page’ %}

Again in the other html page we need to write the above django template

{% block body\_block %}

{% endblock %}

Inside that template we need to write the html content which is unique to that page , and anything outside that template will be inherited from the master html page.

-Filters

These filters are used to make upper cases of some text in one html page and not in the other html page. So it is used to customize your pages of html .

Syntax : {{template variable | filtertype }}

We can also create custom filters , we need to create a folder in the app folder and build a method that can be further used as the filter

We can also create a custom filter as well.

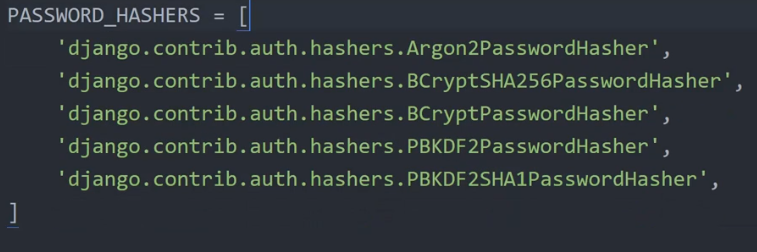
Passwords

This is used to use the password hash function while saving the passwords.

There are two important security libraries which provides the hash function, they are bcrypt , argon2

We should install them to use it.

We should create a variable as PASSWORD\_HASHERS = [ ‘django.contrib.auth.hashers.PBKDF2Pa sswordHasher (this is the Django built in hasher )



Please find the above hash methods to add into the variable , this acts as fall back as well , if one is not applying then other will be working. So it is kind of back up.

-In the auth\_password\_validators , we can add the additional category , in order to satisfy the password.

Media folder :

This in the application is a folder where, when a user imports any pics, this should be present in this folder .To set up the media folder

Similar to the template and static we need to join with the base directory.

MEDIA\_DIR = os.path.join(BASE\_DIR , ‘media’)

And at the bottom

MEDIA\_ROOT = MEDIA\_DIR

MEDIA\_URL = ‘/media/’

In the admin page , we observer two models User and Group model.

This field already has the built in fields , but we can add in extra fields in it , we can add additional fields like profile pic extra, this can be done by addressing that model and adding additional fields by creating a new model

Class based view approach

The first part is in office PC

We have what is known as list view and detailed view.

Here we import ListView and DetailView from Django.views.generic module

And we can create a class similar to the templatView and we can inject the models created

Class classname(ListView):

Model = models.ListView(name of the model)

Similarly for detail view

Class classname(DetailView):

Model = models.DetailView(name of the model)

When we create a list view or detail view , the list view automatically creates a list context dictionary as modelname\_list and detail view creates a context dictionary as modelname in lower case.

While setting up the detailed view , while referencing to the primarykey the views should be

Path(‘<int:pk>’ , regular view)

CRUD applications :

Create , retrieve , update , delete

For using these views we need to import the respective view from the views file

createView , retrieveview , updateView , deleteview

In Django we can import the default models of the user which has many fields for login or sign up

We should import auth from Django.contrib

This auth default has the view , so there is no need to set the views

In the models we should inherit from auth.models.users

For the forms

We should import get\_user\_model

And we should equal that to the model

Eg:

from django.contrib.auth import get\_user\_model

from django.contrib.auth.forms import UserCreationForm

class UserCreateForm(UserCreationForm):

class Meta:

fields = ('username', 'email' , 'password1', 'password2')

model = get\_user\_model()

def \_\_init\_\_(self, \*args,\*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.fields['username'].label = 'Display Name'

self.fields['email'].label = 'Email Addrese'

the \_\_init\_\_ method is to provide the customized label, it is optional

so while creating urls for this

we must import it through

import Django.contrib.auth import views as auth\_views

we have to do aliasing because the views of our application is also views so , but Django provides the default views for the forms , we no need to declare the views for login

in the url pattern

urlpattern = [

path('login/' , auth\_views.LoginView.as\_view(template\_name='accounts/login.html'), name='login')

]

We can import misaka in Django , this is used to get the markup for the fields

We have include which is used inside the middle of the html it is similar to extends but we are extends is used to extend a certain length of the content , but include , includes a specific from the other html and it can be used in the middle.

We can add a tabular inline method which adds the contents to the main model content so that we can edit both in one model content