Django

Flask

↳ Uses MVC architecture

↓

Model View Controller → to pass controls.

↙ we write all the utilities (or) core application code

↘ the html page code.

2, Used to build small application.
(1 page)

3, Non-Monolithic

Django

↳ Uses MVT architecture

↓

Model View Template

↙ different types of templates can be used.

2, Used to build scalable applications.
(Multiple pages)

3, It is Monolithic

↙ defined structure of project
(we need to keep the files in some prescribed structure)

⇒ standard project directory structure.

** → By default we will get admin panel.

** → We get multiple database connectors.

⟶ It follows DRY - Don't Repeat yourself.

** ① Open Anaconda prompt in Admin mode.

```
Administrator: Anaconda Prompt (anaconda3) - "C:\Users\KrishnaAleti\anaconda3\condabin\conda.bat"  activate djang

(base) C:\WINDOWS\system32>conda create -n django_env
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\KrishnaAleti\anaconda3\envs\django_env



Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate django_env
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:\WINDOWS\system32>conda activate django_env

(django_env) C:\WINDOWS\system32>
```

② creating new environment to work for django project.
(conda create -n django_env)

③ activating the newly created environment

④ installing django
(python -m pip install django)

```
(base) C:\WINDOWS\system32>conda activate django_env

(django_env) C:\WINDOWS\system32>python -m pip install django
Collecting django
  Using cached Django-4.0.4-py3-none-any.whl (8.0 MB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting tzdata
  Using cached tzdata-2022.1-py2.py3-none-any.whl (339 kB)
Collecting asgiref<4,>=3.4.1
  Using cached asgiref-3.5.0-py3-none-any.whl (22 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.0 django-4.0.4 sqlparse-0.4.2 tzdata-2022.1
WARNING: You are using pip version 21.1.3; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\KrishnaAleti\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upgrade pip' command.

(django_env) C:\WINDOWS\system32>
```

```
(env_django) C:\WINDOWS\system32>mkdir project

(env_django) C:\WINDOWS\system32>cd project

(env_django) C:\Windows\System32\project>django-admin startproject django_project

(env_django) C:\Windows\System32\project>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is 0CF6-84EF

 Directory of C:\Windows\System32\project

11-04-2022  04:04 PM    <DIR>          .
11-04-2022  04:04 PM    <DIR>          ..
11-04-2022  04:04 PM    <DIR>          django_project
               0 File(s)              0 bytes
               3 Dir(s)  161,304,285,184 bytes free

(env_django) C:\Windows\System32\project>
```

To create a project.

(5) (django-admin startproject project_name)

project created.

```
(base) C:\Windows\System32\project>D:

(base) D:\>cd D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django

(base) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>conda create -n dj_env
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\KrishnaAleti\anaconda3\envs\dj_env



Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate dj_env
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>conda activate dj_env

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>install django
'install' is not recognized as an internal or external command,
operable program or batch file.

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>python -m pip install django
Requirement already satisfied: django in c:\users\krishnaaleti\appdata\local\programs\python\python39\lib\site-packages (4.0.4)
Requirement already satisfied: tzdata in c:\users\krishnaaleti\appdata\local\programs\python\python39\lib\site-packages (from django) (2022.1)
Requirement already satisfied: asgiref<4,>=3.4.1 in c:\users\krishnaaleti\appdata\local\programs\python\python39\lib\site-packages (from django) (3.5.0)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\krishnaaleti\appdata\local\programs\python\python39\lib\site-packages (from django) (0.4.2)
WARNING: You are using pip version 21.1.3; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\KrishnaAleti\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upgrade pip' command.

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>
```

```
(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>django-admin startproject django_project

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>mkdir
The syntax of the command is incorrect.

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>dir
 Volume in drive D is Data
 Volume Serial Number is 849E-AE1F

 Directory of D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django

11-04-2022  04:11 PM    <DIR>          .
11-04-2022  03:45 PM    <DIR>          ..
11-04-2022  02:12 PM    <DIR>          23. Django_iNeuron
11-04-2022  04:11 PM    <DIR>          django_project
               0 File(s)              0 bytes
               4 Dir(s)  945,152,630,784 bytes free

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>
```

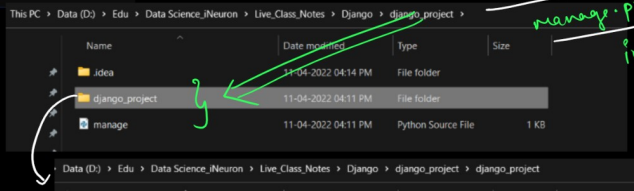
project created.

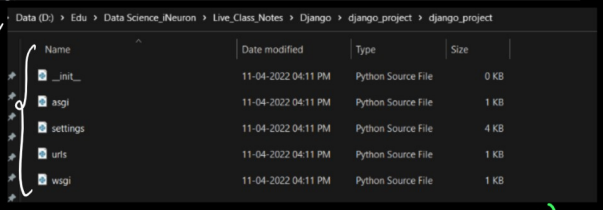
⇒ (django-admin startproject project_name)

folder with project_name will be created.

↳ This will have other folder with name same as project name & manage.py in it.

```
(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django\django_project>dir
 Volume in drive D is Data
 Volume Serial Number is 849E-AE1F

 Directory of D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django\django_project

11-04-2022  04:13 PM    <DIR>          .
11-04-2022  04:11 PM    <DIR>          ..
11-04-2022  04:14 PM    <DIR>          .idea
11-04-2022  04:11 PM    <DIR>          django_project
11-04-2022  04:11 PM               692 manage.py
               1 File(s)            692 bytes
               4 Dir(s)  945,150,267,392 bytes free

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django\django_project>
```





(6) to run the project (python manage.py runserver)

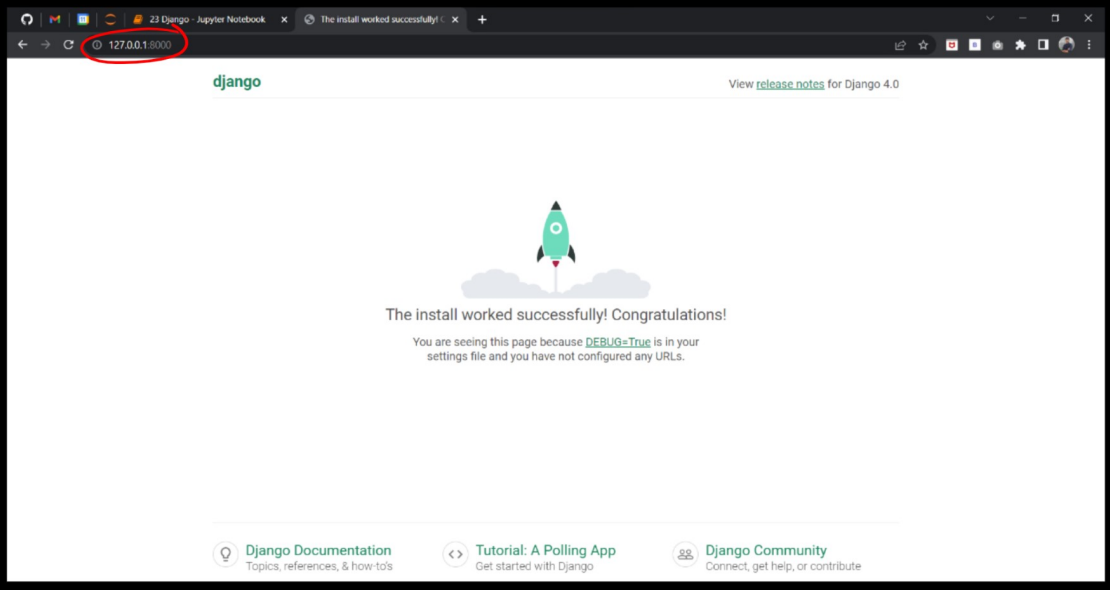
```
(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>python manage.py server
python: can't open file 'D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django\manage.py': [Errno 2] No such file or directory

(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django>
```

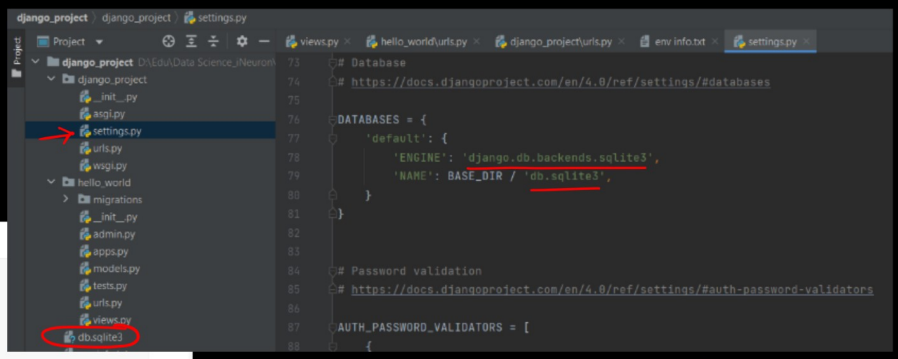
Error

Enter into the project folder & then run the server/project.

```
(dj_env) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Django\django_project>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 11, 2022 - 16:19:25
Django version 4.0.4, using settings 'django_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Server url





Let's look at what **startproject** created:

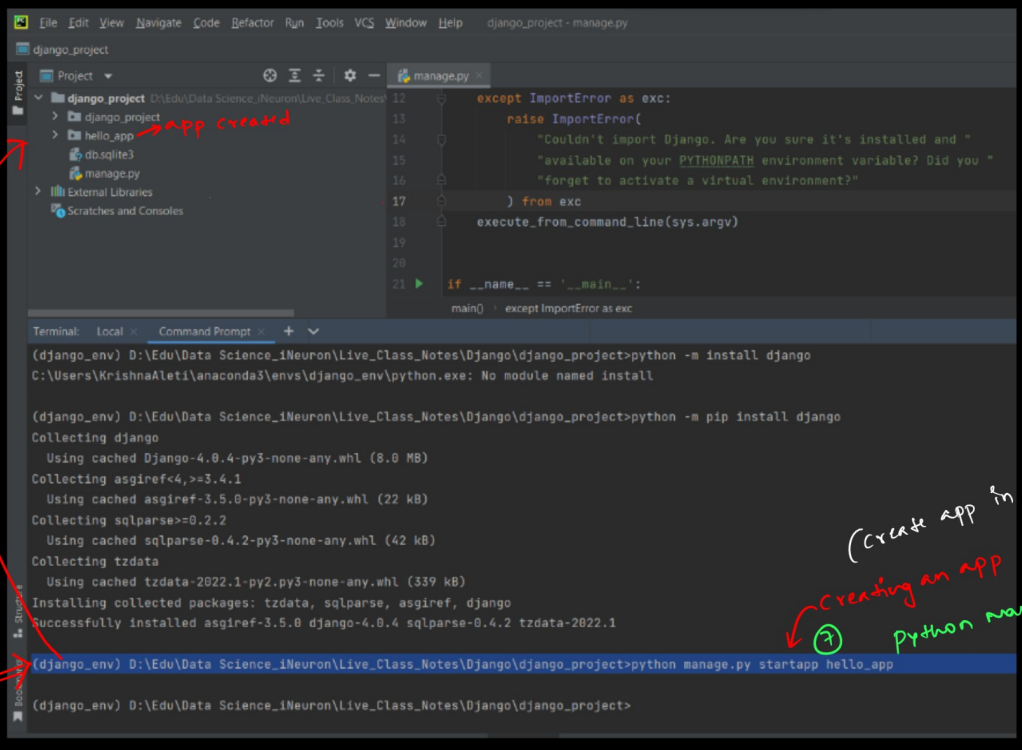
```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

These files are:

- The outer **mysite/** root directory is a container for your project. Its name doesn't matter to Django; you can rename it to anything you like.
- **manage.py**: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about **manage.py** in django-admin and manage.py.
- The inner **mysite/** directory is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. **mysite.urls**).
- **mysite/__init__.py**: An empty file that tells Python that this directory should be considered a Python package. If you're a Python beginner, read more about packages in the official Python docs.
- **mysite/settings.py**: Settings/configuration for this Django project. Django settings will tell you all about how settings work.
- **mysite/urls.py**: The URL declarations for this Django project; a "table of contents" of your Django-powered site. You can read more about URLs in URL dispatcher.
- **mysite/asgi.py**: An entry-point for ASGI-compatible web servers to serve your project. See How to deploy with ASGI for more details.
- **mysite/wsgi.py**: An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details.

When we create the project, in the folder there will be Settings.py & in this Settings.py all the major details will be present. In it by default, the database will be Sqlite3.

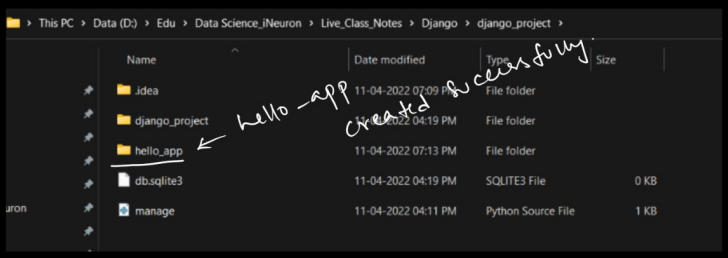
⇒ db.Sqlite3 will be created in the folder.



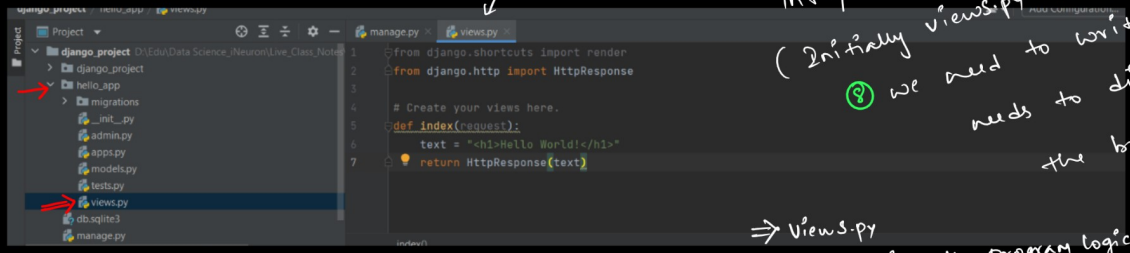
(Create app in the django_project folder)

Creating an app

python manage.py startapp app_name



hello_app created successfully.



view is to show the info/text in browser.

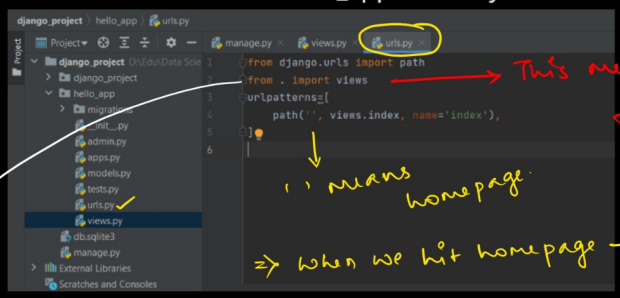
(Initially views.py will be empty, we need to write whatever needs to displayed in the browser)

⇒ views.py contains the program logics.

hello_app > views.py

⇒ To call the view, we need to map it to a URL - and for this we need a URLconf.

To create a URLconf in the hello_app directory, create a file called urls.py and paste code written as below:-

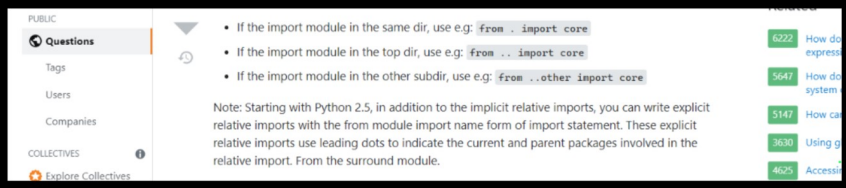


This means from the current directory import views.

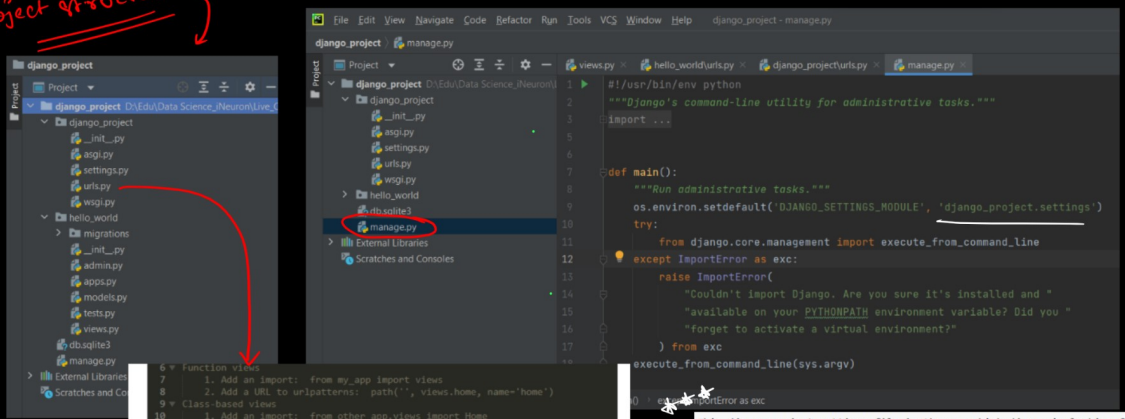
'' means homepage.

⇒ when we hit homepage → views.index will be called

The single dot is a convention from command line applications. It means the current directory. In terms of Django it stands for the directory/module the current file is on

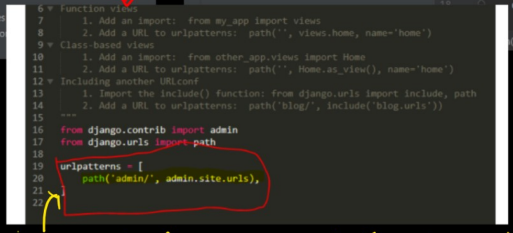


Project structure :



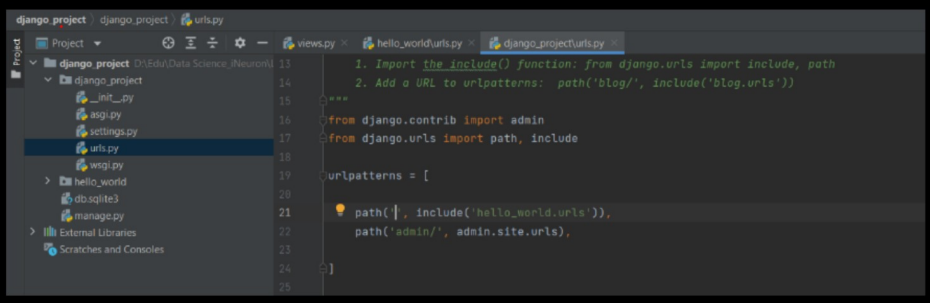
This django_project.settings file is the one which django is looking for for settings, you can define another file as well or you can change the file as per different enviornments different settings file.

→ The project settings are defined in the manage.py file.

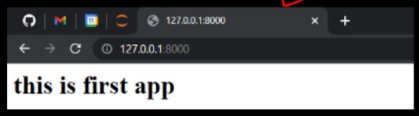


⇒ Here in the highlighted section, we can define the all the applications of the entire project, so that django will route to the apps defined in this urlspatterns (urls.py of the project)

The next step is to point the root URLconf at the django_project.urls main module. In django_project/urls.py, add an import for django.urls.include and insert an include() in the urlpatterns list, so django will look into hello_world application and in that urls we have index defined so it is able to call that view in this way:



⇒ run the server → python manage.py runserver



Steps to create a project > create an app > run the app:

1, open Anaconda prompt in Admin mode (can use an other IDE - but in admin mode)

2, create a virtual environment for the django project → conda create -n django_env
(env. name)

3, activate the newly created env. → conda activate django_env

4, Installing django in the virtual env. → python -m pip install django

5, create a project → django-admin startproject django_project
(project name)

→ A folder with the project_name will be created.
↳ Inside it another folder with project_name & manage.py will be created.

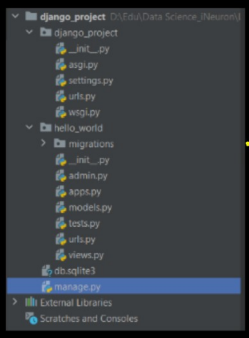
6, We need to enter into the folder in which manage.py is there and run server
(outer)
→ python manage.py runserver

→ Open the browser & enter the server url => http://127.0.0.1:8000/

⇒ A page with 'The install worked successfully! Congratulations!" will be displayed.

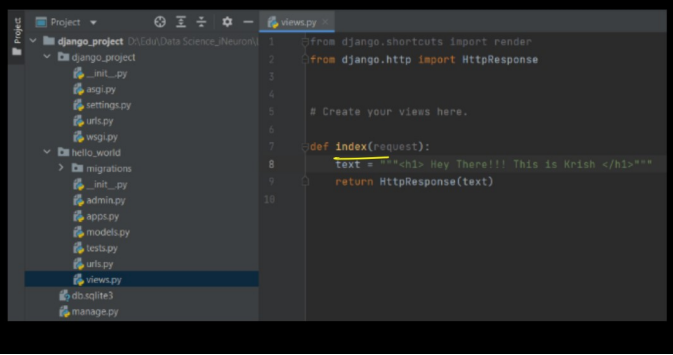
7, Now create our own app in the django_project folder → python manage.py startapp hello_world
(outer folder)
(app name)

→ The app (hello_world | hello_app) gets created successfully.



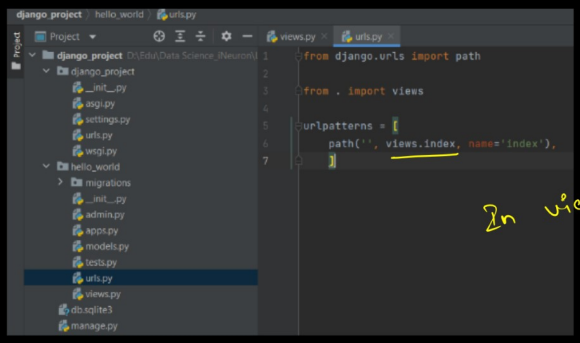
← file structure (we always need to follow this structure only)

8, In the views.py (of the app: hello_world), we need to write whatever we want to display in the browser.



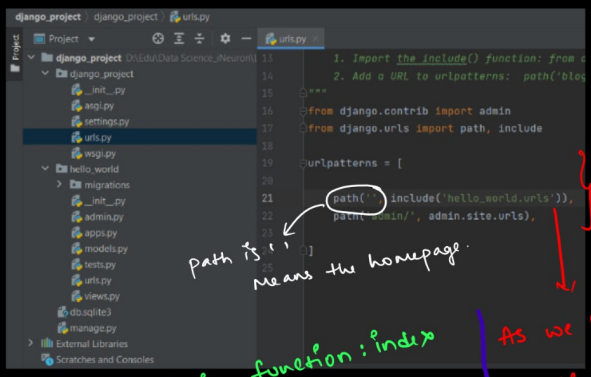
→ Now to call this view, we need to map it to a url — and for this we need a URLconfig.

9, To create URLconfig in the hello_world directory, create a file called urls.py & write the following code:



In views file, we had defined the 'index' function.

10, The next step is to point the root URLconfig at the django-project.urls main module.



path is '' means the homepage.

11, run the server

django will route things from the apps defined in the url patterns.

Flow:

⇒ Write the logic/text in function: index of views.py (of app) → link it to the urls.py (of app) by giving the path views.index

link it to urls.py (of project) by including the path as app_name.urls

⇒ when we run the server, whatever written in the views.py will be displayed.

⇒ django will look into the hello-world application & its url: → in the urls file of hello-world as we had included 'hello-world.urls'

we have views.index → This call the index function defined in the views & the index() gets executed.

the django will route to the paths defined in the urlpatterns of the urls.py (of the project)

↳ 'app.urls'

then routes to path defined in the urlpatterns of urls.py (of app) → views.index function → Reaches views.py

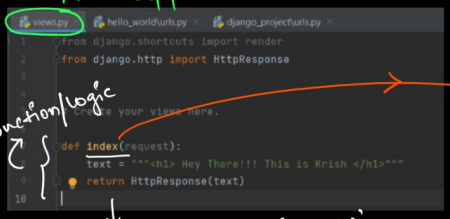
index() executes

whatever written in the function will be displayed in the browser.

→ run the server ⇒ python manage.py runserver

Django files linking / flow:

views.py of app

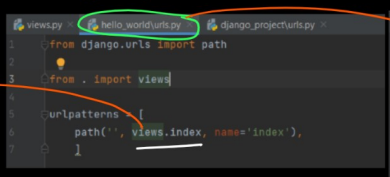


function/logic

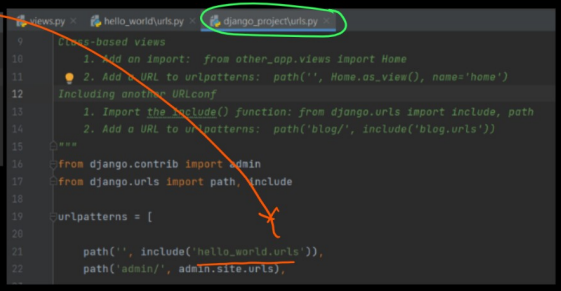
we can call this 'index()' in other modules as views.index()

because the index() function is in views.py file.

urls.py of app



urls.py of project



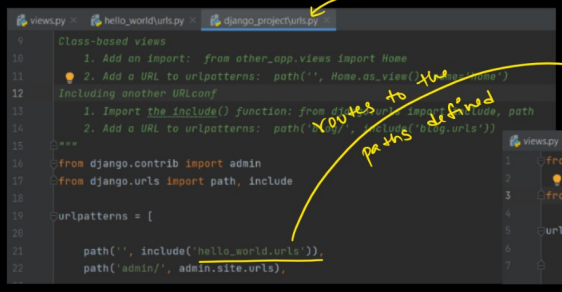
⇒ when we run the server — python manage.py runserver

↙

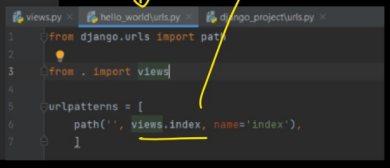
the django looks into the project's urls.py file & routes to the paths/urls defined in the urlpatterns of urls.py (of project)

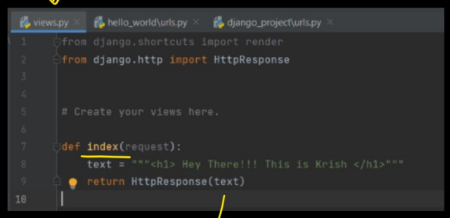
⇒ The flow is:

we run server → django looks into urls.py of project



routes to the defined paths





we get the result.

run server

↓

django looks in to urls.py (of project) i.e., the urlpatterns

urlpatterns = app.urls (urls.py of project) → looks into → urls.py (urlpatterns) of app

urlpatterns = views.index

↓

django looks into the views.py

⇒ and executes the index function

↙

Returns the result.

Django (by Telusko)

Django - Free and open source web Framework
- Used to build web applications
- Combination of Components & packages

⇒ We can use Django as framework to build web applications using 'Python' for backend.

MVC – Model View Controller ← Most of the framework use MVC
- Model → for data
- View → the what/UI
- Controller → Controls the operations

MVC/MVT is an architecture

⇒ Django follows MVT – Model View Template

Controller of MVC = View of MVT

View of MVC = Template of MVT

Django Setup:

Django is a web framework for python

→ Open Anaconda prompt in admin mode

→ create a virtual environment

(Conda create -n django_telusko)

→ activate the virtual environment &

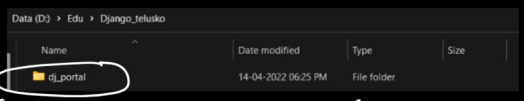
install django : pip install django

→ Create a project => dj-portal

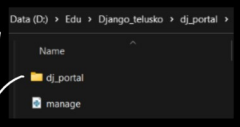
① django-admin startproject projectname

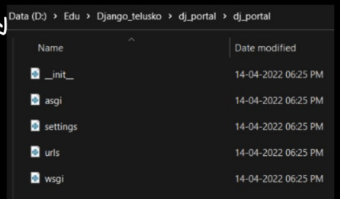
```
(django_telusko) D:\Edu\Django_telusko>django-admin startproject dj_portal
```

⇒ a project will be created (folder with projectname)



Inside it another folder with same (project) name & manage.py will be present





To check version of softwares/libraries (installed in our system)
- In cmd → python --version
- pip --version
- django-admin --version

→ we need to enter the project folder (outer)

```
(django_telusko) D:\Edu\Django_telusko>django-admin startproject dj_portal

(django_telusko) D:\Edu\Django_telusko>cd dj_portal

(django_telusko) D:\Edu\Django_telusko\dj_portal>dir
 Volume in drive D is Data
 Volume Serial Number is B49E-AE1F

 Directory of D:\Edu\Django_telusko\dj_portal

14-04-2022  06:25 PM    <DIR>          .
14-04-2022  06:25 PM    <DIR>          ..
14-04-2022  06:25 PM    <DIR>          dj_portal
14-04-2022  06:25 PM               687 manage.py
               1 File(s)            687 bytes
               3 Dir(s)  944,671,453,184 bytes free

(django_telusko) D:\Edu\Django_telusko\dj_portal>
```

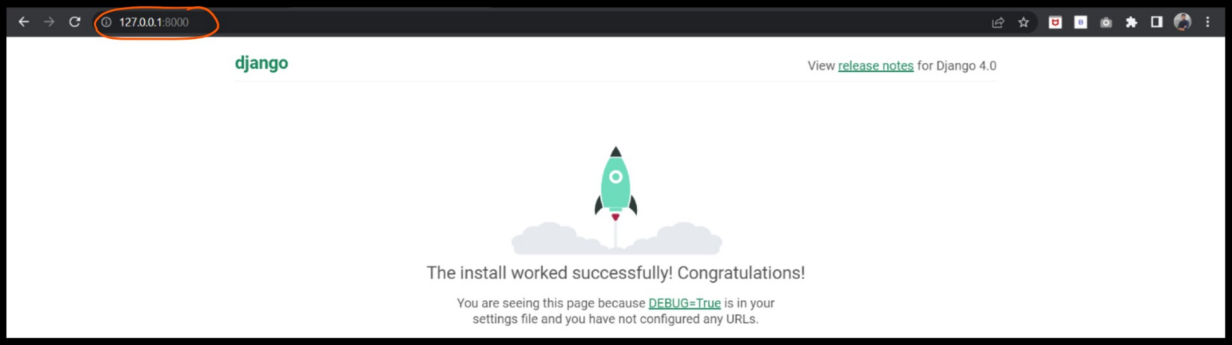
→ django provides a server to us.

② python manage.py runserver

```
(django_telusko) D:\Edu\Django_telusko\dj_portal>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 14, 2022 - 18:34:07
Django version 4.0.4, using settings 'dj_portal.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Server → http://127.0.0.1:8000/

local host port number



Creating App in Django:

Open IDE (pycharm) in admin mode.

In django project we will have multiple apps.

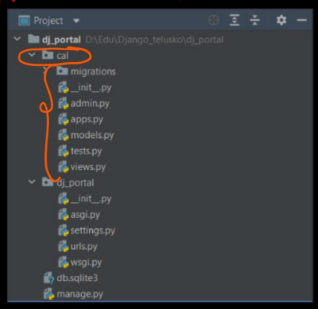
like Amazon — it has Cart, wishlist etc..., So if we want to build Amazon project, we need to create different apps for different Modules (Cart, wishlist etc...)

→ Create an app ⇒ A folder with app name (calc) will be created.

③ ⇒ python manage.py startapp calc

(calc = app name)

```
(django_telusko) PS D:\Edu\Django_telusko\dj_portal> python manage.py startapp cal
(django_telusko) PS D:\Edu\Django_telusko\dj_portal>
```

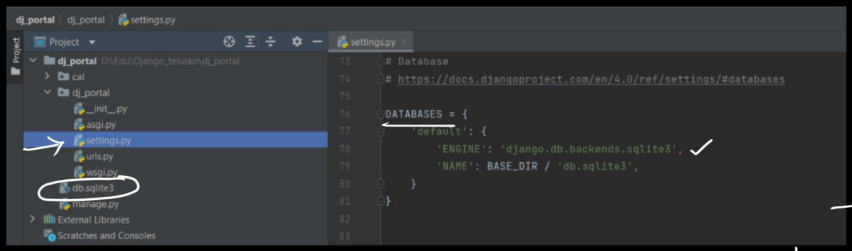


Data (D:) › Edu › Django_telusko › dj_portal

Name	Date modified	Type	Size
.idea	14-04-2022 06:55 PM	File folder	
cal	14-04-2022 07:25 PM	File folder	
dj_portal	14-04-2022 06:34 PM	File folder	
db.sqlite3	14-04-2022 06:34 PM	SQLITE3 File	0 KB
manage	14-04-2022 06:25 PM	Python Source File	1 KB

Data (D:) › Edu › Django_telusko › dj_portal › cal ›

Name	Date modified	Type	Size
migrations	14-04-2022 07:25 PM	File folder	
__init__	14-04-2022 07:25 PM	Python Source File	0 KB
admin	14-04-2022 07:25 PM	Python Source File	1 KB
apps	14-04-2022 07:25 PM	Python Source File	1 KB
models	14-04-2022 07:25 PM	Python Source File	1 KB
tests	14-04-2022 07:25 PM	Python Source File	1 KB
views	14-04-2022 07:25 PM	Python Source File	1 KB



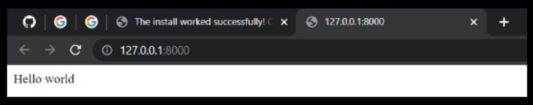
⇒ when we create a project, a settings.py will also created & it will have imp. info.

→ In it, the database is sqlite3 by default, hence a db.sqlite3 file will be created in the project.

④ write functions & urls in urls.py in views.py

⑤ Python manage.py server

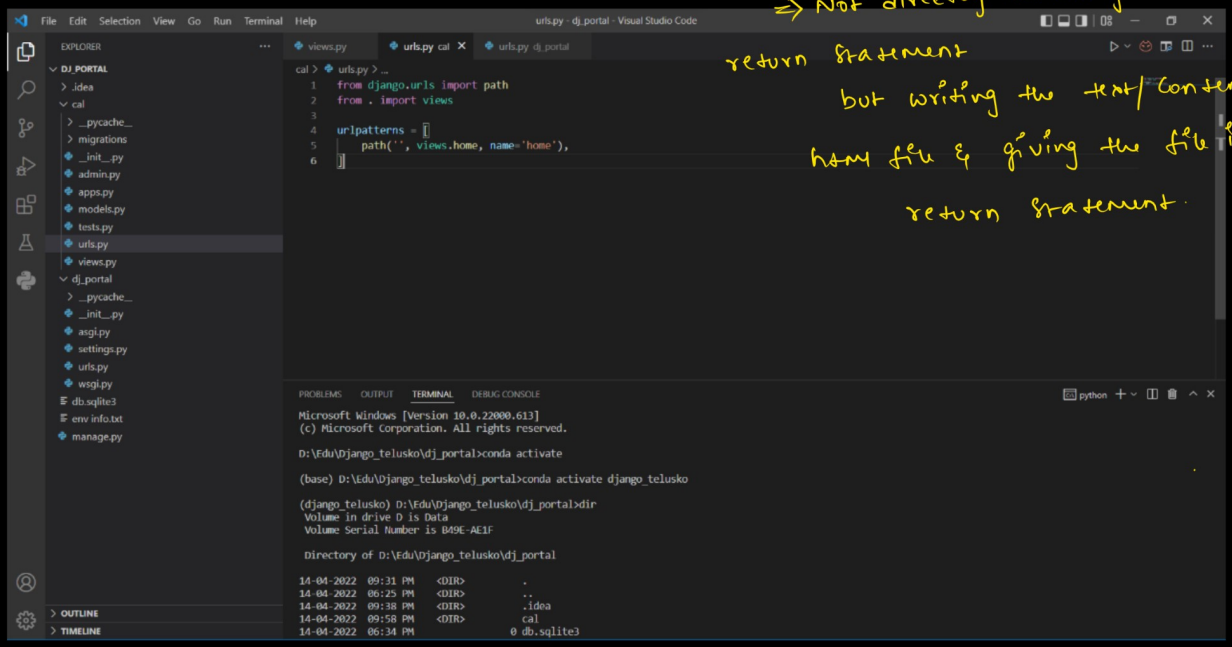
Result:



Django Template Language (DTL):

to display the dynamic content

used for printing Dynamic content which means ⇒ Not directly writting the text in return statement but writing the text/content in html file & giving the file in return statement.

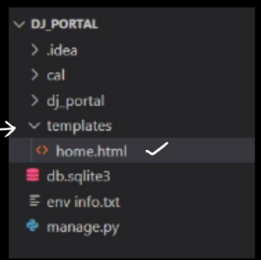


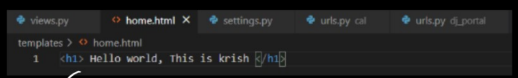
In templates, we will have html files.

→ The layout may be fixed but the content can be made dynamic.

① Create 'templates' folder & add .html file in it.

create a folder: 'templates' in the project folder (outer) & create a html file in the templates.





Mention the text to be displayed in home.html when it is called.

⇒ In 'templates' folder, we keep our html files.

② Add the created 'templates' folder path in the 'DIRS' of the 'Templates' in the settings.py of the project.

```
ROOT_URLCONF = 'dj_portal.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        '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',
            ],
        },
    },
]

WSGI_APPLICATION = 'dj_portal.wsgi.application'
```

In the settings.py, in Templates, we need to mention the directory where our templates folder is present. So that the django can look into the files present in our 'templates' folder

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, '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',
            ],
        },
    },
]

WSGI_APPLICATION = 'dj_portal.wsgi.application'
```

Giving the directory of our 'templates' folder.

We need to render our template (.html file) because template is a file & it can have some dynamic content. So, we have to merge the static content with dynamic content and that is called 'rendering'.

③ In views.py call the .html file

```
from django.shortcuts import render, HttpResponse
from django.http import HttpResponse


# Create your views here.

def home(request):
    # return HttpResponse("Hello world")
    return render (request, 'home.html')
```

.html file name

④ Run server

Result

127.0.0.1:8000

Hello world, This is krish

---

flow: (for dynamic Content)

This is the flow for getting the dynamic Content (Code-wise we can make changes)

Start server ①

```
<h1> Hello world, This is krish </h1>
```

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

```
"""
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('cal.urls')),
    path('admin/', admin.site.urls),
]
```

Result

127.0.0.1:8000

Hello world, This is krish

Making the Content more dynamic:

```
templates > home.html
1  <h1> Hello world, This is {{name}} </h1>
```

we can get this name from anywhere, this make the content more dynamic

```
cal > views.py > ...
1  from django.shortcuts import render, HttpResponse
2  from django.http import HttpResponse
3
4
5  # Create your views here.
6
7  def home(request):
8      # return HttpResponse("Hello world")
9      return render (request, 'home.html', {'name':'krishna'})
10
```

⇒ we are not fixing the content in the file to be displayed, we are fetching the info (name) from other file/code which makes our code dynamic.
(↳ python code - views.py)

```
templates > home.html
1  <h1> Hello world, This is {{name}} </h1>
2
```

the curly bracket here means that the code is dynamic,

→ this is "Jinja pattern" (or) "Jinja Template"

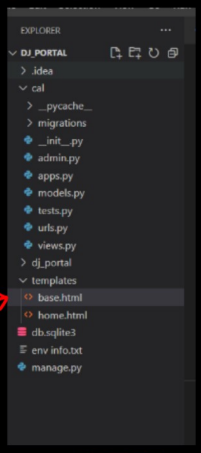
→ In the curly brackets, instead of variable name, we can give expressions also.

Making the Content Dynamic - Creating base.html:

In the website, we will have multiple pages. The UI can be same but the (↳ layout, color scheme etc..)
content displayed in different pages will be different.

→ So, we can take the content/code which is same for all pages & keep it in a file.

Creating a html file (base.html) & keeping basic code which will be common for all the pages.



We kept the common code in base.html, which means the pages will use this base.html & their own code as per the page requirements.

Extending the base.html to home.html

basic html code

```
templates > base.html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=0device-width, initial-scale=1.0">
6       <meta http-equiv="X-UA-Compatible" content="ie=edge">
7       <title>Krish_Dj</title>
8   </head>
9   <body bgcolor='cyan'>
10      {% block content %}
11
12
13      {% endblock %}
14  </body>
15  </html>
16
```

→ Color for the page

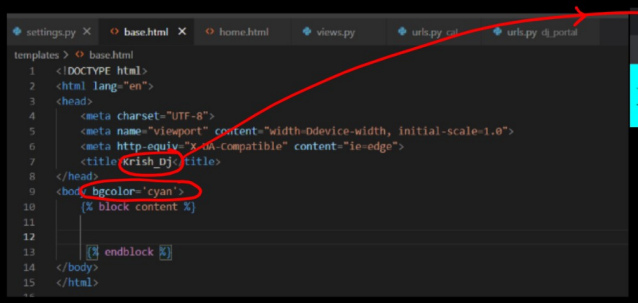
→ In the body we mention the 'block content'

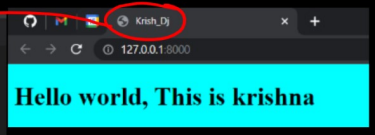
```
{% extends 'base.html' %}
{% block content %}
<h1> Hello world, This is {{name}} </h1>
{% endblock %}
```

→ As we need to use base.html with home.html

Content written in the block content will be sent to 'block content' of base.html

```
cal > views.py > home
1  from django.shortcuts import render, HttpResponse
2  from django.http import HttpResponse
3
4
5  # Create your views here.
6
7  def home(request):
8      # return HttpResponse("Hello world")
9      return render (request, 'home.html', {'name':'krishna'})
```

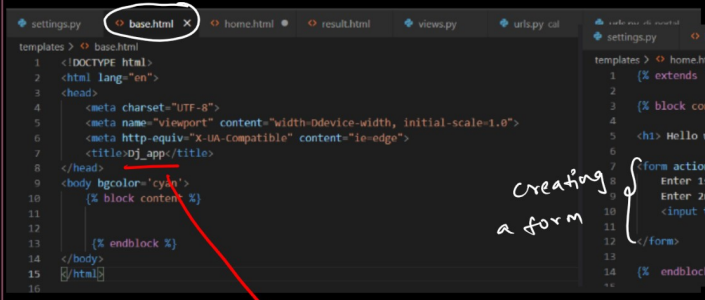


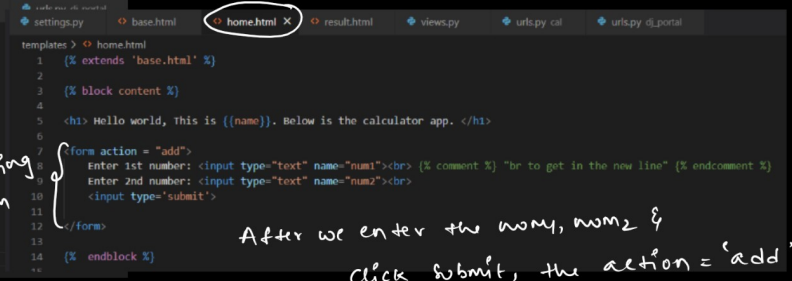


⇒ Server → urls.py (dj_portal) → urls.py (cal)
↓
views.py
↓
home.html

In home.html, we had written 'extends base.html', which means base.html + home.html

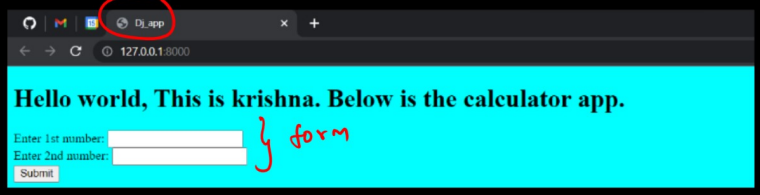
Addition of two numbers in Django



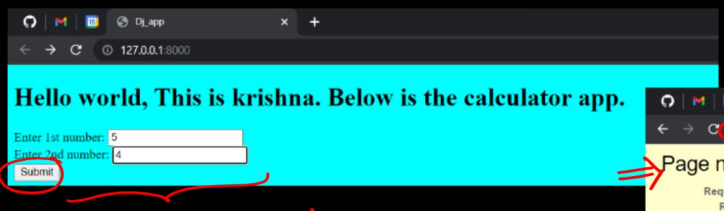


creating a form

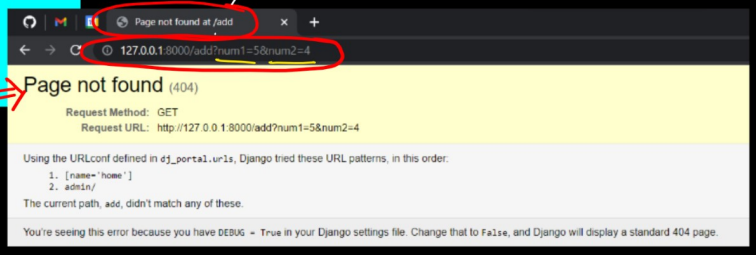
After we enter the num1, num2 & click submit, the action = 'add'



} form



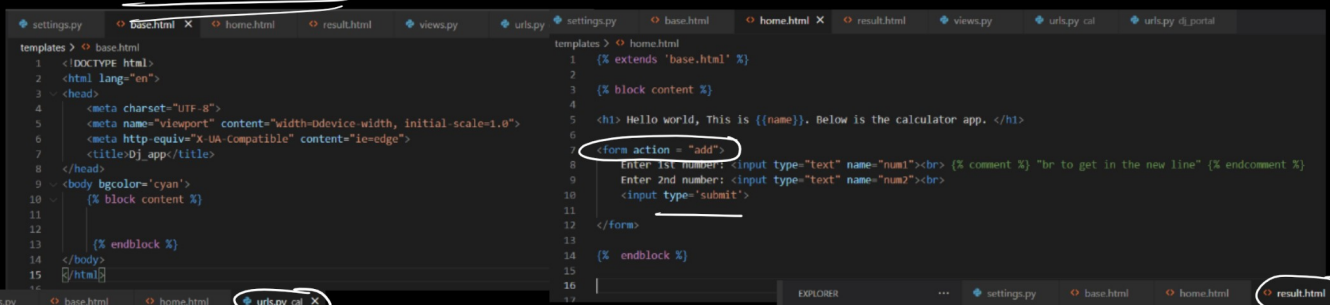
Giving num1, num2 & click on submit.

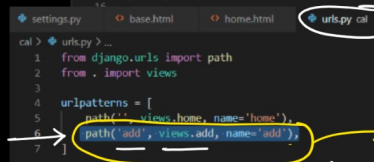


→ we need to include the path for 'add' to fix the issue.

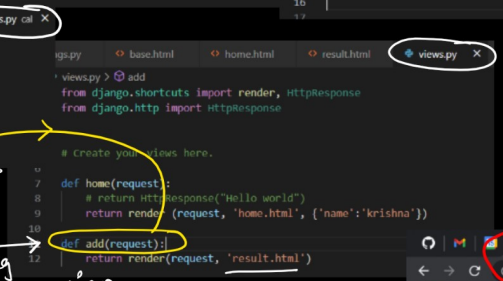
then the action is 'add', which we had mentioned by not mapped anywhere → so, we will get error.

→ Mapping the path for 'add':

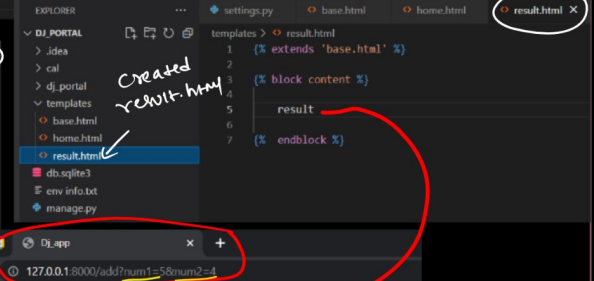




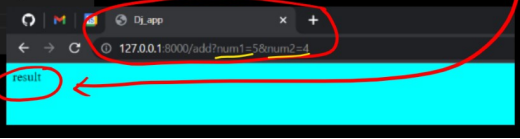
adding the path for 'add' in urlpatterns



defining the add() function



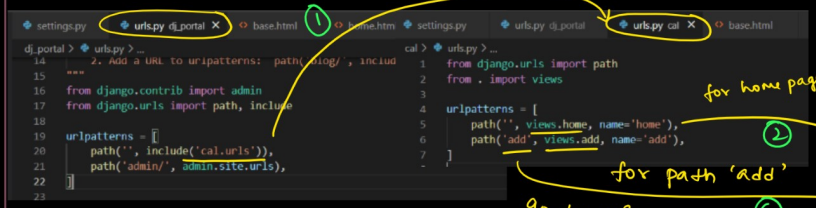
Created result.html



Flow for adding two numbers in Django

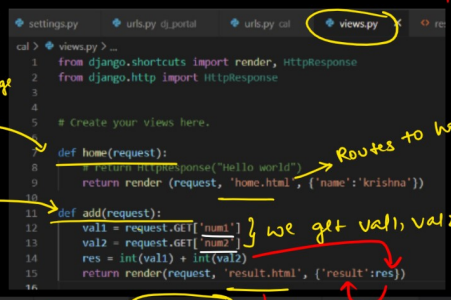
⇒ Used base.html, home.html, result.html

(We need to add our templates path in 'DIRS' of Templates in settings.py)



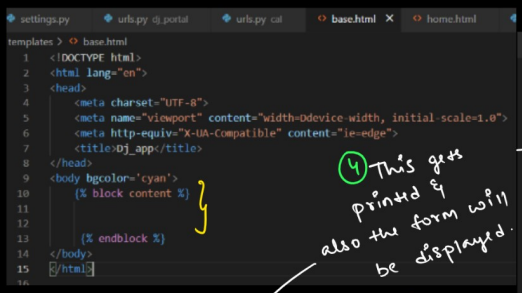
① ② for home page

for path 'add' ⑥ go to views.add

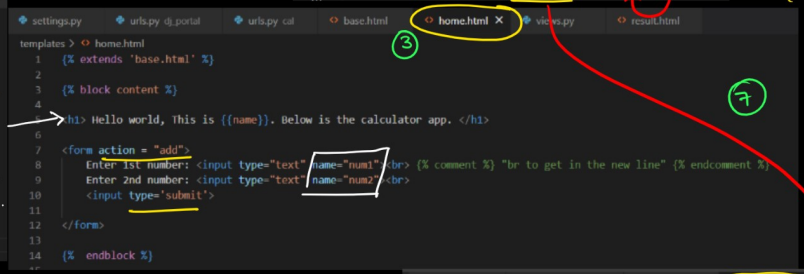


Routes to home.html

we get val1, val2 entered in the form (browser)

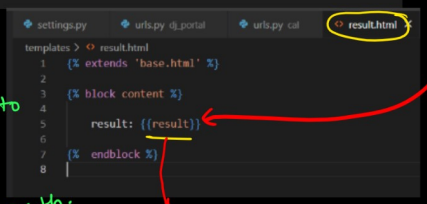


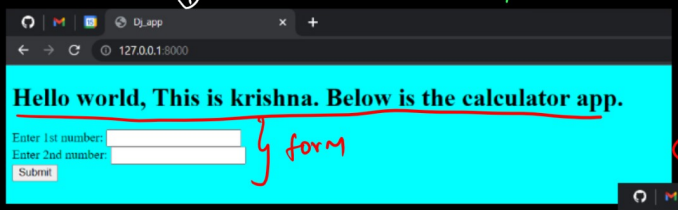
④ This gets printed & also the form will be displayed.



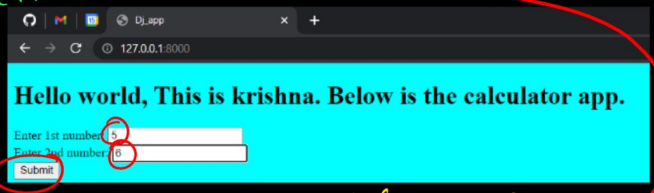
③ ⑦

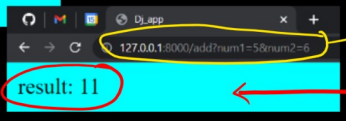
⑤ Once we enter the values & click 'submit' then the django will look for the form's action to perform which is 'add' ⇒ django will look into "urlpatterns" of urls.py (of app) for the path defined for 'add' & routes to the module/function defined in that path.



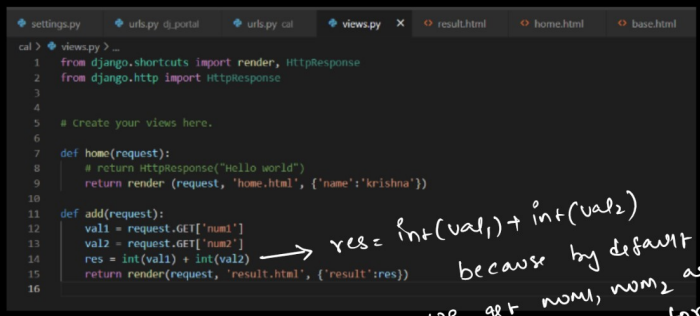


form





data(num1, num2) are displayed.



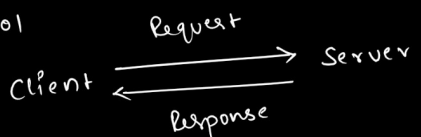
$res = int(val_1) + int(val_2)$

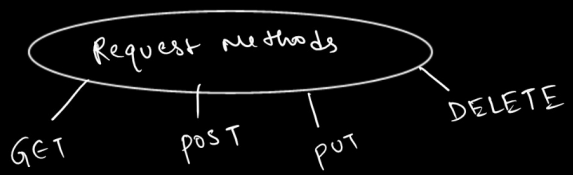
because by default we get num_1, num_2 as strings from the form

Django (Telusko)

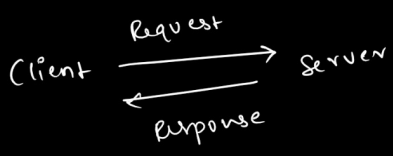
GET Vs POST - HTTP Methods

HTTP protocol





→ When we want to fetch resource from the server, by sending some data, we use GET Request.



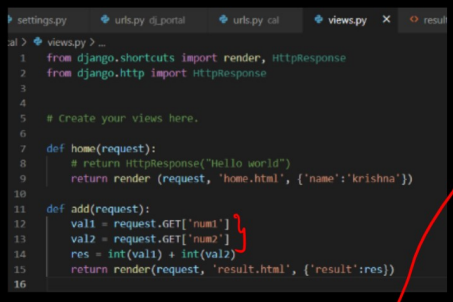
(we are expecting something from the server)

→ POST - Sending something to server (like filling a form & submitting it)

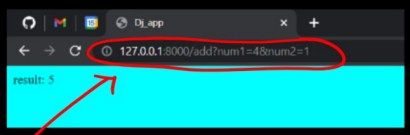
(Adding data to server)

Client → Server

→ PUT - updating data on the server



```
from django.shortcuts import render, HttpResponse
from django.http import HttpResponse


# Create your views here.

def home(request):
    # return HttpResponse("Hello world")
    return render (request, 'home.html', {'name':'krishna'})

def add(request):
    val1 = request.GET['num1']
    val2 = request.GET['num2']
    res = int(val1) + int(val2)
    return render(request, 'result.html', {'result':res})
```



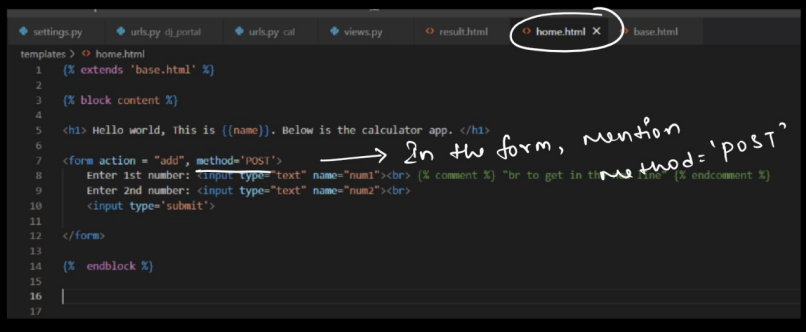
We had used 'GET' method in our code for the addition of the 2 numbers.

→ In GET method, we send the data (num_1, num_2) in the address bar.

⇒ In this case our data will be visible to everyone & is not secured.

(for eg.. login credentials)

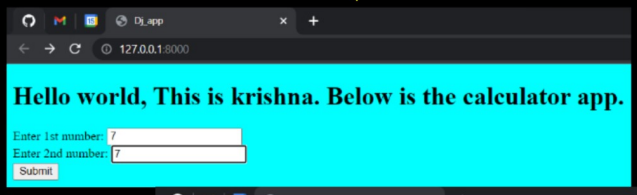
→ So to overcome this we use 'POST' method.

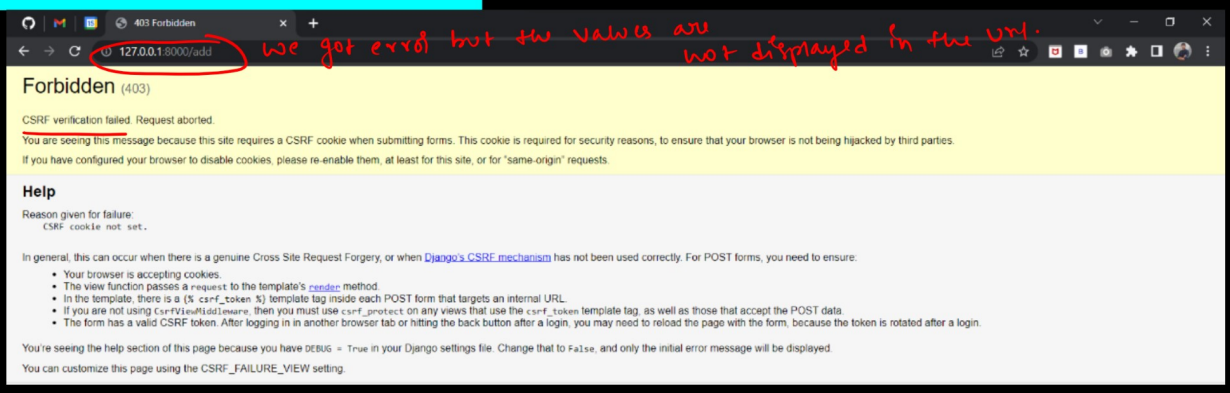


```
{% extends 'base.html' %}

{% block content %}

<h1> Hello world, This is {{name}}. Below is the calculator app. </h1>

<form action = "add", method='POST'>
    Enter 1st number: <input type="text" name="num1"><br> {% comment %} "br to get in the next line" {% endcomment %}
    Enter 2nd number: <input type="text" name="num2"><br>
    <input type='submit'>

</form>

{% endblock %}
```

→ In the form, mention method='POST'

The entire code of 'Addition of two numbers in Django' will be same. We just need to change the method from GET to POST.





Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated. CSRF attacks exploit the trust a Web application has in an authenticated user.

So, to avoid this insecurity, In django we have Middlewares.
We need to use them.

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

home.html

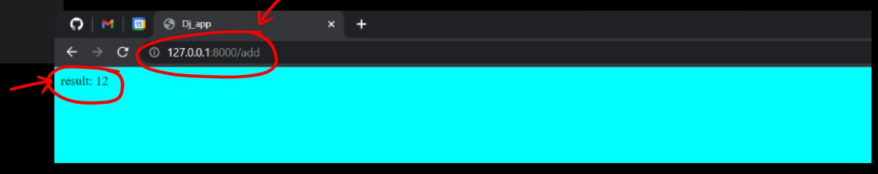
```
{% extends 'base.html' %}

{% block content %}

<h1> Hello world, This is {{name}}. Below is the calculator app. </h1>

<form action = "add", method='POST'>

    {% csrf_token %}
    Enter 1st number: <input type="text" name="num1"><br> {% comment %} "br to get in the new line" {% endcomment %}
    Enter 2nd number: <input type="text" name="num2"><br>
    <input type='submit'>

</form>

{% endblock %}
```

So, we are using csrf_token to send the data now.

views.py

```
from django.shortcuts import render, HttpResponse
from django.http import HttpResponse


# Create your views here.

def home(request):
    # return HttpResponse("Hello world")
    return render (request, 'home.html', {'name':'krishna'})

def add(request):
    val1 = request.POST['num1']
    val2 = request.POST['num2']
    res = int(val1) + int(val2)
    return render(request, 'result.html', {'result':res})
```

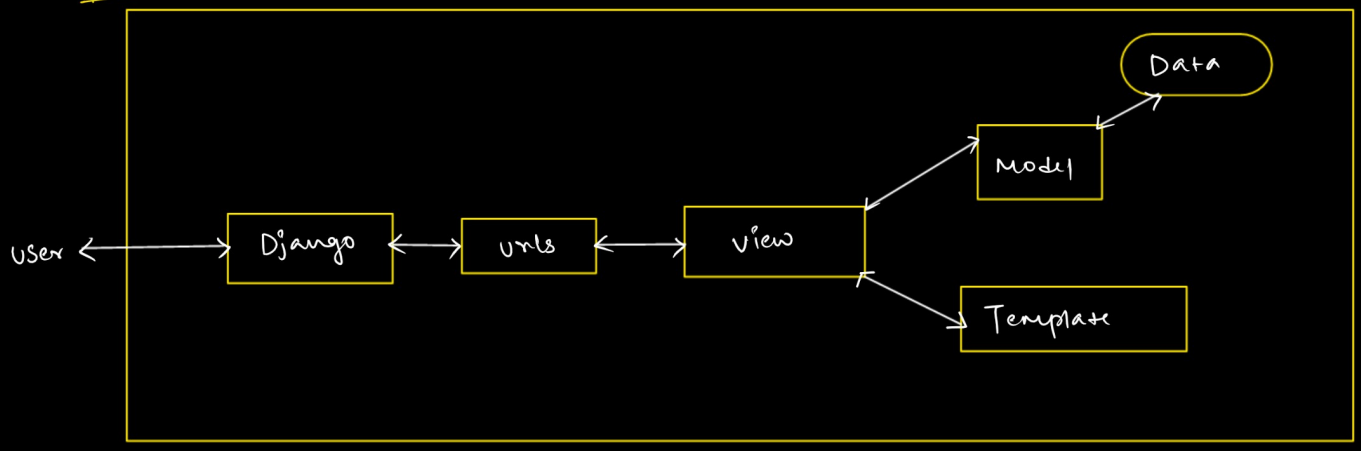
request.POST['num1']



Model View Template MVT / MTV

→ In Django, we used MVT.
↳ Model View Template
→ In every web app we need - layout, logic, data

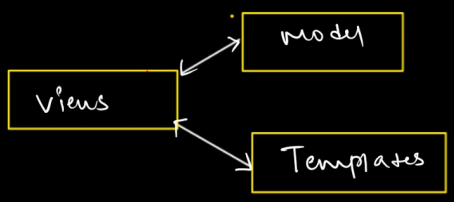
MVT Framework:



1, Model is for the data. The data will be coming from the database & that can be linked to the 'Model' object.

2, Template is of Normal html. we will have html, css, frontend Javascript.
and it will also have Django Template Language (DTL).
↙
used to include Dynamic data in the page.

3, The business logics will be written in views.



The views will use the model object & the Templates.
→ what data has to be sent on Template will be decided by 'View'.
→ The connection to the models will be done through views.

⇒ Model will work with data
View will work with logics
Templates is for the layout.

This is called 'Seperation of Concerns',
→ we had different Modules (Model, View, Template) for different Concerns (data, logics, layout).