15-5-2024

**Django**

**Objective:** The main objective of learning django is developing web applications or web enabled applications or internet applications.

**Full Stack Python**

**Module-1 Python**

**Core Python**

**Adv Python**

**Module-2 Python for Data Science**

**Numpy**

**Pandas**

**Matplotlib**

**Module-3 UI Module**

**HTML**

**CSS**

**JavaScript**

**Bootstrap**

**AngularJS**

**Module-4 Backend Module**

**Django 🡪 Project**

**Flask 🡪 Project**

**REST API 🡪 Project**

**Module-5 Database**

**MySQL**

**MongoDB**

**Module-6 Tools**

**AWS Basics**

**Testing**

**Git/GIT HUB**

**Docker**

**Pre Learning Django is,**

1. Basics of Python Language
   1. Data Types
   2. Functions
   3. Classes and objects
2. UI (HTML,CSS,JS,Bootstrap)

**Types of Applications**

1. Console Based Applications
2. Windows Based Applications
3. Web Applications
4. Enterprises Applications
5. Mobile Applications

**Console Based Applications**

Console based applications are standalone applications. Console based applications are also called CUI (Character User Interface) applications. In CUI, user interacts with application by typing commands.

**Example:**

# CUI

num1=int(input(“Enter first number “))

num2=int(input(“Enter second number “))

opr=input(“Enter any operator “)

match(opr):

case ‘+’:

print(num1+num2)

case ‘-‘:

print(num1-num2)

case ‘\*’:

print(num1\*num2)

case ‘/’:

print(num1/num2)

**Output**

Enter first number 10

Enter second number 20

Enter any operator +

30

**Windows based applications or GUI applications**

GUI stands for Graphical User Interface, these application end user interact by clicking graphical components or widgets (Button, radio, checkbox,…).

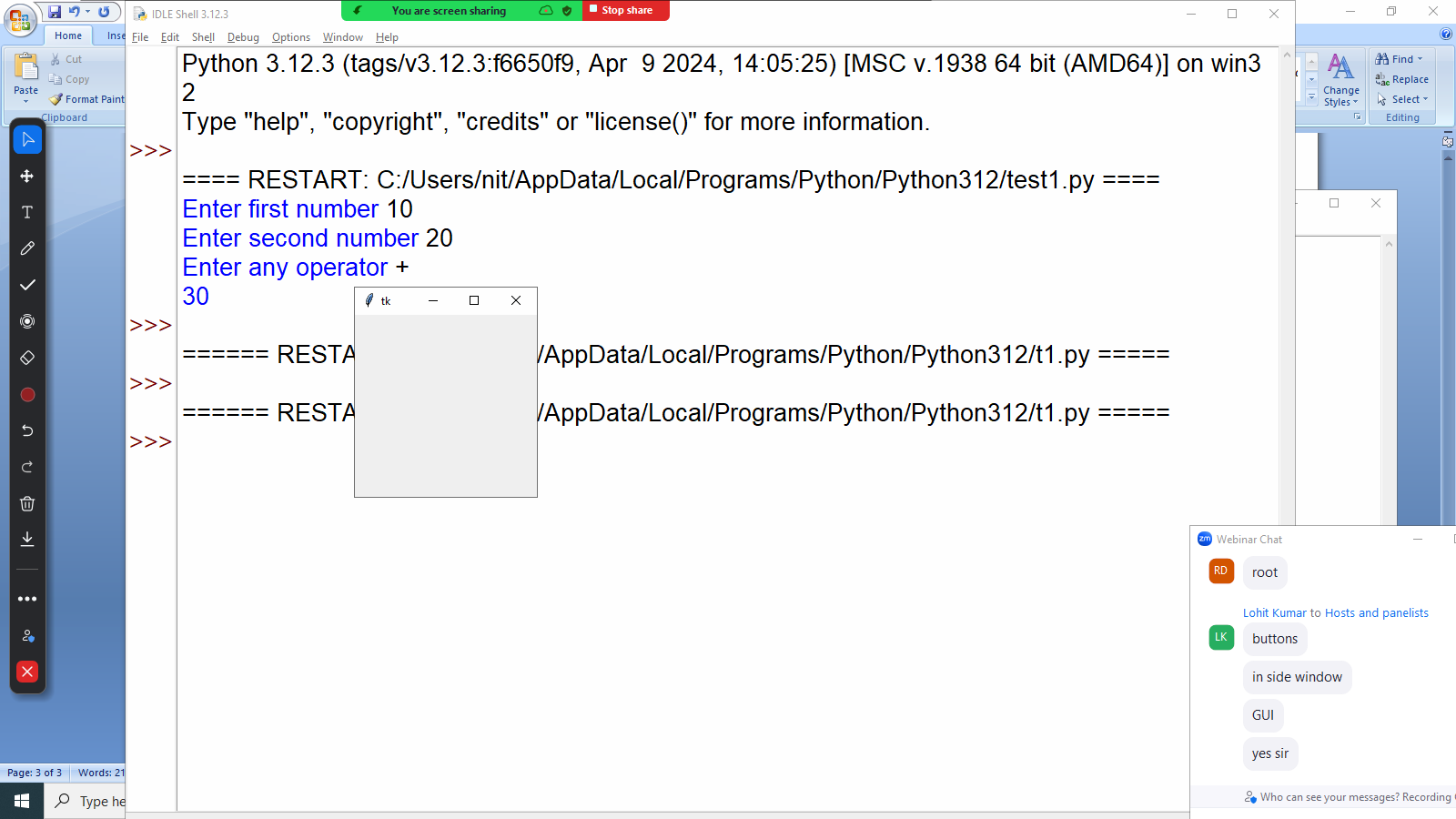
1. Tkinter
2. Kivy
3. PyQT

**Example:**

import tkinter

window=tkinter.Tk()

**Output**



**Web Applications**

Application is a collection of programs or services, accessing these services using web or internet.

1. Front End (UI)
   1. HTML :Hyper Text Markup Language used for create user interface or web page or define structure of web page.
   2. CSS : Cascading Style Sheet, Styling language used styling or formatting web page
   3. JavaScript : Java script is an object oriented programming language, define functionality of web page
   4. Bootstrap : CSS Library, contain predefined style sheets
   5. AngularJS : JavaScript frontend Framework
2. BackEnd
   1. Django
   2. Flask
   3. RestAPI

**Mobile Applications**

The applications run inside mobile device.

Kivy is a library used for developing mobile applications.

16-05-2024

**What is web application?**

A web application (web app) is an application program that is stored on a remote server and delivered over the internet through a browser interface

**Types of Web Applications**

There are two types of web applications

1. Static web application
2. Dynamic web application

**Advantage of web applications**

1. This application not required to install in system
2. These applications can run irrespective hardware devices (Desktop, Mobile, …), provided interface/user-agent(Browser)
3. Not required having high configure system (SAAS)
4. Not required maintain different versions.

**Networking required two programs**.

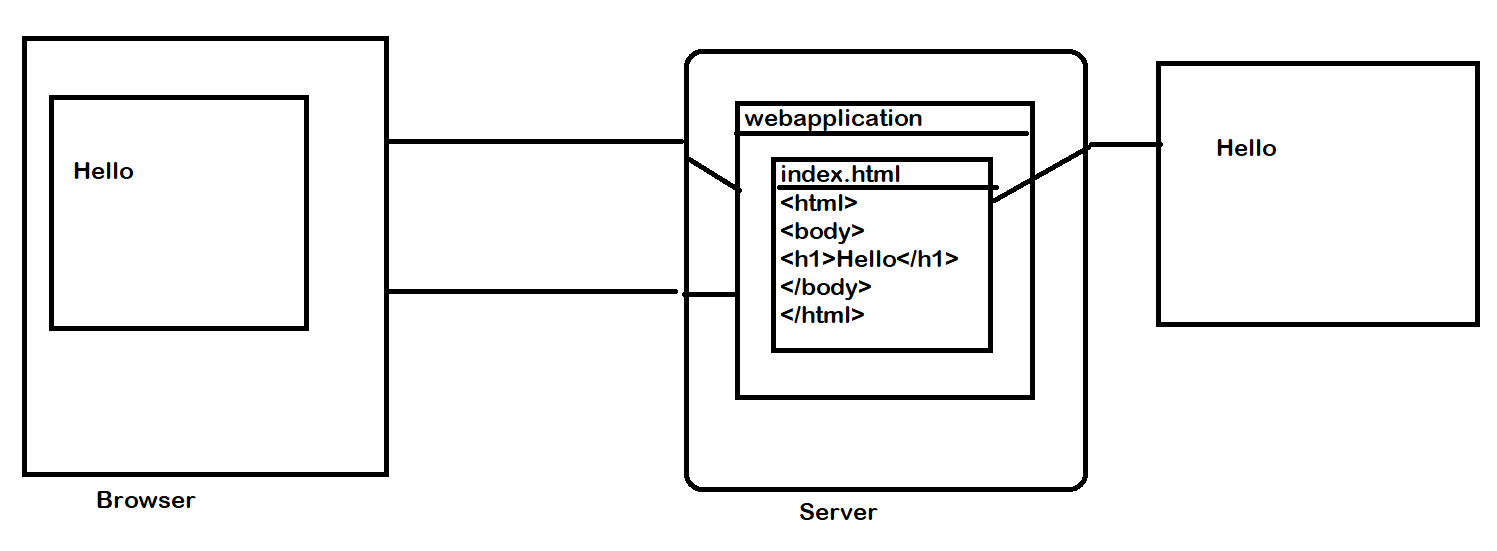
1. Client (Browser) (User-Agent)
2. Server (Software)

**Static web application**

Static web application is collection of,

1. Web pages (HTML)
2. CSS files
3. Js files
4. Images
5. Audios
6. Videos

In static web application, web page is designed with pre-written content. This content does not change from one user another user.



Static web application response (output) does not changes from one request to another request. For developing static web application not required any programming at server side.

**Dynamic web applications**

Dynamic web application whose repose changes from one request to another request.

Dynamic page is generated by inserting data from database, other programs or files or perform some operations.

1. HTML
2. CSS
3. JS
4. BootStrap
5. Angular
6. Django/Flask
7. RESTAPI

**What is Framework?**

Framework is software which provides development environment and execution environment. It also provides architecture for designing applications.

1. Client Side Frameworks
   1. Angular
   2. React
2. Server Side Frameworks
   1. Django (Python)
   2. Falsk (Python)
   3. Spring Boot (Java)
   4. ASP.NET (.Net)

**What is library?**

Library is set of predefined programs with predefined functionality.

Python libraries are represented as modules and packages.

17-05-2024

**What is Django?**

* Django is a Python framework that makes it easier to create web sites using Python.
* [Django](https://djangoproject.com/) is an open-source web framework written in the [Python](https://www.python.org/) programming language. Named after the jazz guitarist [Django Reinhardt](https://en.wikipedia.org/wiki/Django_Reinhardt), it is used by some of the largest websites in the world including Instagram, Mozilla, and NASA

**[Django](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "where_did_it_come_from) History**

* Django was initially developed by web programmers at the **Lawrence Journal-World newspaper,** specifically Adrian Holovaty, Simon Willison, and Jacob Kaplan-Moss.
* Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lots of common code and design patterns. This common code evolved into a generic web development framework, which was open-sourced as the "Django" project in July 2005.
* It was first released to the public as an open source package in 2005 and is currently maintained by the non-profit [Django Software Foundation](https://www.djangoproject.com/foundation/).
* Django has continued to grow and improve, from its first milestone release (1.0) in September 2008 through to the version 5.0 in late 2023.

<https://www.djangoproject.com/>

**Features**

* **[Complete](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "complete)**

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and [up-to-date documentation](https://docs.djangoproject.com/en/stable/" \t "_blank).

* **[Versatile](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "versatile)**

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format

* **[Secure](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "secure)**

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically

20-05-2024

**Features**

* **[Complete](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "complete)**

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and [up-to-date documentation](https://docs.djangoproject.com/en/stable/" \t "_blank).

* **[Versatile](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "versatile)**

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format

* **[Secure](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "secure)**

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically

* **[Scalable](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "scalable)**

Django uses a component-based "[shared-nothing](https://en.wikipedia.org/wiki/Shared_nothing_architecture" \t "_blank)" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers.

* **[Maintainable](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "maintainable)**

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the [Model View Controller (MVC)](https://developer.mozilla.org/en-US/docs/Glossary/MVC) pattern).

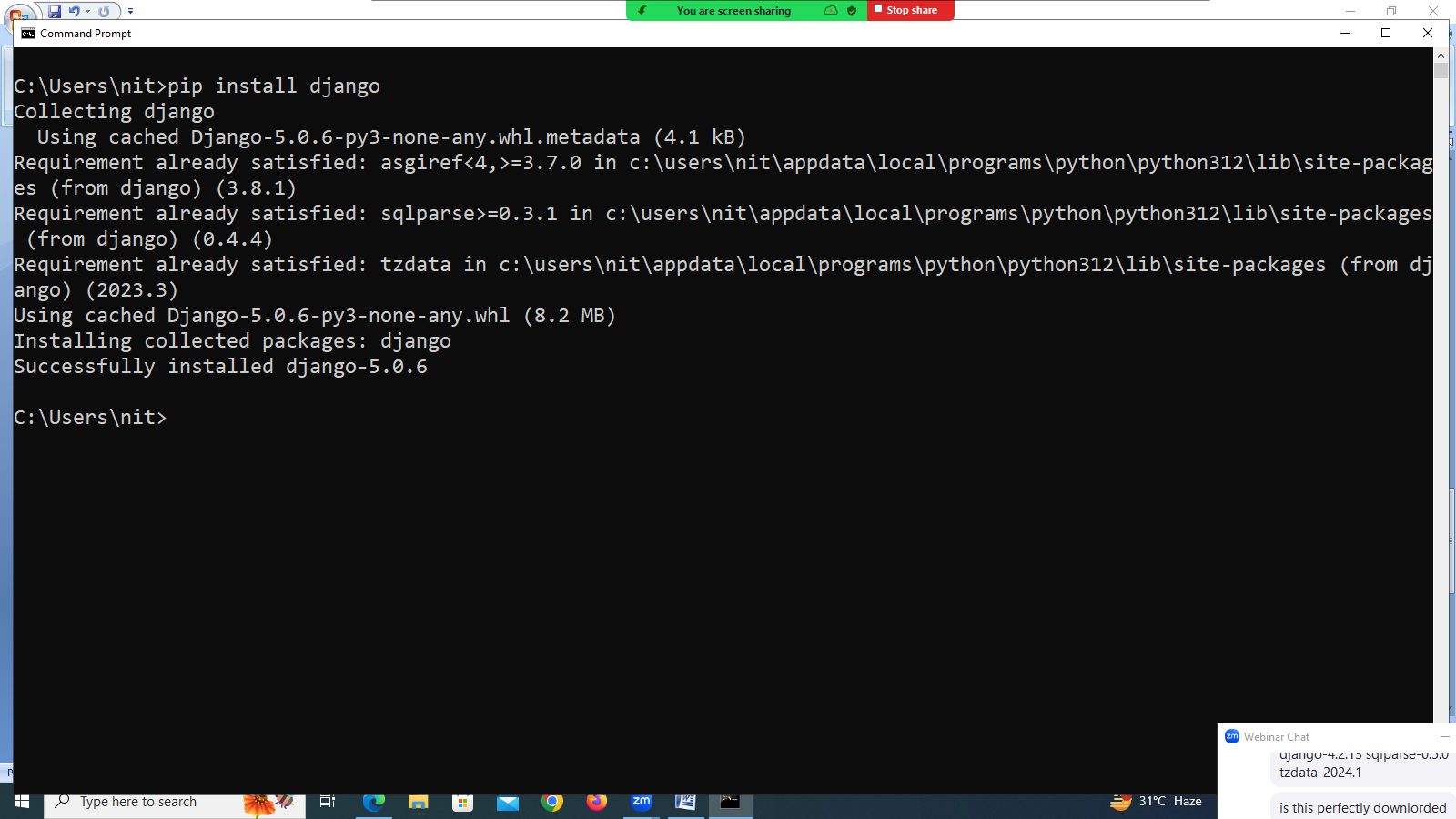
* **[Portable](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction" \l "portable)**

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavors of Linux, Windows, and macOS.

What Python version can I use with Django?

| Django version | Python versions |
| --- | --- |
| 3.2 | 3.6, 3.7, 3.8, 3.9, 3.10 |
| 4.0 | 3.8, 3.9, 3.10 |
| 4.1 | 3.8, 3.9, 3.10, 3.11 |
| 4.2 | 3.8, 3.9, 3.10, 3.11, 3.12 |
| 5.0 | 3.10, 3.11, 3.12 |

**Installing django package/software/django framework**



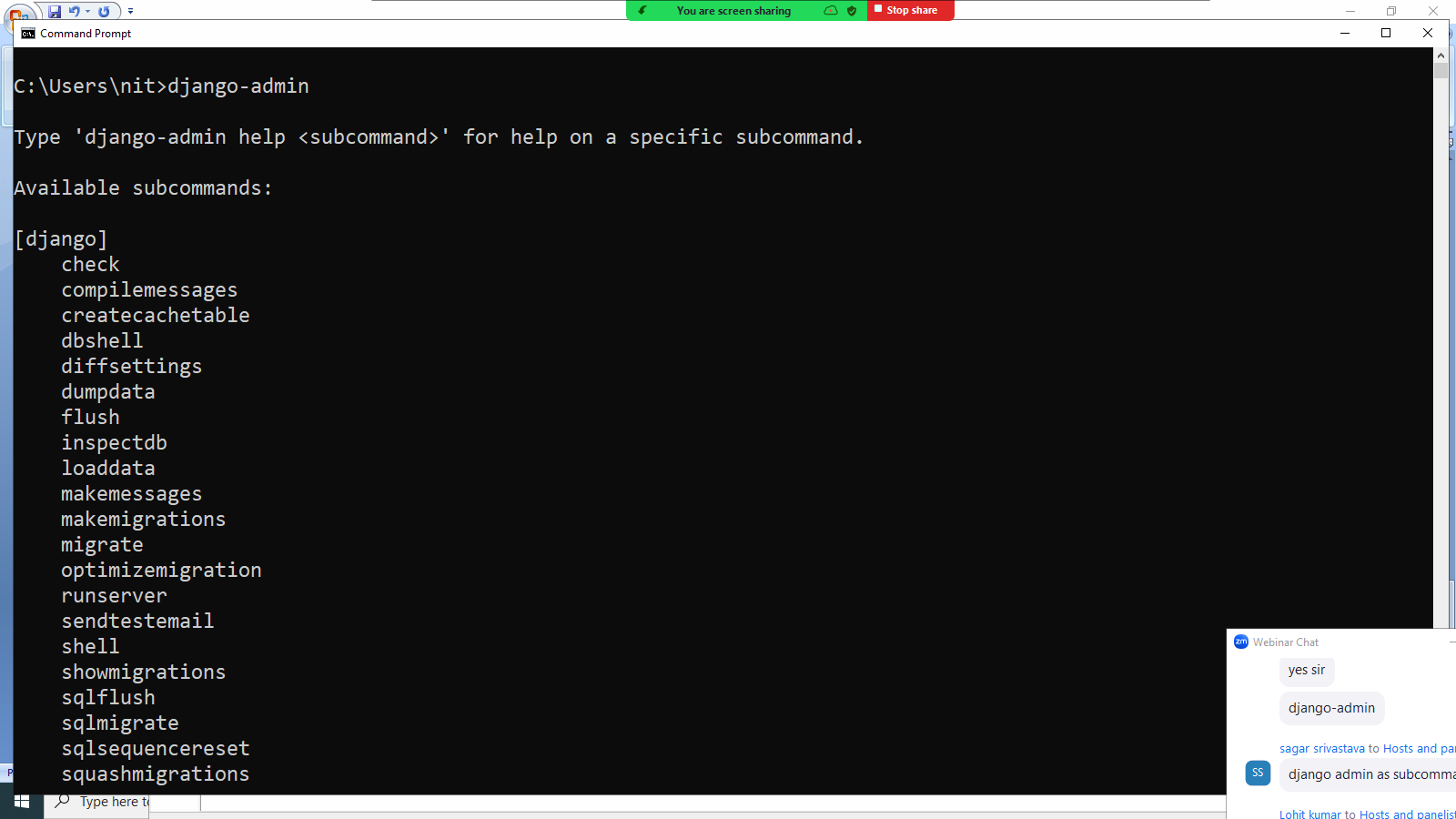
**django-admin command**

django-admin is a command line utility.

This dango-adimn is available in script folder or django.

Django-admin command is used for performing administrative task.

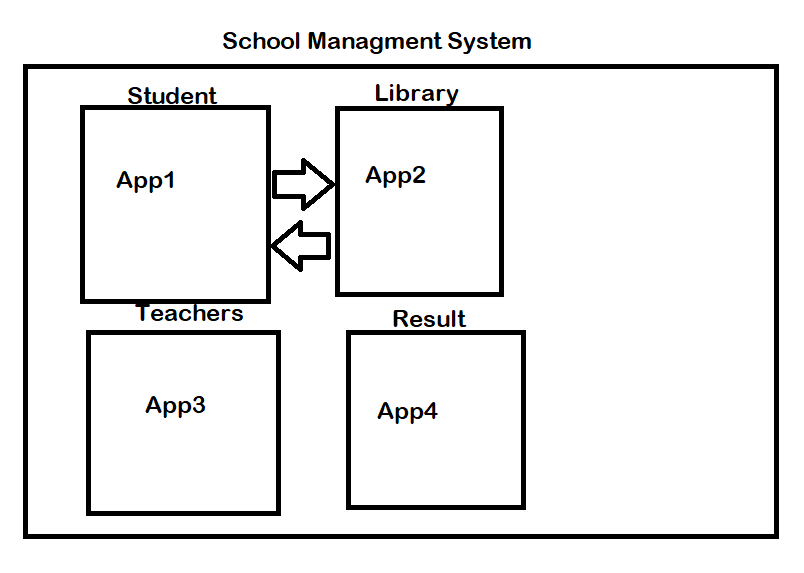
**How to see the command of django-admin?**

****

**Creating django project**

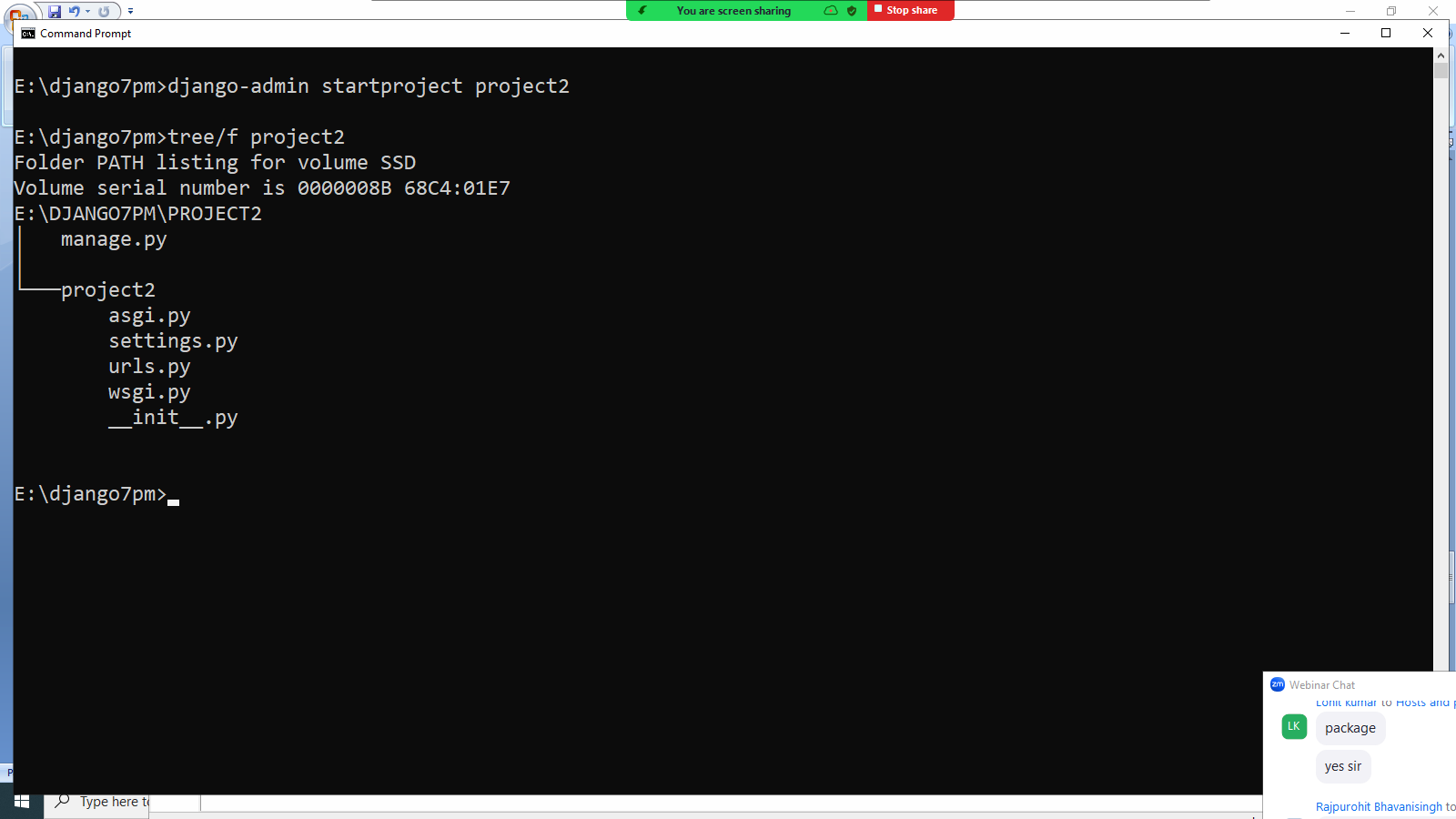
**What is django project?**

Django project is collection applications and configuration files.



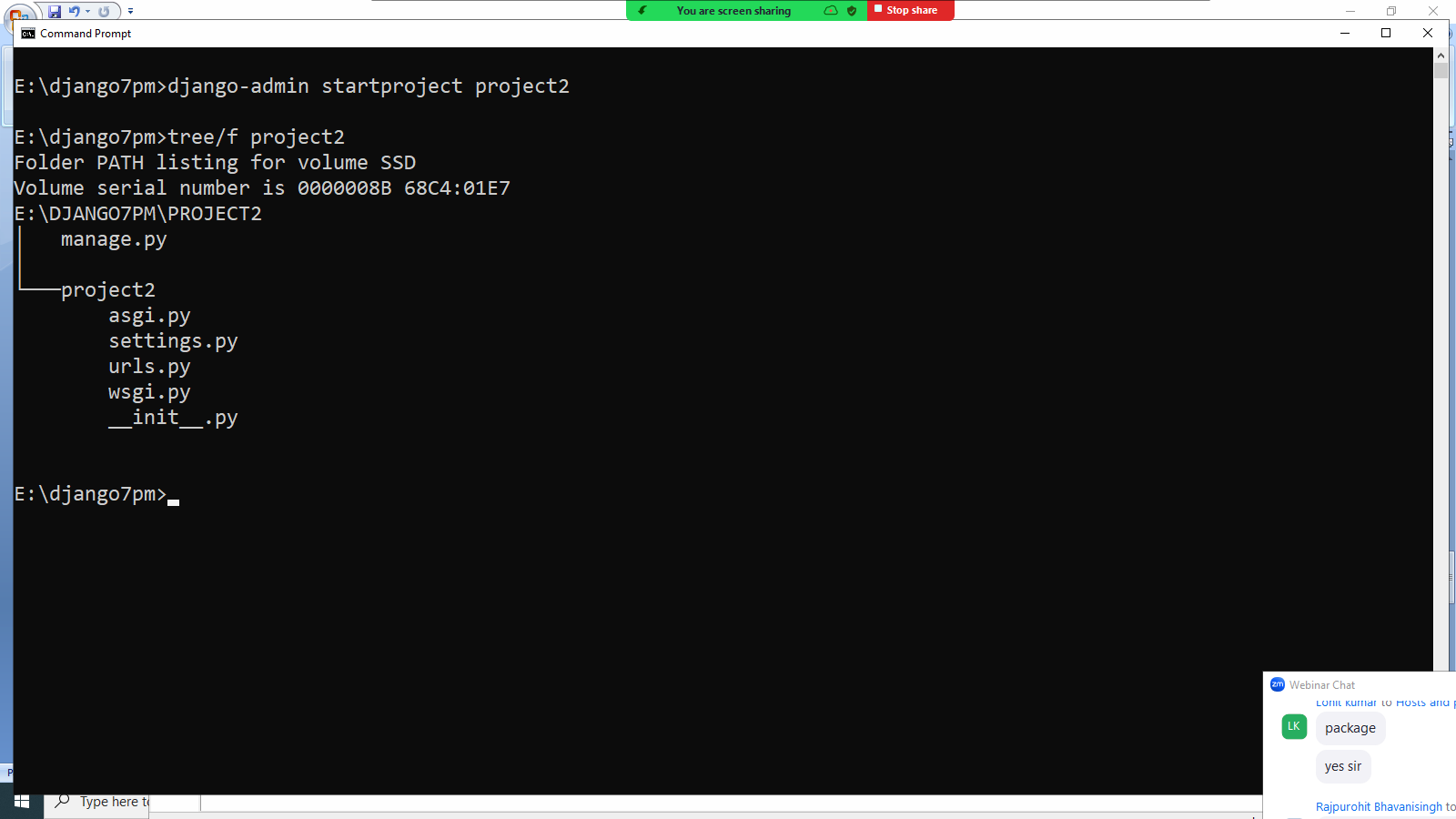
**How to create project?**

django-admin startproject <project-name>



**21-08-2024**

django-admin startproject <project-name>



**manage.py**

**manage.py is command line utility** which is used to manage project.

Example: create applications, startserver,...

**The project contains the following modules**

for every project python create a package with project name.

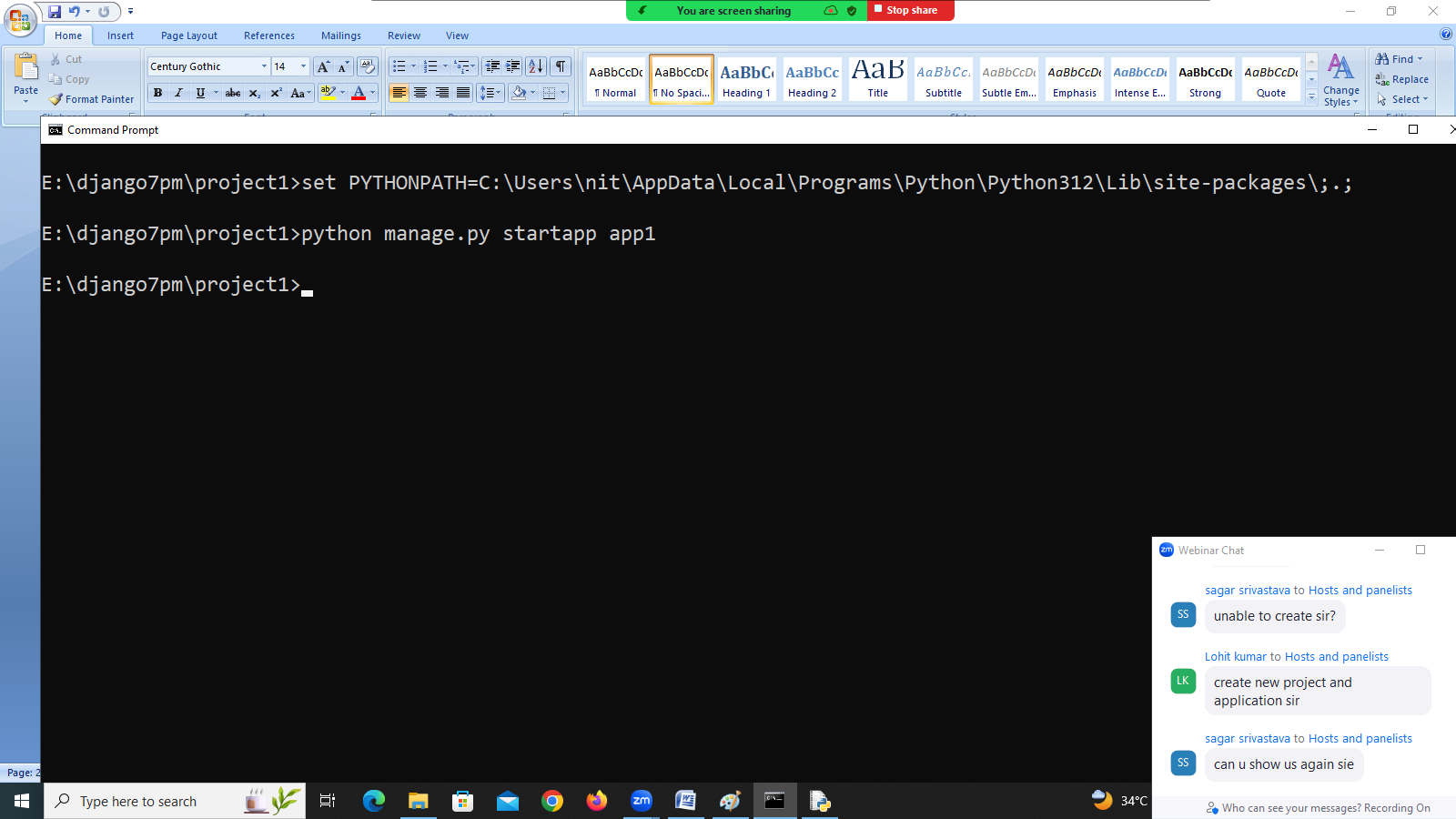
1. **\_\_init\_\_.py : \_\_init\_\_.py is** called package configuration file. This file executed automatically when package is imported or when module is imported from package. It is file which is used identified for identifying a folder as package.
2. **settings.py :** project configuration file.
3. Installing applications
4. Defining middleware
5. Defining database
6. Defining path for static files
7. **Urls.py:** This module is to define url-patterns at project level.
8. **Wsgi.py : WSGI web server gateway interface.** Wsgi servers are synchronized. It is able to process one request from one client. Wsgi.py is used to configure wsgi servers.
9. **Asgi.py : Asynchronous server gateway interface. ASGI based server process** multiple request of same client. For configuring asgi based server we use asgi.py

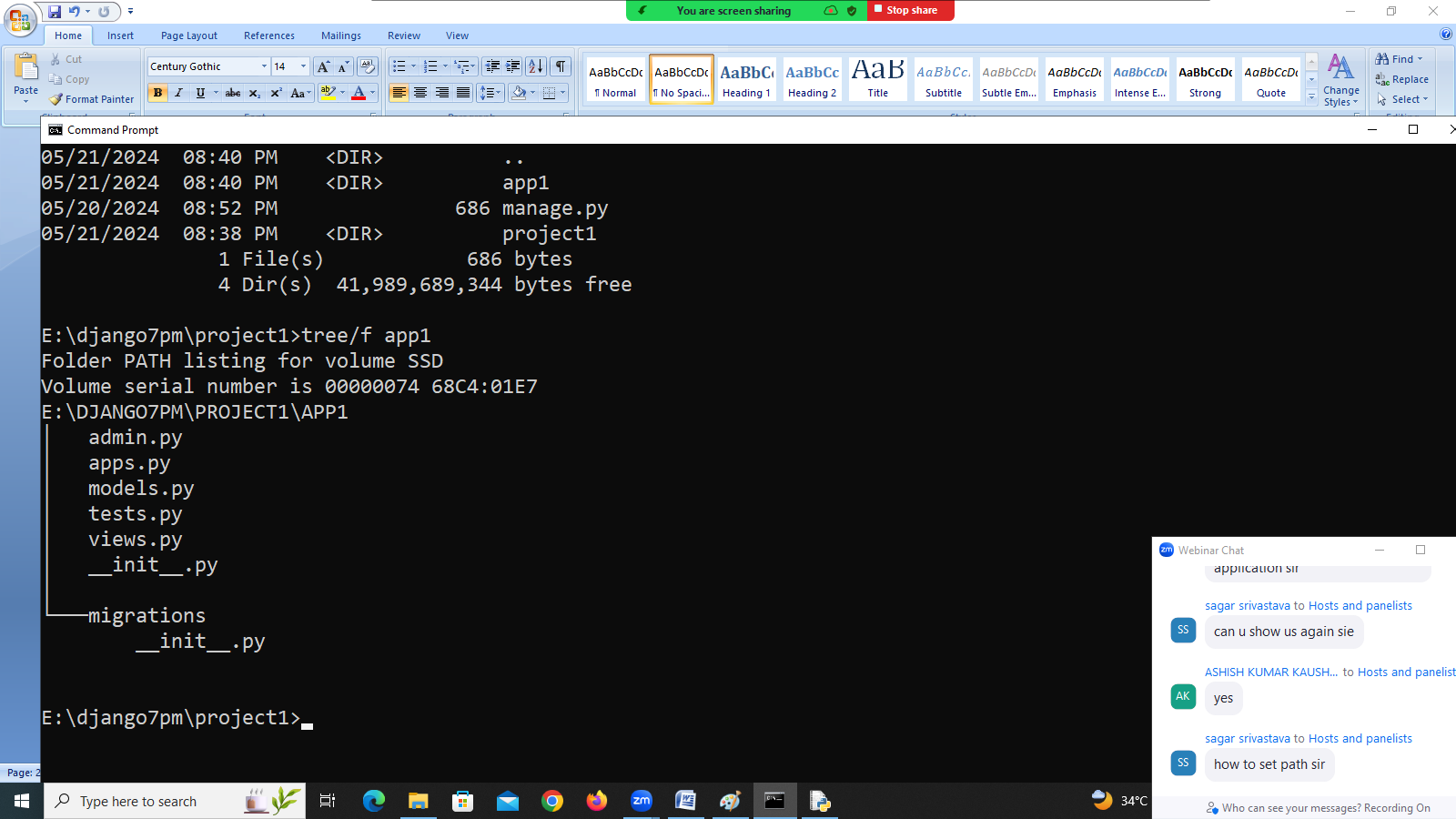
**Creating application**

Django project is a collection of applications.

An application cannot exists without project.

**How to create application?**



****

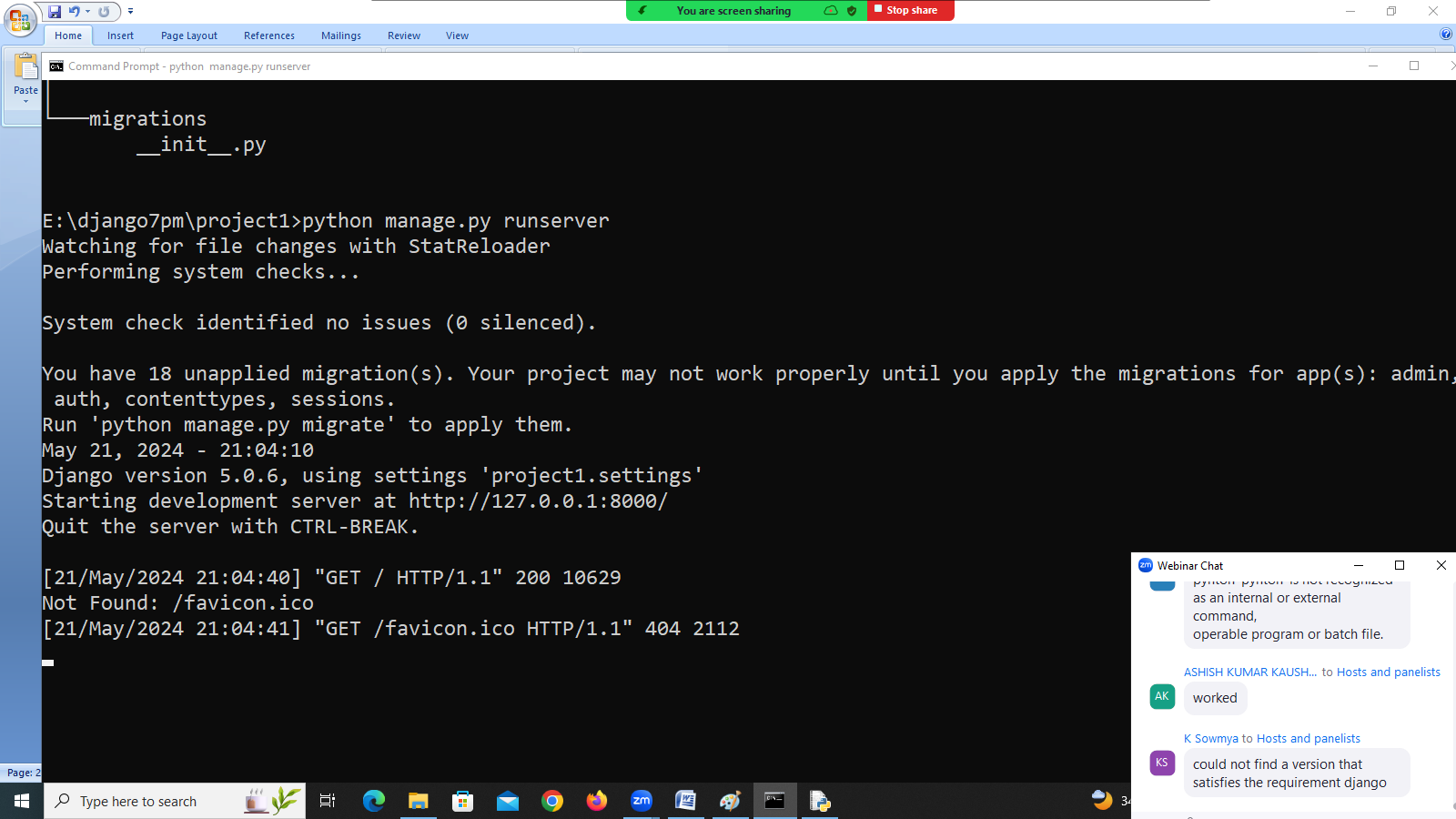
Every application is one package. This package contains number of modules (.py).

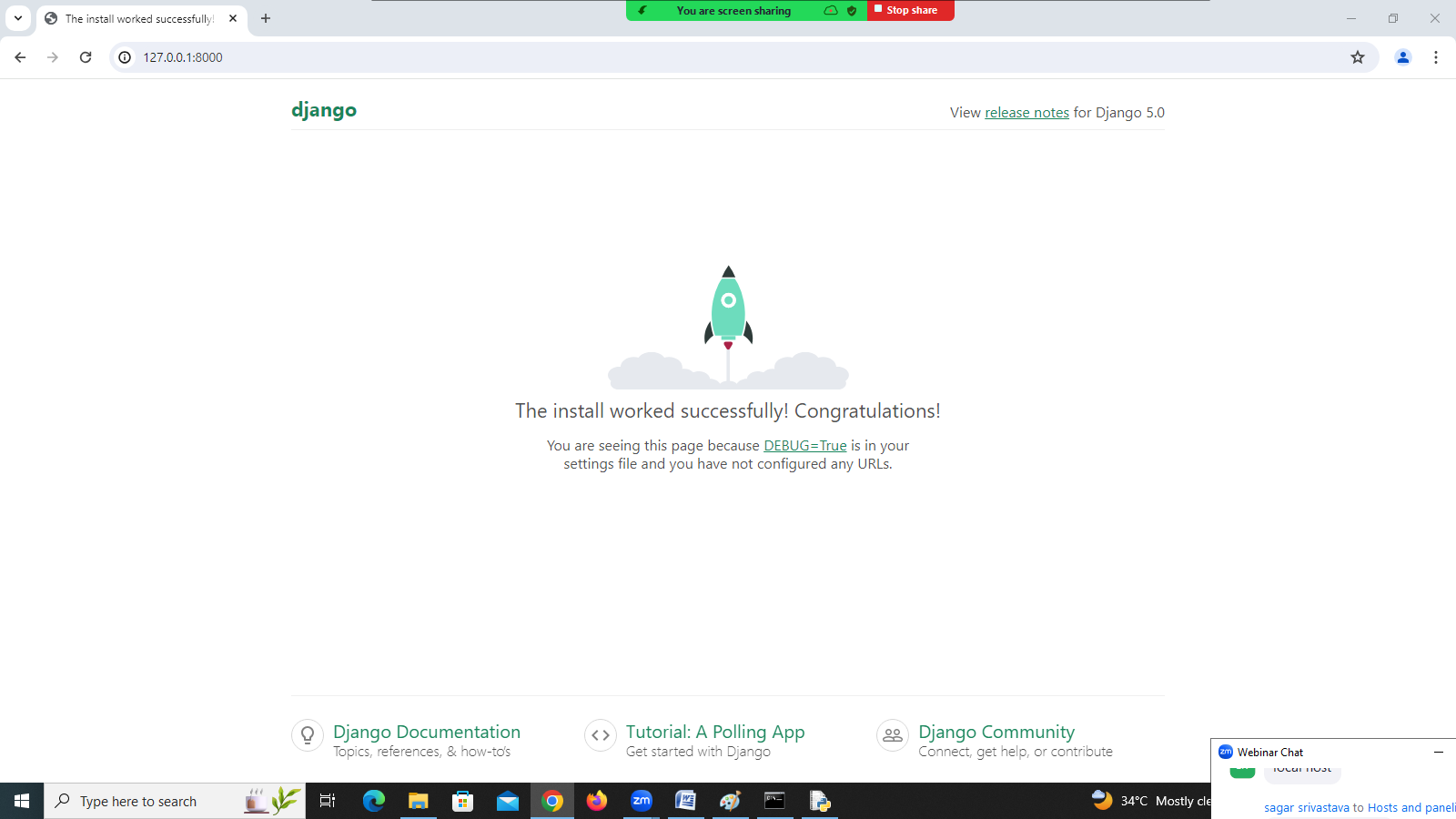
1. Admin.py : Used for registering models with admin
2. Apps.py : Configuring applications
3. Models.py : Used for developing models
4. Tests.py : Write test scripts
5. Views.py :
6. \_\_init\_\_.py : application is one package.

Django is framework which provides one design pattern MVT.

1. Model : Persistent Logic (Database)
2. View : View is a responsible for receive request from client and generating response.
3. Template : Template (Dynamic Page(HTML))

**Views.py**

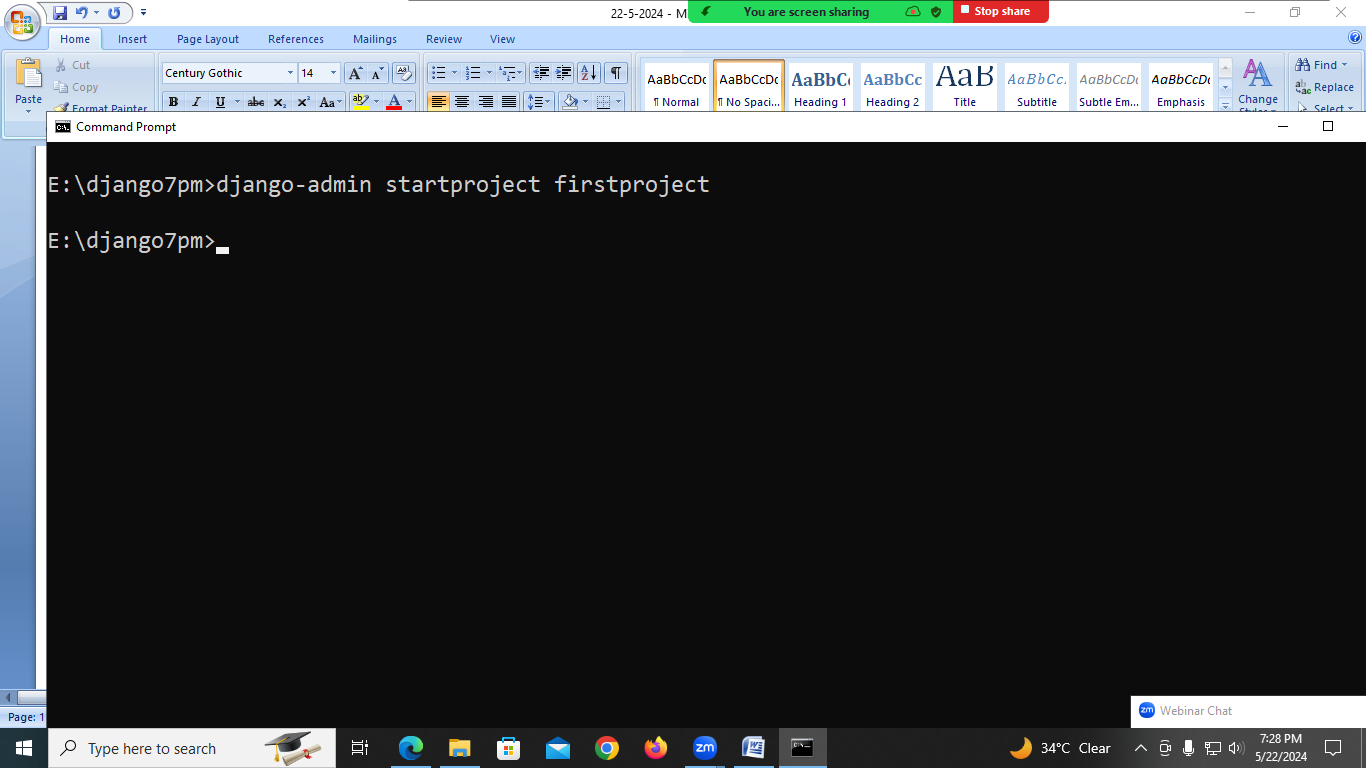
****



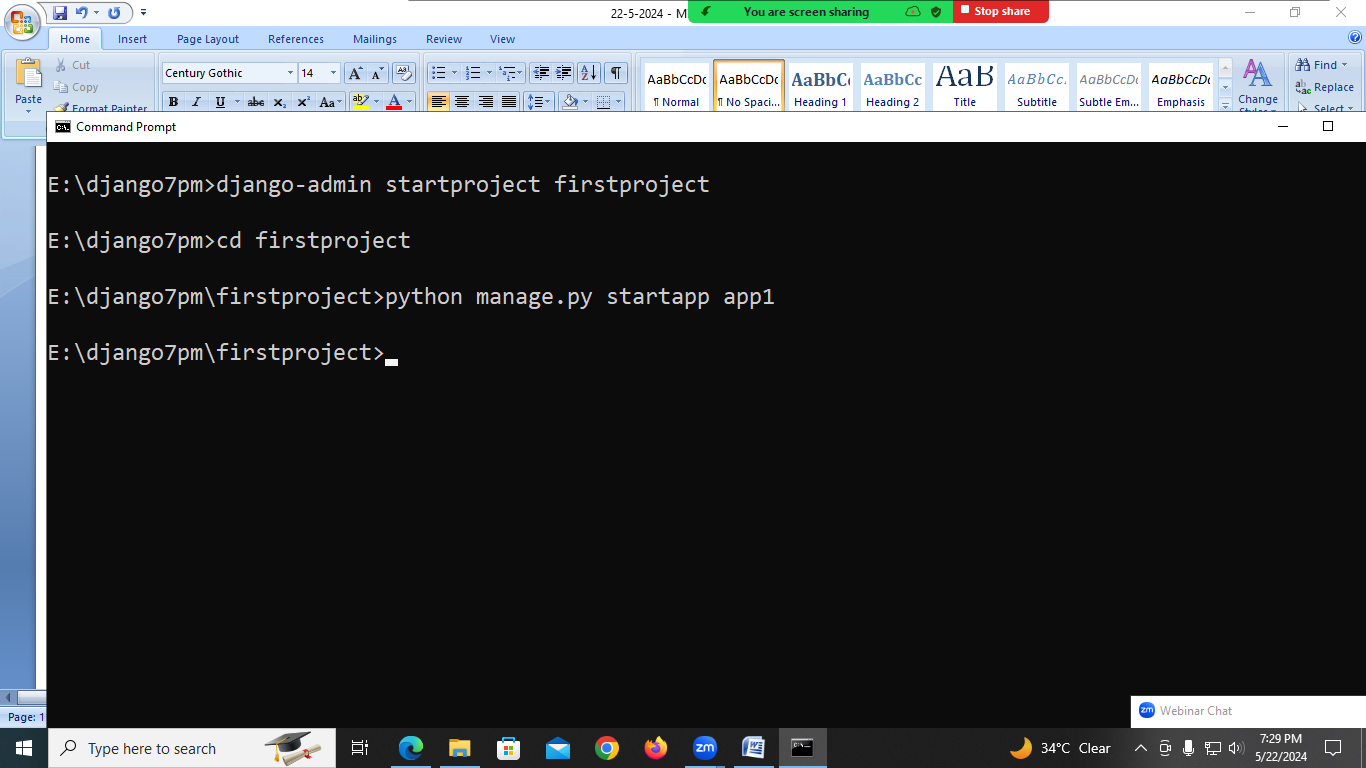
22-05-2024

**Basic Steps for creating django project**

1. Create a Project
2. Create a Application
3. Write a View
4. Registering an application
5. Run Server
6. Open Browser
7. Define URL
8. **Creating a Project**



1. **Open Project and Create Application**



1. **Open views.py**

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def hello(request):

msg="<h1>Hello Welcome To Django</h1>"

return HttpResponse(msg)

1. **Define URL pattern for view (urls.py) of project folder**

from django.contrib import admin

from django.urls import path

from app1 import views

urlpatterns = [

path("welcome/",views.hello),

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

]

1. **Register app with server/install application within server**

**Open settings.py (project folder)**

INSTALLED\_APPS = [

'app1',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

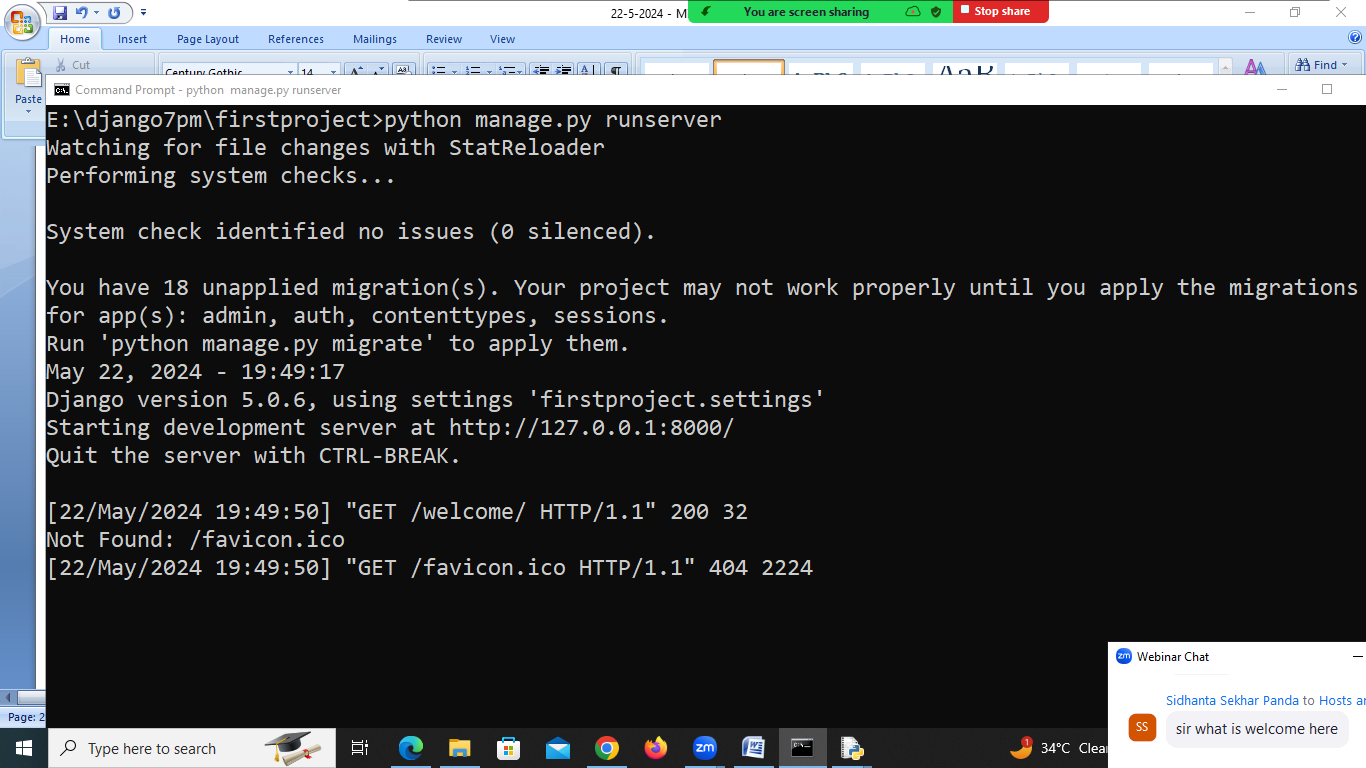
'django.contrib.sessions',

'django.contrib.messages',

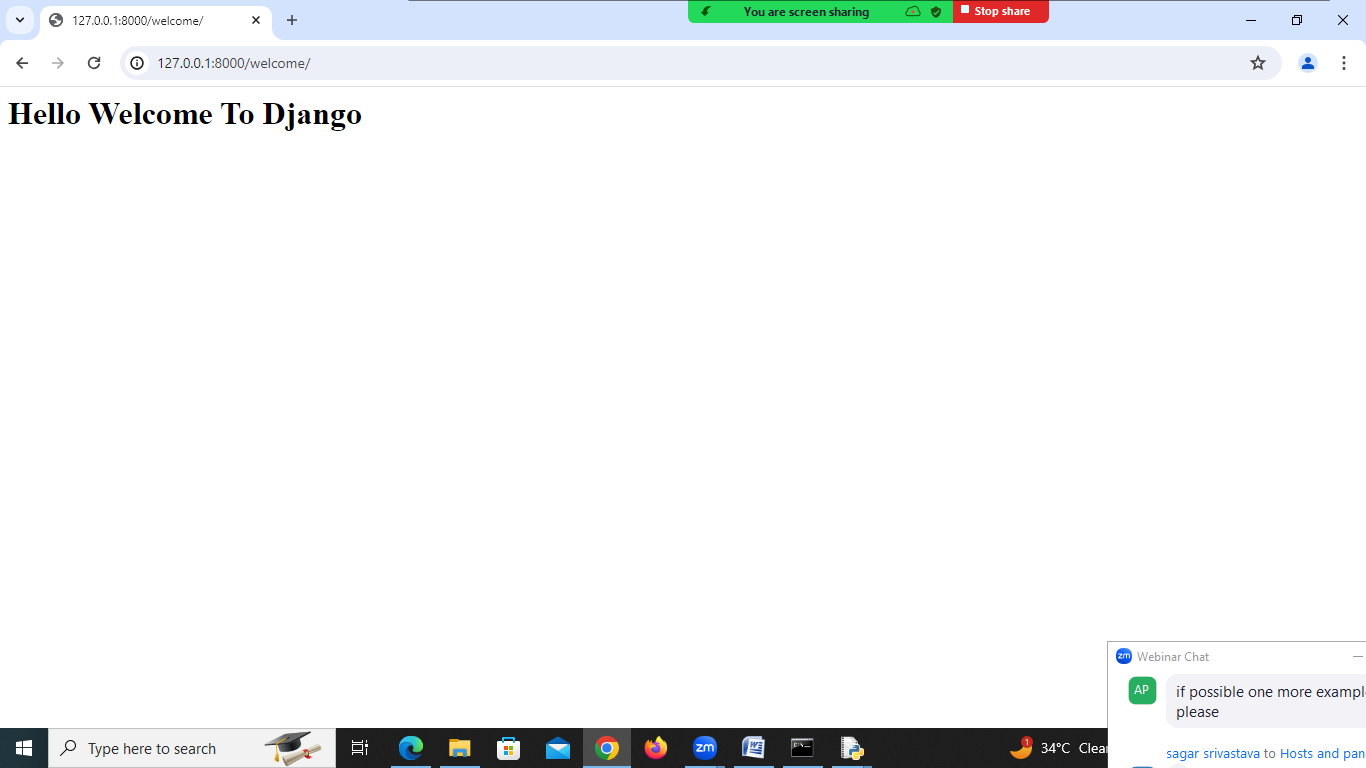
'django.contrib.staticfiles',

]

1. Start Server or Run Server



1. Open Browser



**What is View?**

A view is module. Which contains views, the responsibility of views is receiving request from client and generating response.

These views are developed in two ways.

1. Function based view
2. Class based view

In function based view, a function defined with one parameter request.

Syntax:

def <function-name>(request):

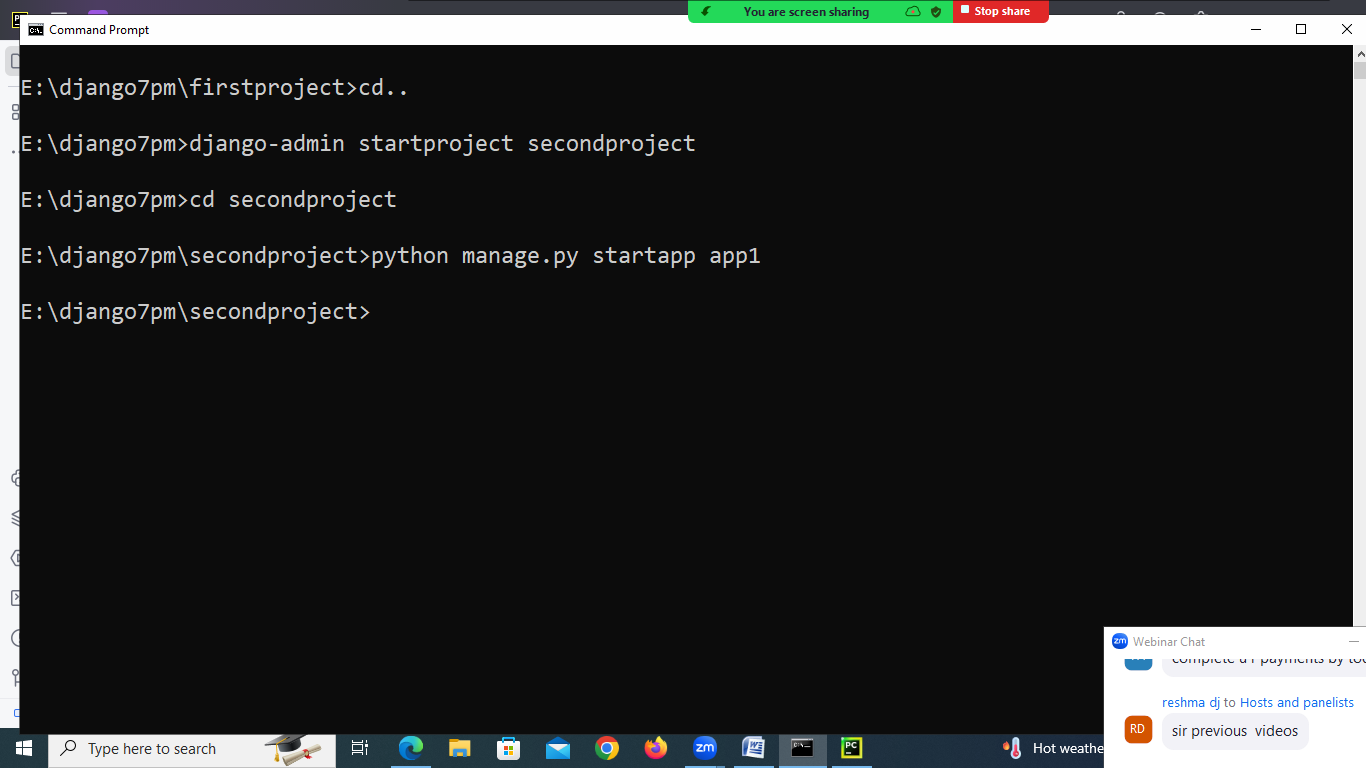
statement-1

statement-2

return HttpResponse(msg)

views.py module contains one or more view functions or classes.

**Project with multiple views**

****

* **Open pycharm**
* **Open project within pycharm**

**Viws.py**

from django.shortcuts import render  
from django.http import HttpResponse  
*# Create your views here.*def first(request):  
 return HttpResponse("<h1>This is from first View </h1>")  
  
def second(request):  
 return HttpResponse("<h1> This is from second view </h1>")  
  
def third(request):  
 return HttpResponse("<h1> This is from third view </h1>")

**urls.py (project folder)**

from app1.views import first  
from app1.views import second  
from app1.views import third  
urlpatterns = [  
 path('f1/',first),  
 path('f2/',second),  
 path('f3/',third),  
 path('admin/', admin.site.urls),  
]

**settings.py (project)**

INSTALLED\_APPS = [  
 'app1',  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]

[pythonbygupta@gmail.com](mailto:pythonbygupta@gmail.com)

Codewithsatishgupta

23-05-2024

Day-1 https://youtu.be/CjGybmNAPoEDay-2 https://youtu.be/rOrUGZ7\_9vUDay-3 https://youtu.be/3nGYOhUN7goDay-4 https://youtu.be/E5byXf1Y9I8Day-5 https://youtu.be/sOoxKF1WiukDay-6 <https://youtu.be/-bR9_vQsenM>

Set the following environment

1. Set path to scripts folder –(django-admin)
2. Set PYTHONPATH :

C:\Users\nit\AppData\Local\Programs\Python\Python312\Lib\site-packages

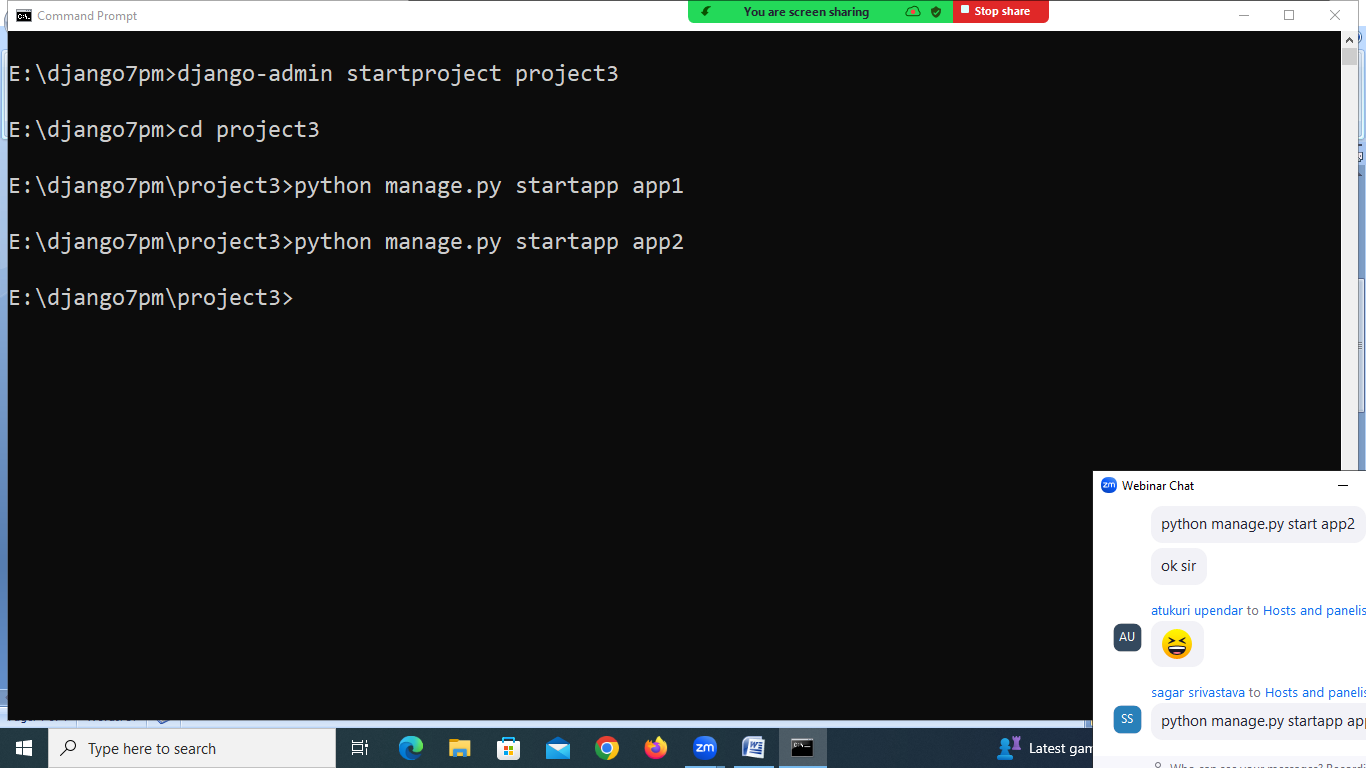
**Creating project with multiple applications**

A project contains one or more than one application.

Each application having one or more than one view.(views.py)

Each application is collection of modules.

1. Model (Database)
2. View (Request/Response)
3. Template (HTML OR Dynamic HTML)



**In app1 🡪 views.py**

from django.shortcuts import render  
from django.http import HttpResponse  
  
*# Create your views here.*def open\_account(request):  
 output='''<HTML>  
 <BODY>  
 <H1> WELCOME TO HDFC BANK </H1>  
 </BODY>  
 </HTML>'''  
 response=HttpResponse(output)  
 return response  
  
def close\_account(request):  
 output='''<HTML>  
 <BODY>  
 <H1> Thanks using services of HDFC BANK </H1>  
 </BODY>  
 </HTML>'''  
 response=HttpResponse(output)  
 return response

**in app2 🡪 views.py**

from django.shortcuts import render  
from django.http import HttpResponse  
*# Create your views here.*def home\_loan(request):  
 output = '''<HTML>  
 <BODY>  
 <H1> WELCOME TO HDFC BANK </H1>  
 </BODY>  
 </HTML>'''  
 response=HttpResponse(output)  
 return response

Install applications within server (OR) register applications with server

Settings.py of project3

INSTALLED\_APPS = [  
 'app1',  
 'app2',  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]

Define URL patterns for views at project level

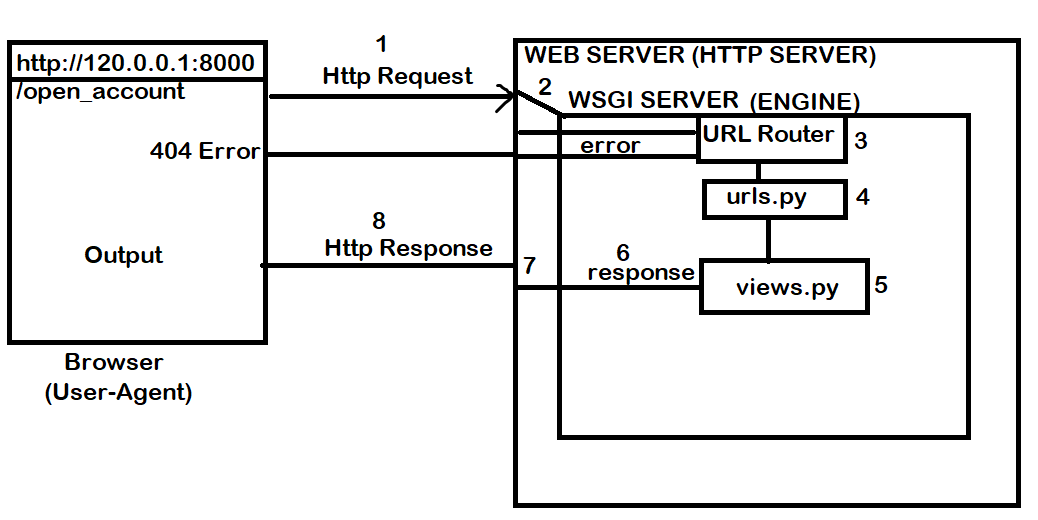
Open urls.py exists in project.

from app1.views import open\_account as oa  
from app1.views import close\_account as ca  
from app2.views import home\_loan as hl  
urlpatterns = [  
 path('open\_account/',oa),  
 path('close\_account/',ca),  
 path('home\_loan/',hl),  
 path('admin/', admin.site.urls),  
]

**Django web application execution flow**

Web server is a HTTP server whose responsibility is serving static resources. It is understand only HTTP. Web server receives request send by client.

WSGI server is responsible in execution of django application.



24-05-2024

**Web Server:**

1. **Nginx**
2. **Apache**

**WSGI Server (App Server)**

1. **Uwsgi**
2. **GUNICORN**

Django framework provides development server, database server,…

**URL Routing or URL configuration**

URL configuration is done using one module urls.py

Urls.py contains url-patterns for view functions or view classes.

View functions are server side, in order to access these function we need define url-pattern.

django.urls modules provides the following functions.

This url-pattern is define by using various functions.

1. path()
2. re-path()
3. include()

This module contains a list of urlpatterns.

**path()**

path() function returns urlpattern element or value.

**Syntax:**

path(url-pattern,view,kwargs=None,name=None)

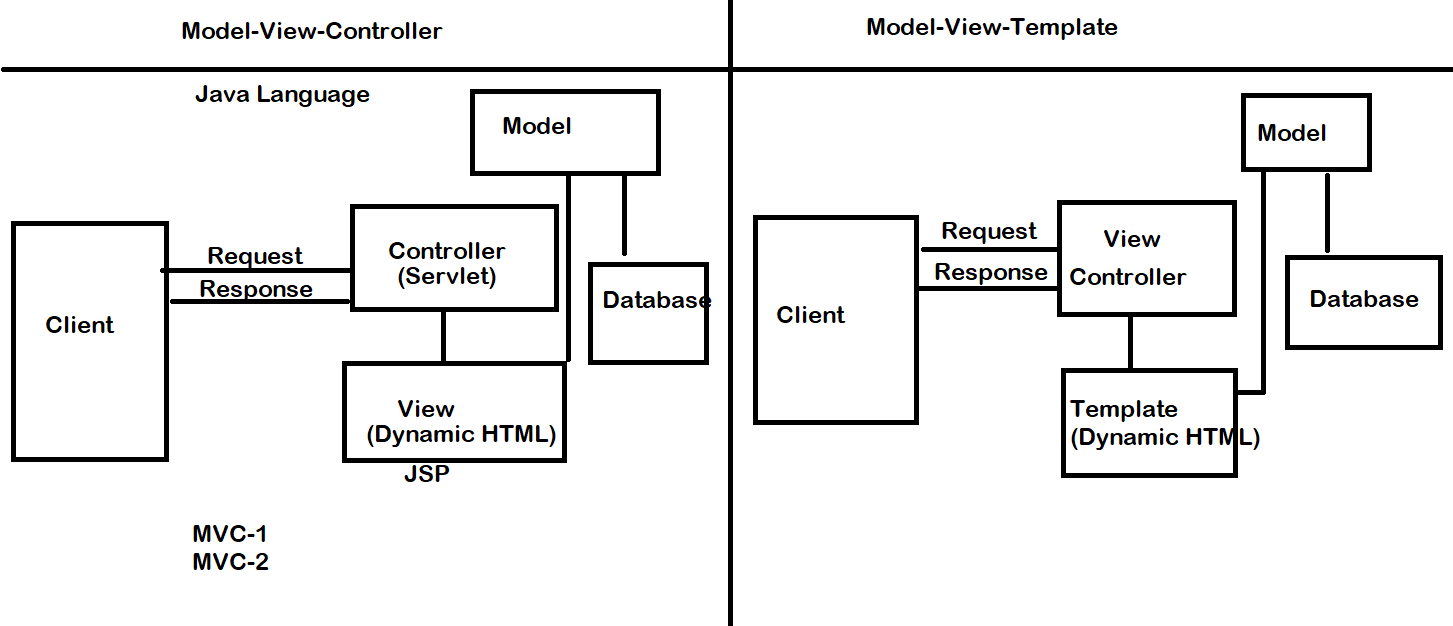
path function having total 4 parameters.

2 parameters are required.

1. Urlpattern 🡪 string
2. View 🡪 view function

Kwargs are optional arguments, these arguments are used to send values to view function.

name : view can accessed or called using name (private)



Views.py

from django.shortcuts import render  
from django.http import HttpResponse  
*# Create your views here.*def home(request):  
 output="<h1>This is Home Page </h1>"  
 response=HttpResponse(output)  
 return response  
  
def index(request):  
 output="<h1> This is index Page </h1>"  
 response=HttpResponse(output)  
 return response  
  
def content(request,value):  
 output=f'<h1> Value is {value} </h1>'  
 response=HttpResponse(output)  
 return response

urls.py

from app1.views import home,index,content  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('',home),  
 path('home/',home),  
 path('index/',index),  
 path('content/<value>',content)  
]

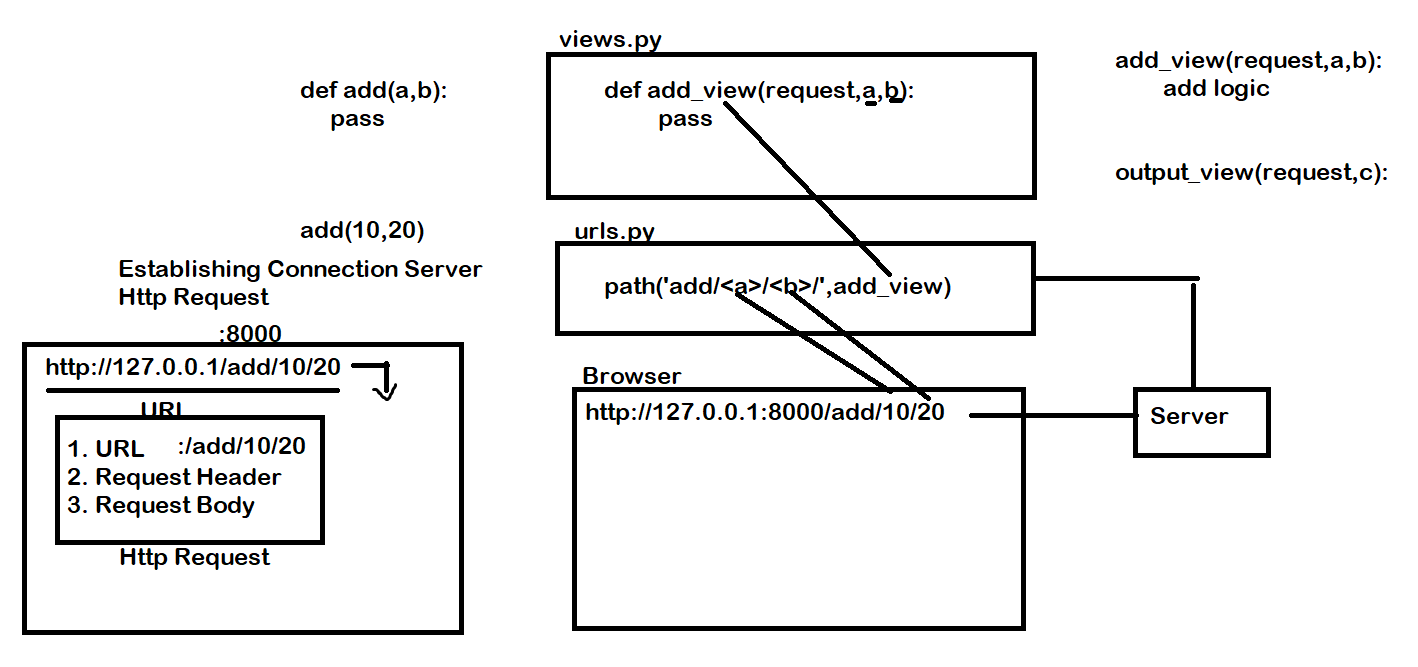
**settings.py**

INSTALLED\_APPS = [  
 'app1',  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]

25-05-2024

**Captured Parameters**

A view function receives values (OR) view function can be defined with more than one parameter. Except request parameters remaining parameters are called captured parameters. These parameters have to configure as part of url pattern.



**Syntax of captured parameters**

<param-name> 🡪 Generic parameter

<datatype:param-name> 🡪 parameter with specific data type

**Data types:**

1. Int
2. Float
3. Str/slug

**Views.py**

from django.http import HttpResponse  
import random  
*# Create your views here.*def otp\_view(request,name):  
 otp=random.randint(1000,9999)  
 output=f'''  
<HTML>  
<BODY>  
<H1> Hello {name} and your otp is {otp} </H1>  
</BODY></HTML>  
'''  
 response=HttpResponse(output)  
 return response  
  
def add\_view(request,a,b):  
 c=a+b  
 output=f"<h1> sum of {a} and {b} is {c} </h1>"  
 response=HttpResponse(output)  
 return response

**urls.py**

from app1.views import otp\_view,add\_view  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('otp\_view/<name>/',otp\_view),  
 path('add/<int:a>/<int:b>/',add\_view)  
]

**settings.py**

INSTALLED\_APPS = [  
 'app1',  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
]

Path(url-pattern,view,kwargs=None,name=None)

codewithsatishgupta

27-05-2024

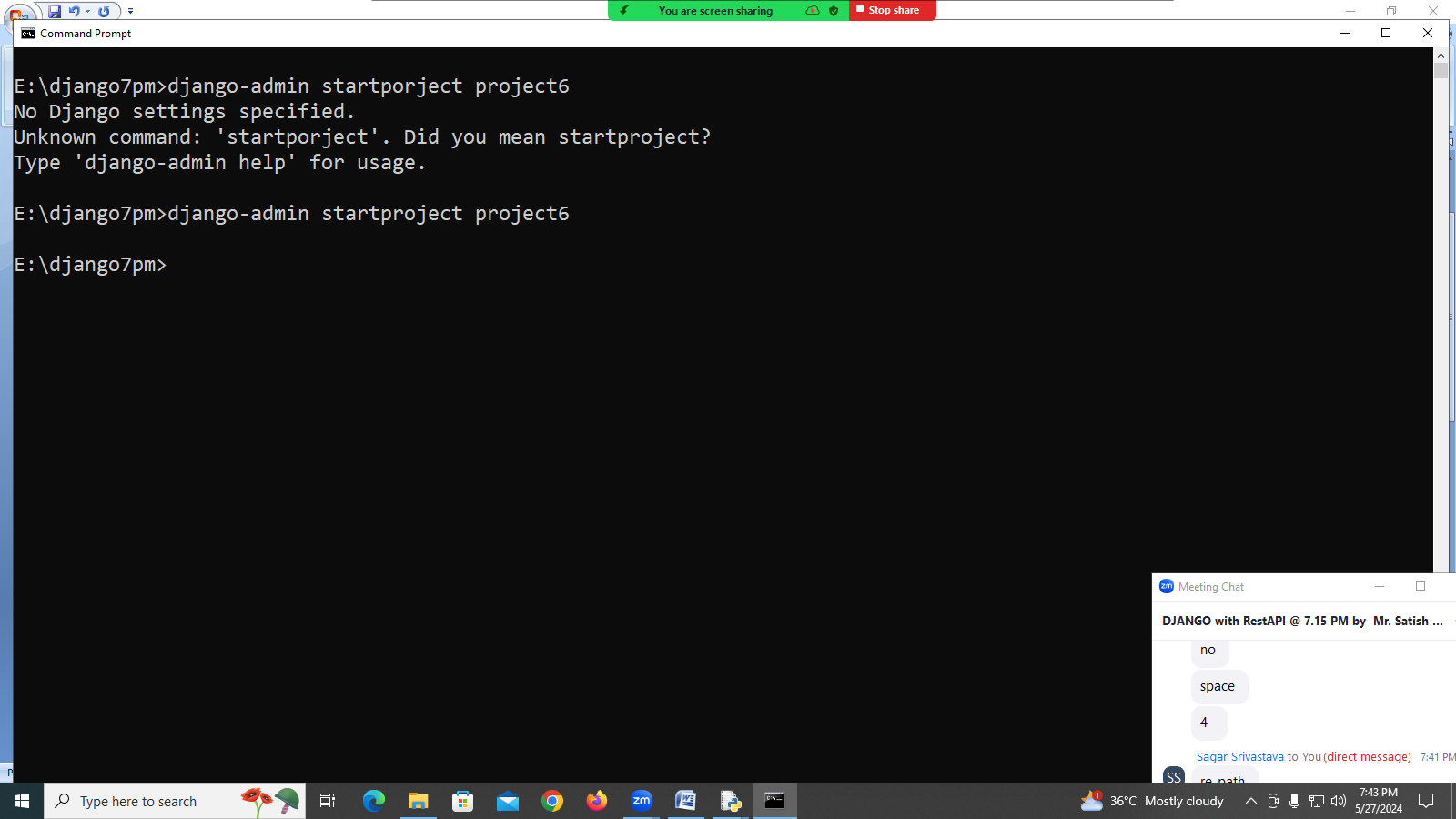
**re\_path**

This function is used to define url-pattern as regular expression pattern.

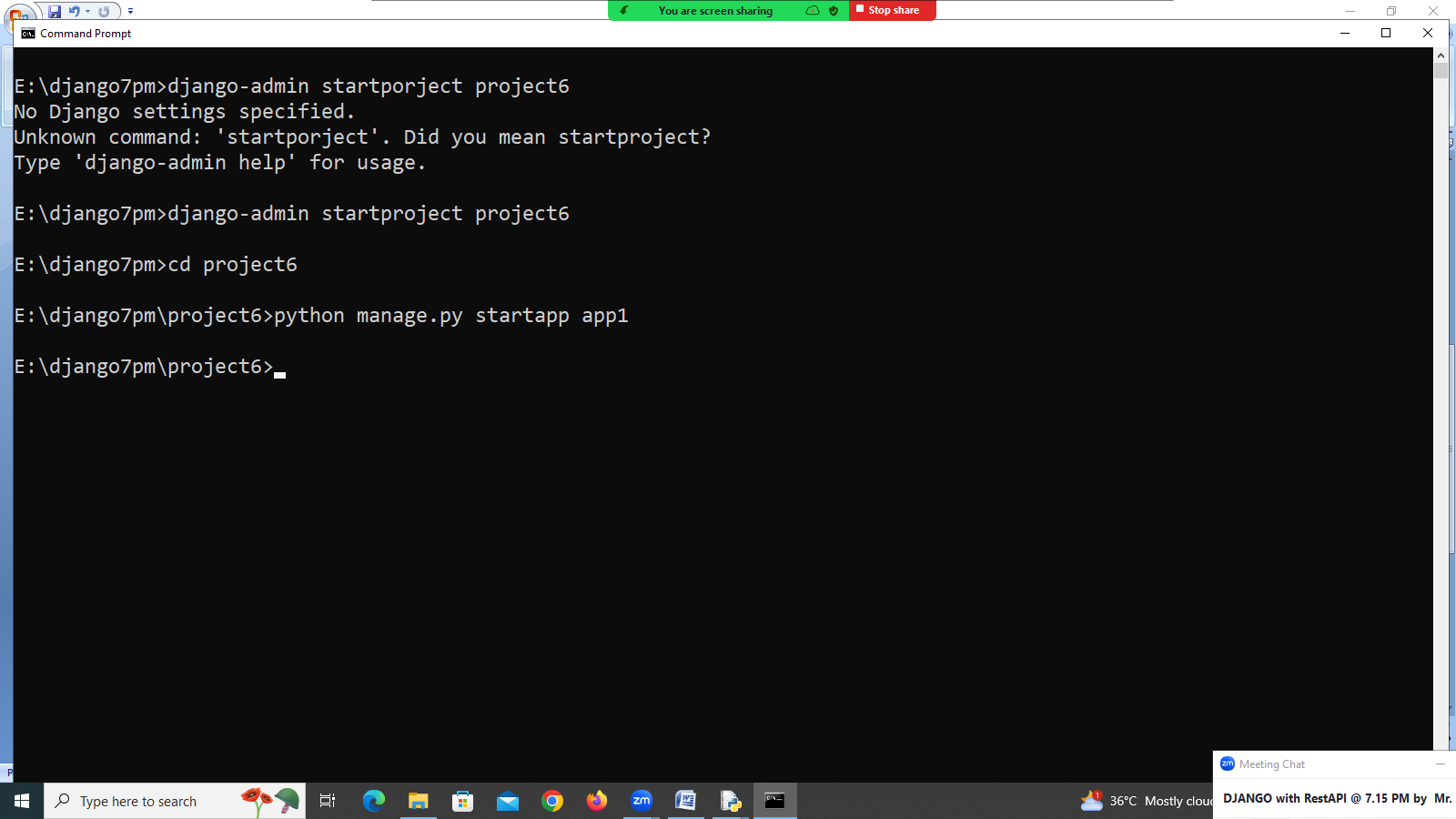
**Syntax: re\_path(url-pattern,view,kwargs=None,name=None)**

This function returns path element or url-pattern which is used as part of URL. re\_path() function internally uses “re” module for matching or searching pattern.

1. Create project



1. Create Application within project



1. Create a view (open application folder (views.py))

from django.shortcuts import render  
from django.http import HttpResponse  
*# Create your views here.*def index(request):  
 output="<h1> This is index View </h1>"  
 response=HttpResponse(output)  
 return response  
  
def home(request):  
 output="<h1> This is home view </h1>"  
 response=HttpResponse(output)  
 return response

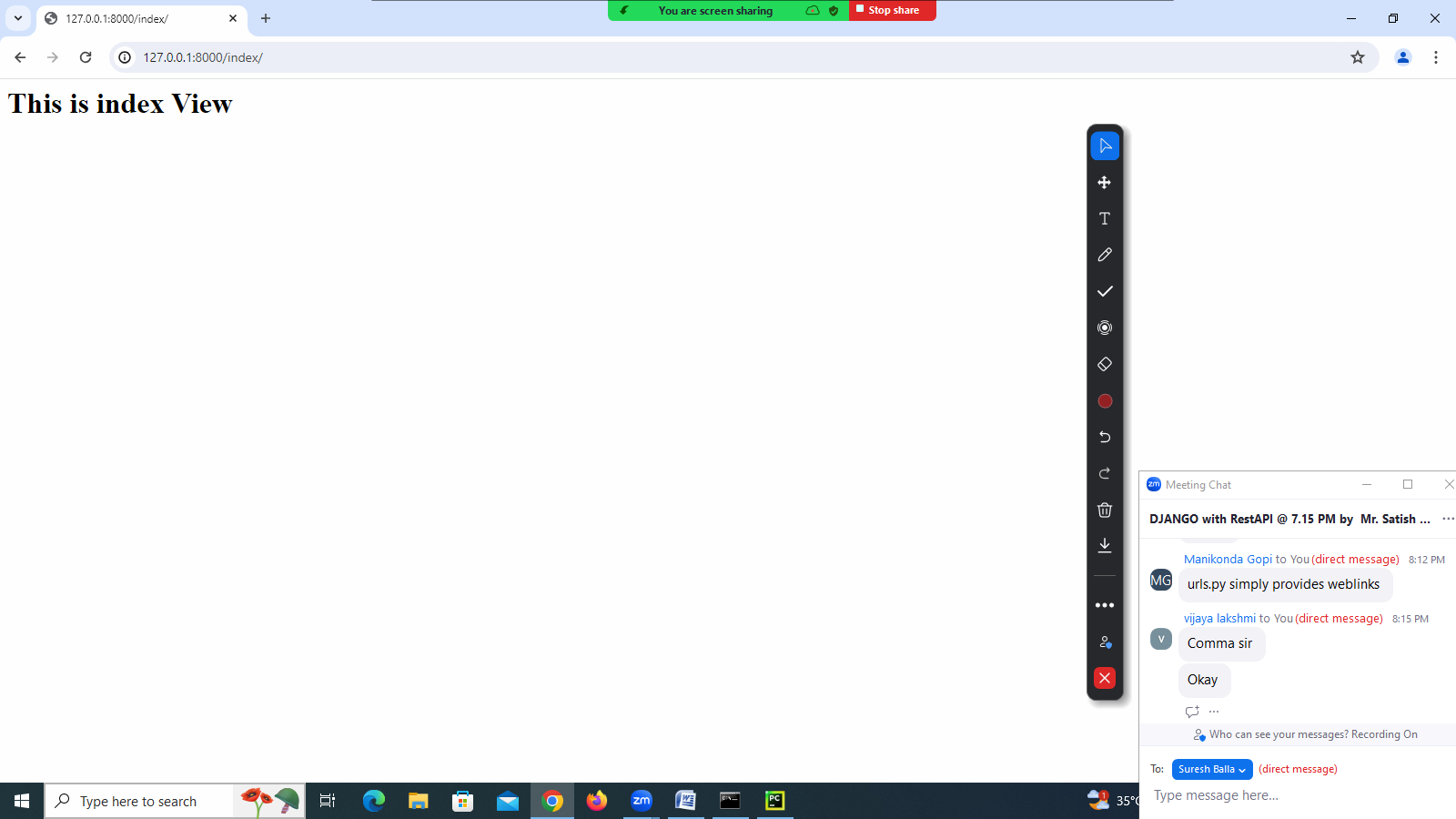
1. Configure url patterns in urls.py (project level)

from django.contrib import admin  
from django.urls import path,re\_path  
from app1.views import index  
from app1.views import home  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path(r'^index/$',index),  
 re\_path(r'^home/$',home)  
]

1. Open settings.py and install application

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'app1'  
]

1. Open Browser



**(?P<name>...)**

Similar to regular parentheses, but the substring matched by the group is accessible via the symbolic group name *name*. Group names must be valid Python identifiers, and in [bytes](file:///C:\\Users\\nit\\AppData\\Local\\Programs\\Python\\Python312\\Doc\\html\\library\\stdtypes.html" \l "bytes" \o "bytes) patterns they can only contain bytes in the ASCII range. Each group name must be defined only once within a regular expression. A symbolic group is also a numbered group, just as if the group were not named.

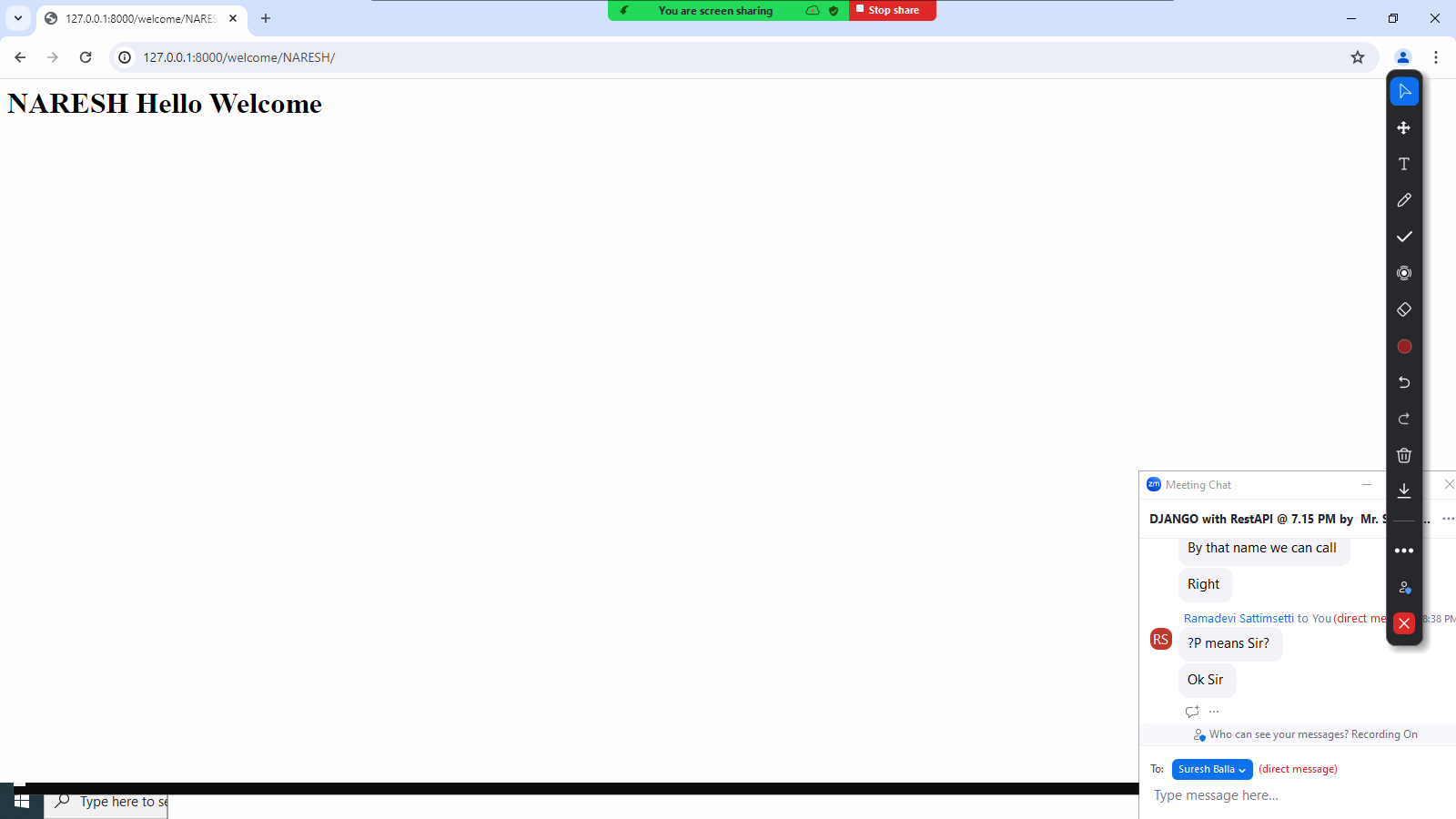
Inside views.py add another view function with parameter

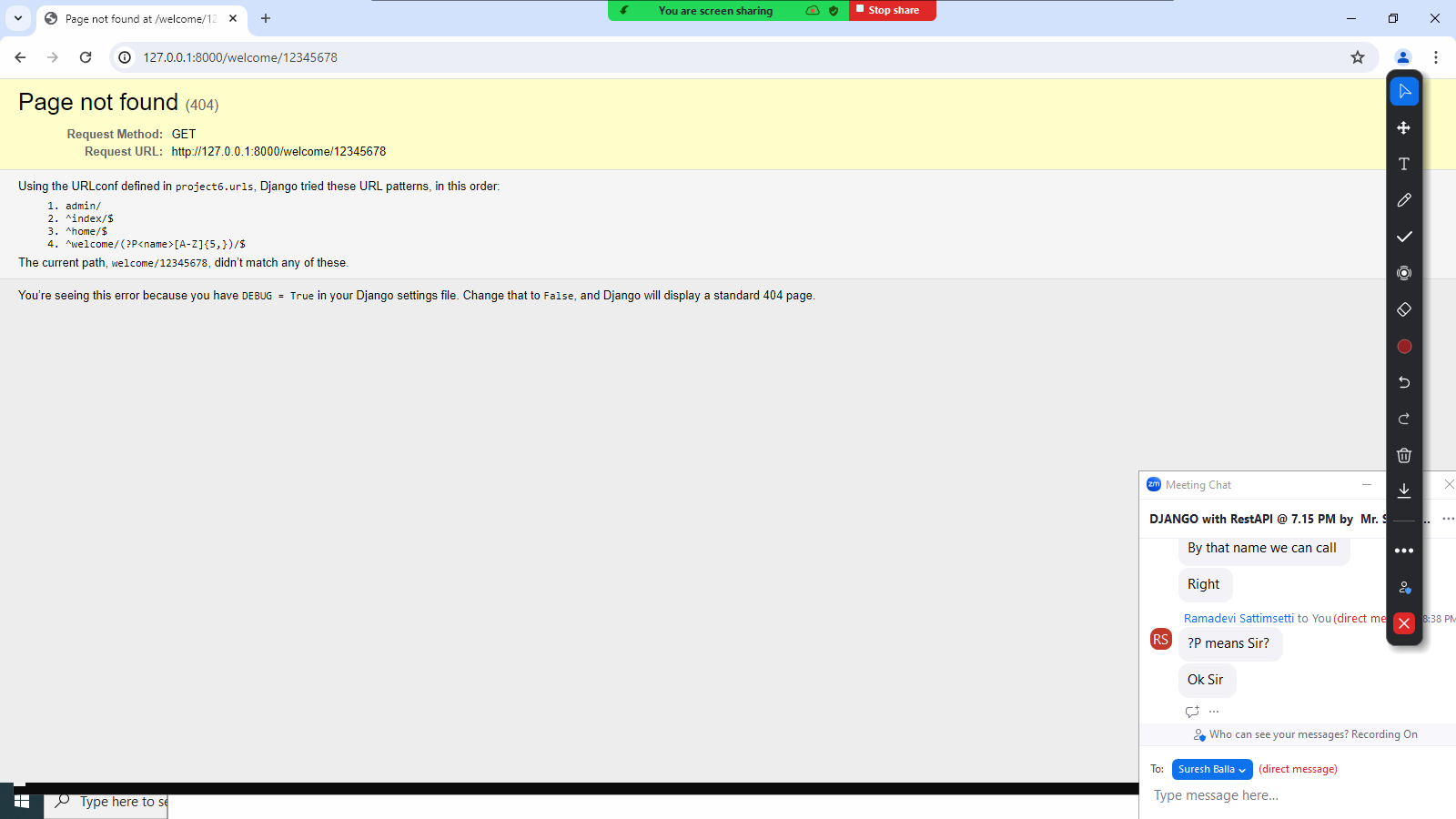
def welcome(request,name):  
 output=f"<h1> {name} Hello Welcome </h1>"  
 response=HttpResponse(output)  
 return response

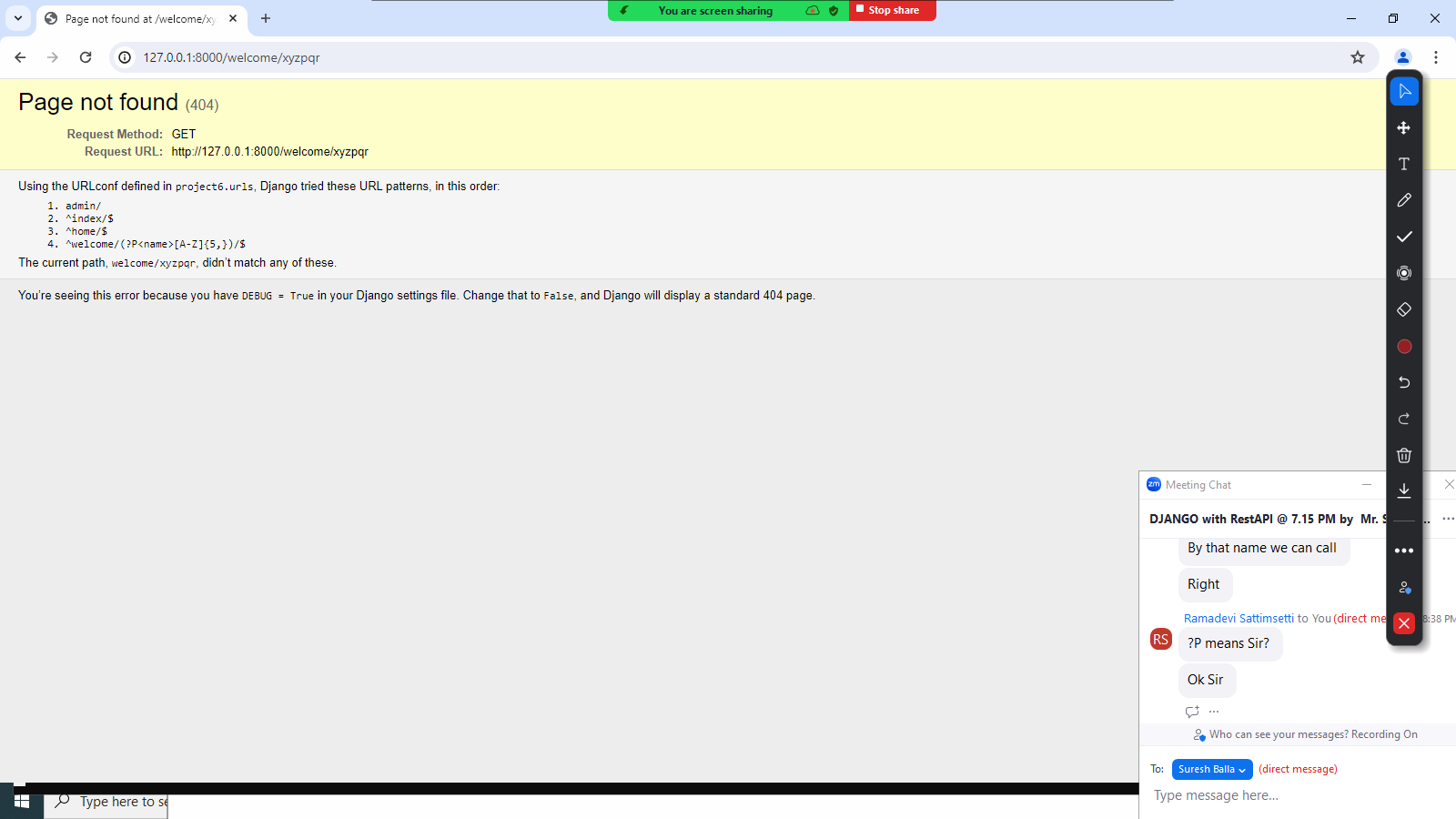
Inside url.py add another url pattern

urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path(r'^index/$',index),  
 re\_path(r'^home/$',home),  
 re\_path(r'^welcome/(?P<name>[A-Z]{5,})/$',welcome)  
  
]

**Open the browser**







**Home Work**

1. Develop view which receives date in the following format

dd-mm-yyyy

1. Develop view which receives uname in the following format

Alphabets, digits and one special character \_

**Defining url pattern at application level and including at project level**

Web server search for url configuration file (urls.py) inside project folder.

28-05-2024

Always URL configuration is done at project level. Web server search for url configuration file at project level. When client send request, the request URL, this request URL received by server it search this URL pattern inside URL configuration file (urls.py) defined inside project level.

**Defining URL patterns at application level**

URL configuration file can be created at application level (urls.py).

The urls.py at application level contains URL patterns of that application.

**Advantage of creating urls.py at application level**

1. Readability or Maintaining URL patterns easy
2. Modularity: dividing URL patterns into application level (urls.py)
3. Reusability : Reusability URL patterns in multiple projects

url configuration done at application level must included inside project level.

**include()**

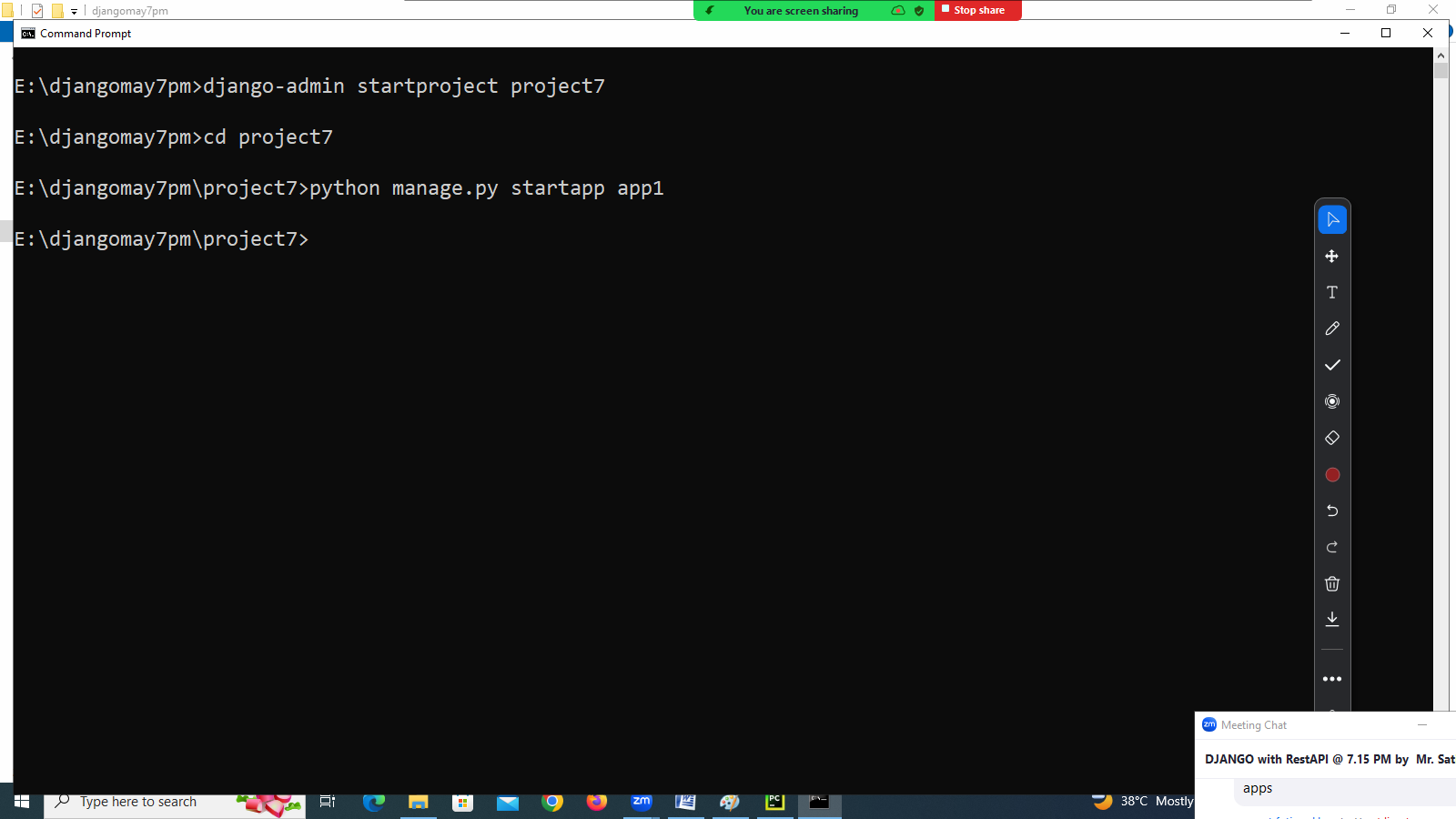
This function is used to include other url configuration file inside project level.

**At project Level (urls.py)**

**Syntax: path(url-pattern,view)**

**Example: path(“app1/”,include(“app1.urls”))**

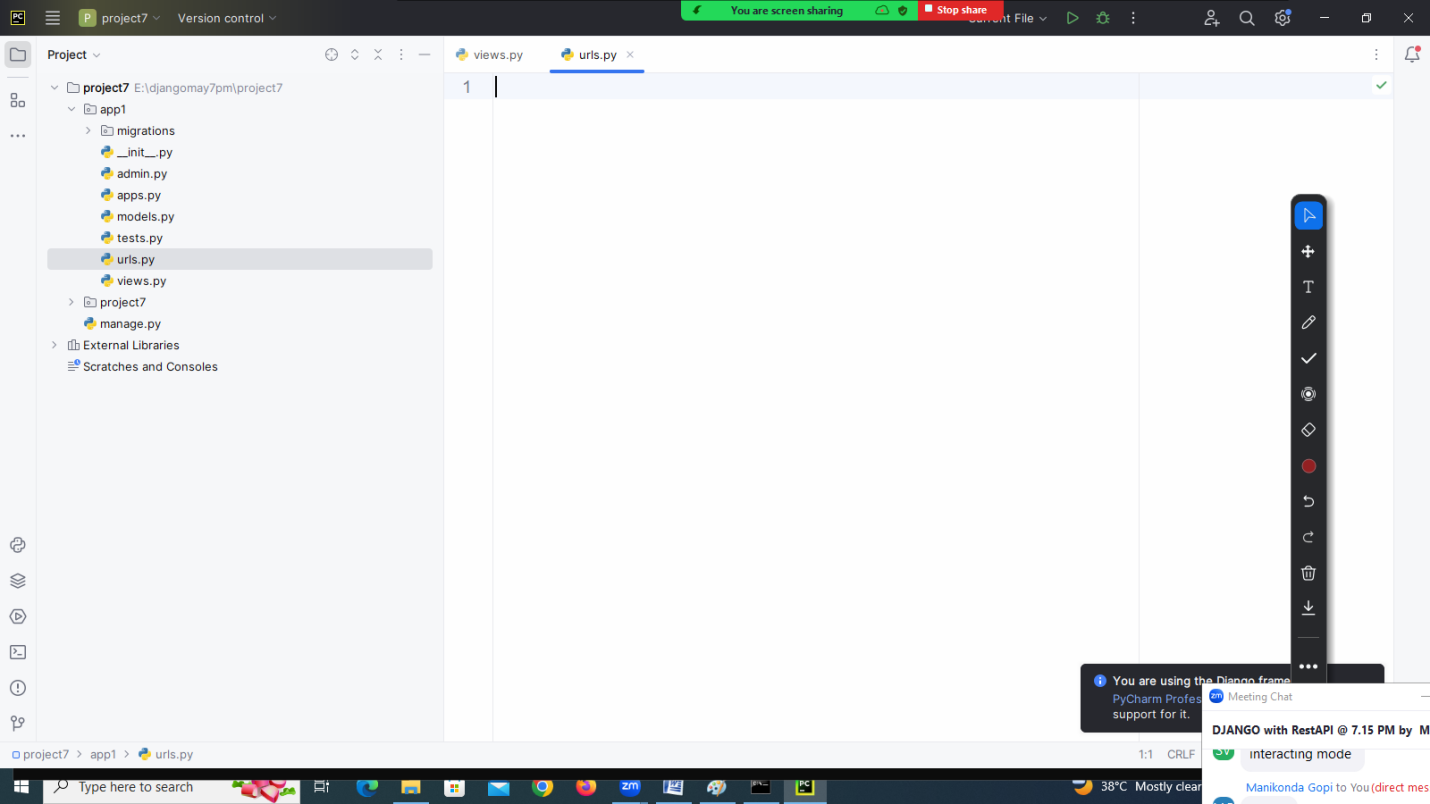
**path(“app2/”,include(“app2.urls”))**



**Views.py (app1)**

from django.shortcuts import render  
from django.http import HttpResponse  
*# Create your views here.*def stud\_adm(request):  
 output="<h1> Student Adm View </h1>"  
 response=HttpResponse(output)  
 return response   
  
def stud\_info(request):  
 output="<h1> Student Info View </h1>"  
 response=HttpResponse(output)  
 return response

**in app1 create one module urls.py**



from django.urls import path  
from . import views  
urlpatterns = [  
 path('adm/',views.stud\_adm),  
 path('info/',views.stud\_info)  
]

**Open urls.py at project level**

from django.contrib import admin  
from django.urls import path,include  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('app1/',include('app1.urls'))  
]

**open settings.py**

install application

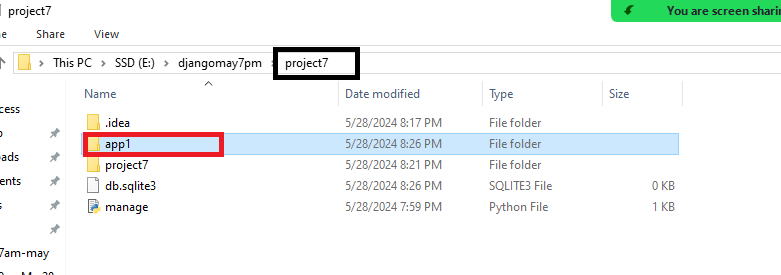
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'app1'  
]

**How to include urls.py at application level inside another project**

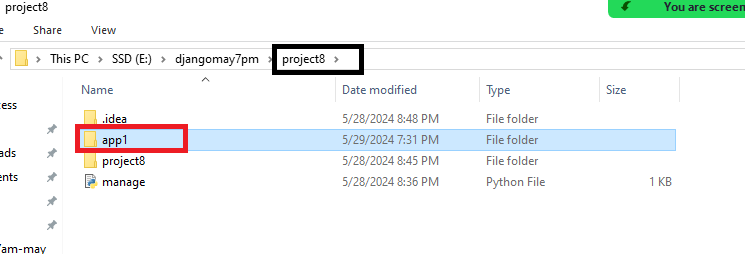
This allows including application of one project inside another project.

Basic steps for including application of one project inside another project.

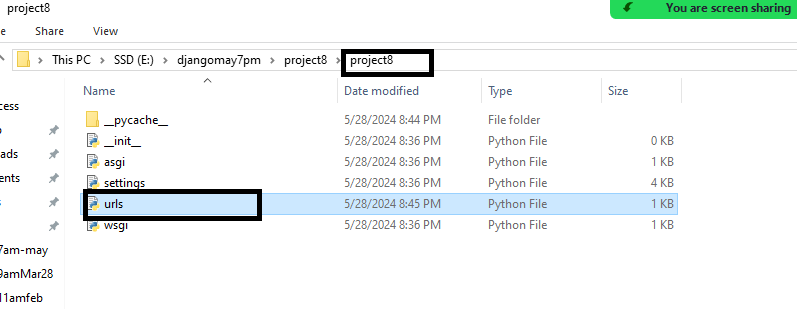
1. Open project7



1. Copy app1 of project7 into project8



Open urls.py at project level (project8)



from django.contrib import admin

from django.urls import path,include

urlpatterns = [

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

path('app1/',include('app1.urls'))

]

**Settings.py**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'app1',

]

**Template**

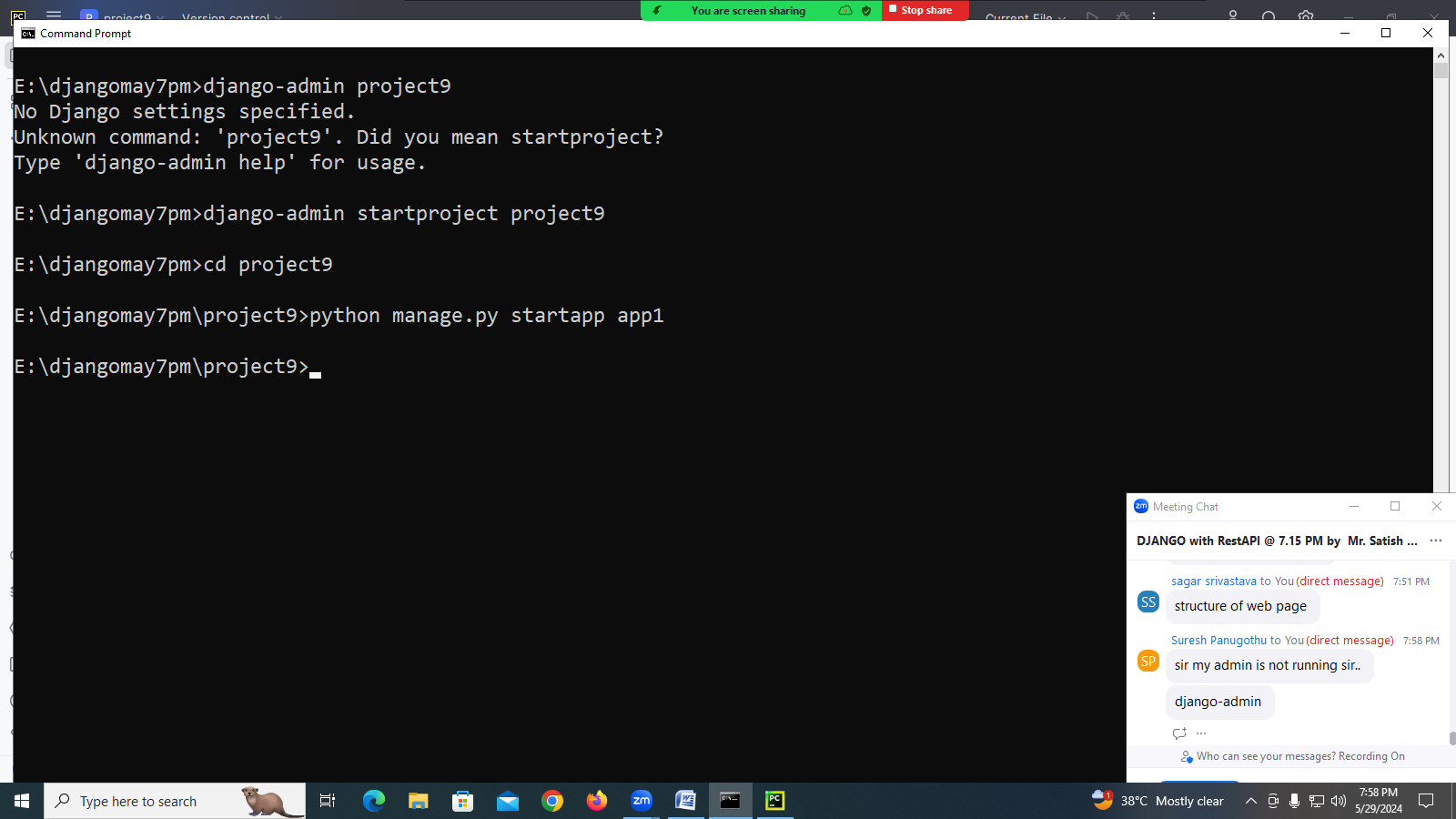
Django application is developed using one design pattern MVT (Model-View-Template).

View 🡪 View Responsibility is receiving the request and generating response. Always response is generated in HTML.

HTML 🡪 Hyper Text Markup Language, it is used for defining the structure of webpage. HTML is used for two things

1. Presentation Data or information
2. User Interface

**Example**



**Views.py**

import datetime  
from django.http import HttpResponse  
def display(request):  
 dt=datetime.datetime.today()  
 d=dt.date()  
 h=dt.hour  
 if h<=12:  
 msg="<h1> Good Morning </h1>"  
 elif h<=16:  
 msg="<h1> Good Evening </h1>"  
 elif h<=24:  
 msg="<h1> Good Night </h1>"  
 output=f'''<HTML>  
 <BODY>  
 <h1> Current Date {d} --> {msg} </h1>   
 <a href="http://www.nareshit.com">NARESHIT</a><br>  
 </BODY>  
 </HTML>'''  
 return HttpResponse(output)

**urls.py 🡪 project level**

from django.contrib import admin  
from django.urls import path  
from app1.views import display  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('disp/',display)  
]

**settings.py 🡪 project level**

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'app1'  
]

**The problem with above application is,**

1. HTML code is included with views.py (Business Logic and Presentation Logic both are combined)
2. Maintaining code becomes complex
3. There is no preview is available
4. Developing HTML is becomes complex

**Template:**

**What is template?**

Template is file which contains static content and dynamic content.

**What is static content?**

This static content is represented using HTML. HTML is used for generating static web pages.

<html>

<body>

<h1> Hello Welcome NARESH </h1>

</body>

</html>

**What is dynamic content?**

<html>

<body>

<h1> Hello welcome {{ }} </h1>

</body>

< /HTML>

DTL 🡪 Django Template Language, it is standard given by django for developing django template engine. JINGA2 it popular DTL engine.

Template is a file static content and django template language.

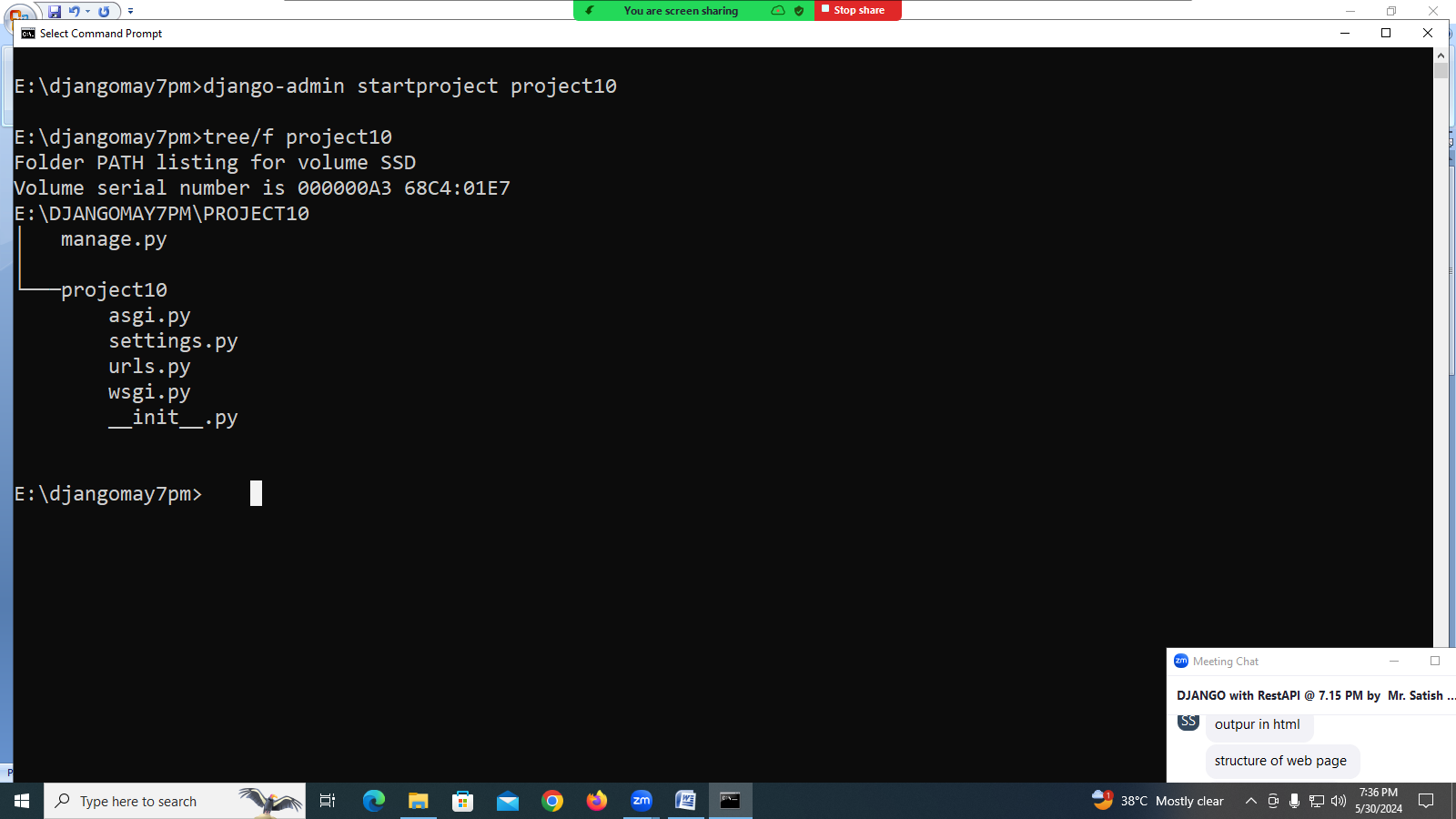
Template is a collection HTML tags and DTL.

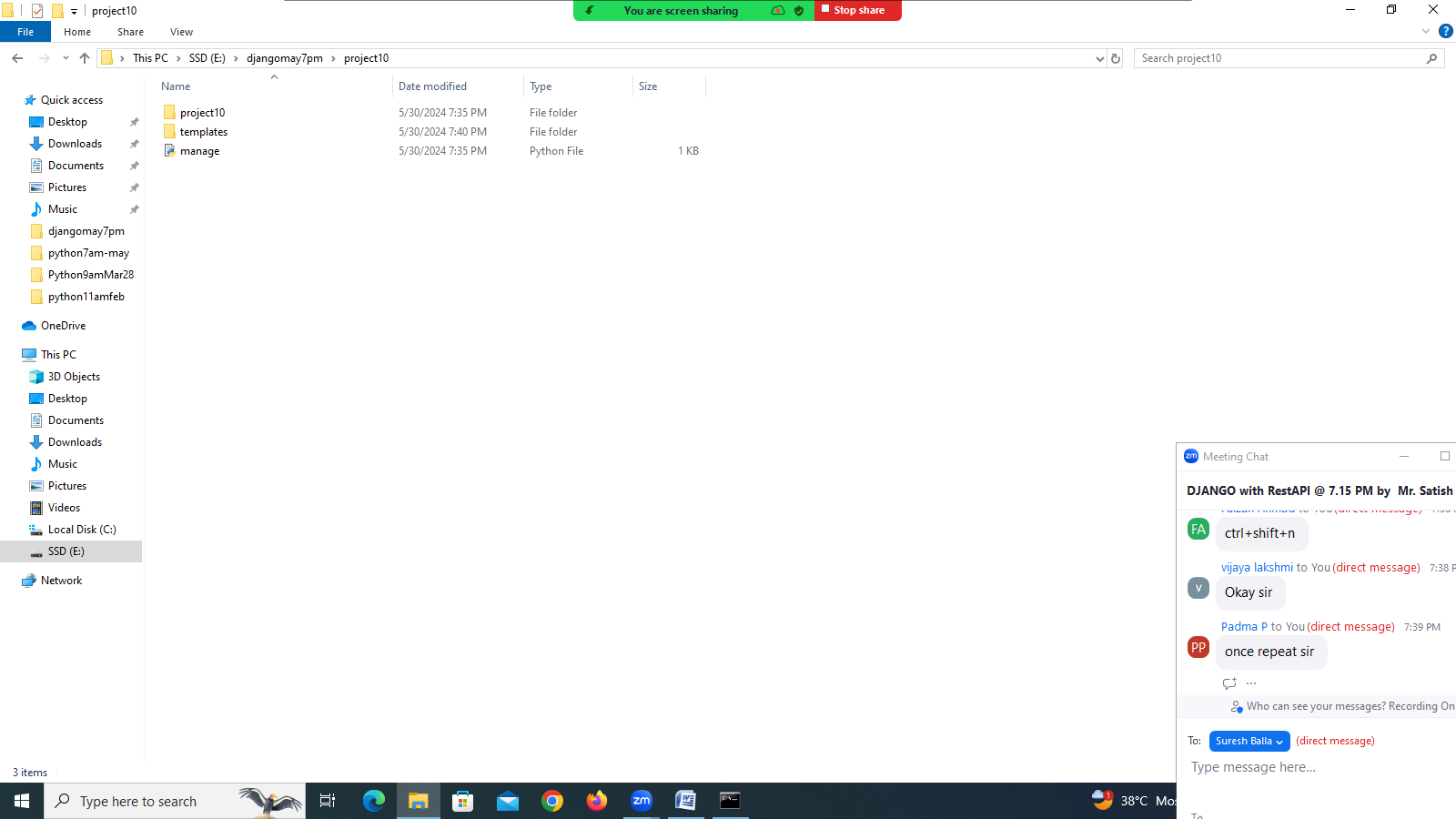
**Advantage of template**

1. Separation of business logic and presentation logic
2. Presentation logic can be changed without doing changes in business logic (Marinating code becomes easy)
3. Simultaneously more than one person can work with project

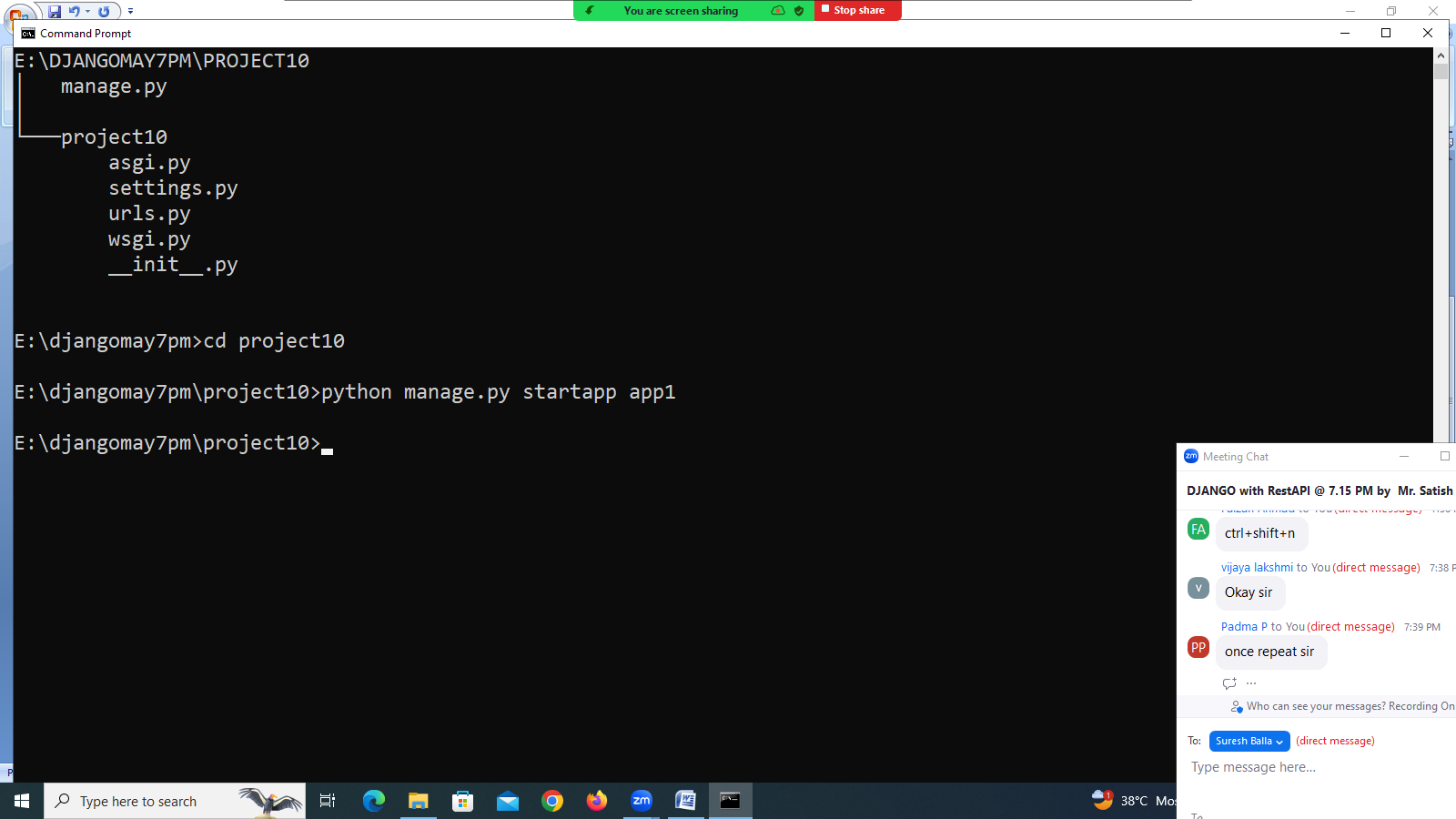
**Basic steps for creating template**

1. Create templates folder.
2. Create this folder inside project container directory/folder

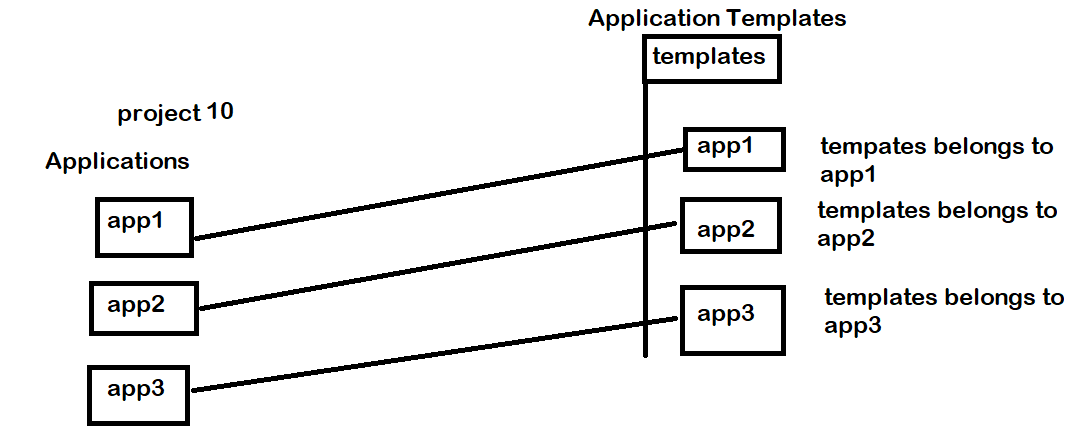


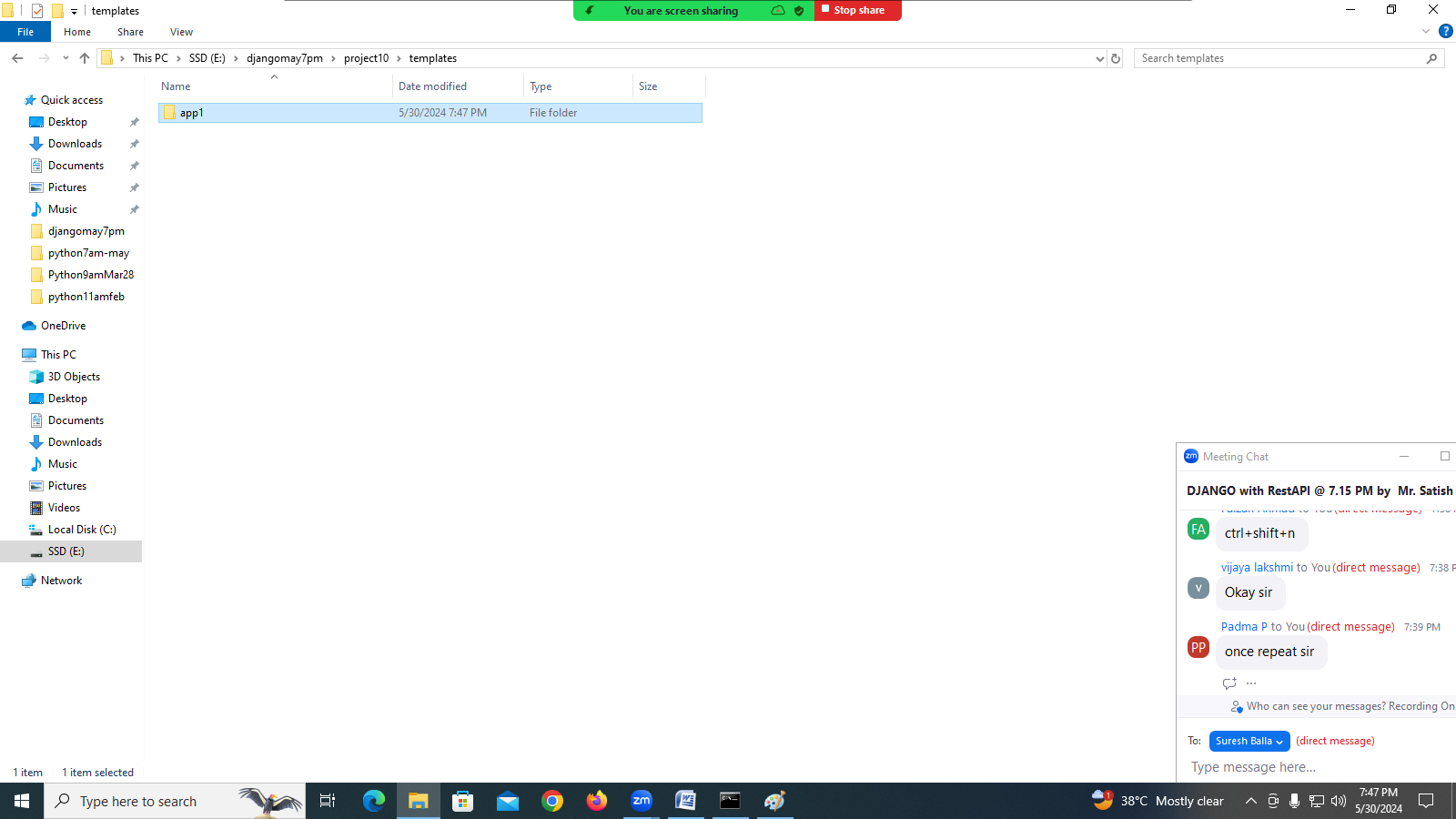


1. Create application

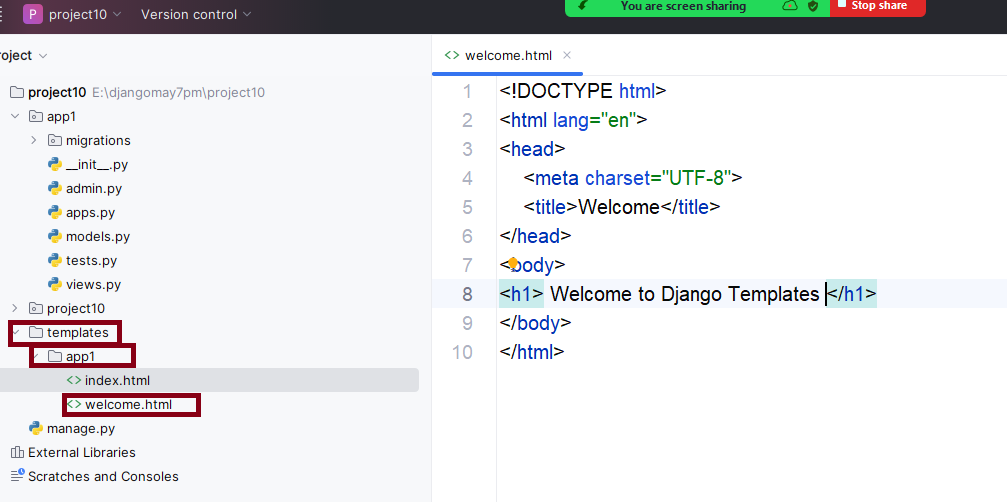


1. Create folder inside templates folder with application name





1. Inside app1 of templates folder create template (html file)



1. Open views.py of app1

from django.shortcuts import render  
  
*# Create your views here.*def welcome(request):  
 response=render(request,'app1/welcome.html')  
 return response

1. Inside urls.py configure url-pattern for view function

from app1.views import welcome  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('welcome/',welcome)  
]

1. Install application in settings.py

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'app1'  
]

1. Configure template folder (template path) settings.py

TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': ['E:\\djangomay7pm\\project10\\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',  
 ],  
 },  
 },  
]

**How to generate dynamic path?**

**In settings.py**

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
TEMPLATE\_DIR=BASE\_DIR/'templates'

TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 **'DIRS': [TEMPLATE\_DIR],** 'APP\_DIRS': True,

**What is \_\_file\_\_?**

\_\_file\_\_ is a magic variables or attribute.

Each module is having attributes named \_\_file\_\_

This attribute hold location where module is saved or available (own path)

\_\_file\_\_ attribute returns string

**What is Path?**

Path is a class exists in pathlib.

Path is used to created Path object, which is platform independent.

Using path object we can manipulate path.

Path(\_\_file\_\_) 🡪 this returns path of settings.py

**What is resolve()?**

Resolve is a method of Path object

By resolving path object, we can apply path functions.

.parent 🡪 move to parent folder/directory

**What is BASE\_DIR?**

BASE\_DIR environment variable, which locate to container folder.

31/05/2024

**render() function**

It is a function of shortcuts of django package.

This function returns HttpResponse object

This function returns HttpResponse object by including,

1. Template
2. Context
3. Request

**Syntax: render(request,template,context=None,content-type=None)**

**request**

The request object used to generate this response.

**template\_name**

The full name of a template to use or sequence of template names. If a sequence is given, the first template that exists will be used.

**Optional**

**context**

A dictionary of values to add to the template context. By default, this is an empty dictionary. If a value in the dictionary is callable, the view will call it just before rendering the template.

**content\_type**

The MIME type to use for the resulting document. Defaults to 'text/html'.

**Django Template Language (DTL)**

Django template language is markup language. This language is embedded within HTML. This language is used for generating dynamic content.

Django Template Language syntax used is Jinja2.

Django Template Language provides,

1. Variables
2. Tags
3. Filters
4. ….

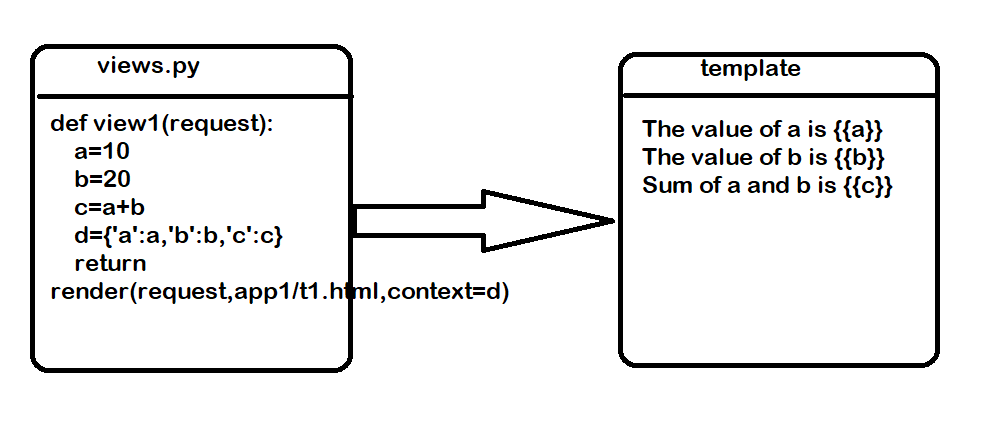
**Variables**

The variables or context variables can be inserted by using variables syntax provided DTL.

Syntax: {{variable-name}}

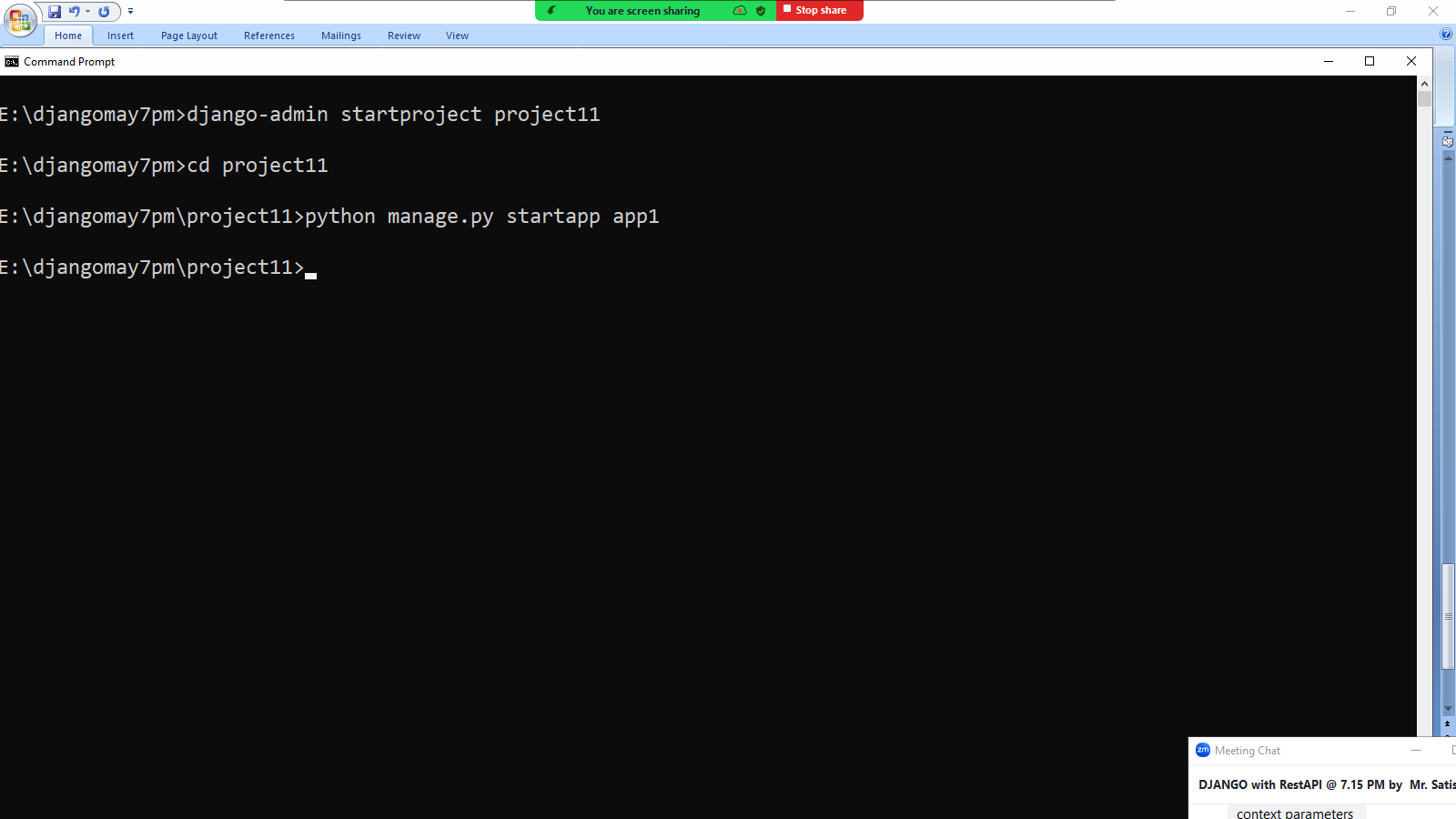
These variables are sending from view to template using context.

Context is dictionary which is having key and value. Variable names are nothing but keys.

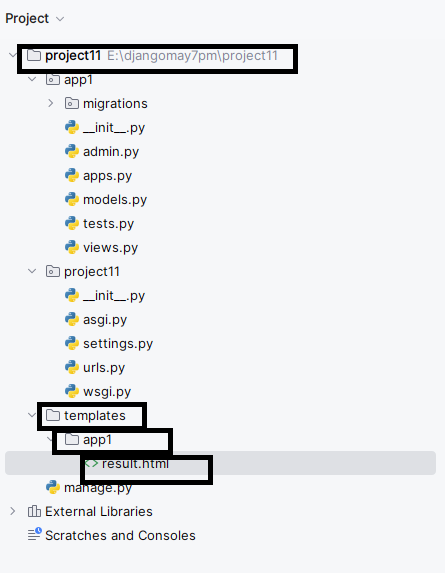


**Example**

1. Create Project
2. Create Application



1. Create templates
2. Inside template folder create app1



**result.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Result</title>  
</head>  
<body>  
 <h1> Sum of Two Numbers</h1>  
 <h2> First Number {{num1}}</h2>  
 <h2> Second Number {{num2}}</h2>  
 <h2> Sum is {{result}}</h2>  
</body>  
</html>

**Views.py**

from django.shortcuts import render  
  
*# Create your views here.*def add(request):  
 num1=10  
 num2=20  
 result=num1+num2  
 d={'num1':num1,'num2':num2,'result':result}  
 return render(request,'app1/result.html',context=d)

**urls.py at project level**

from app1.views import add  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('add/',add)  
]

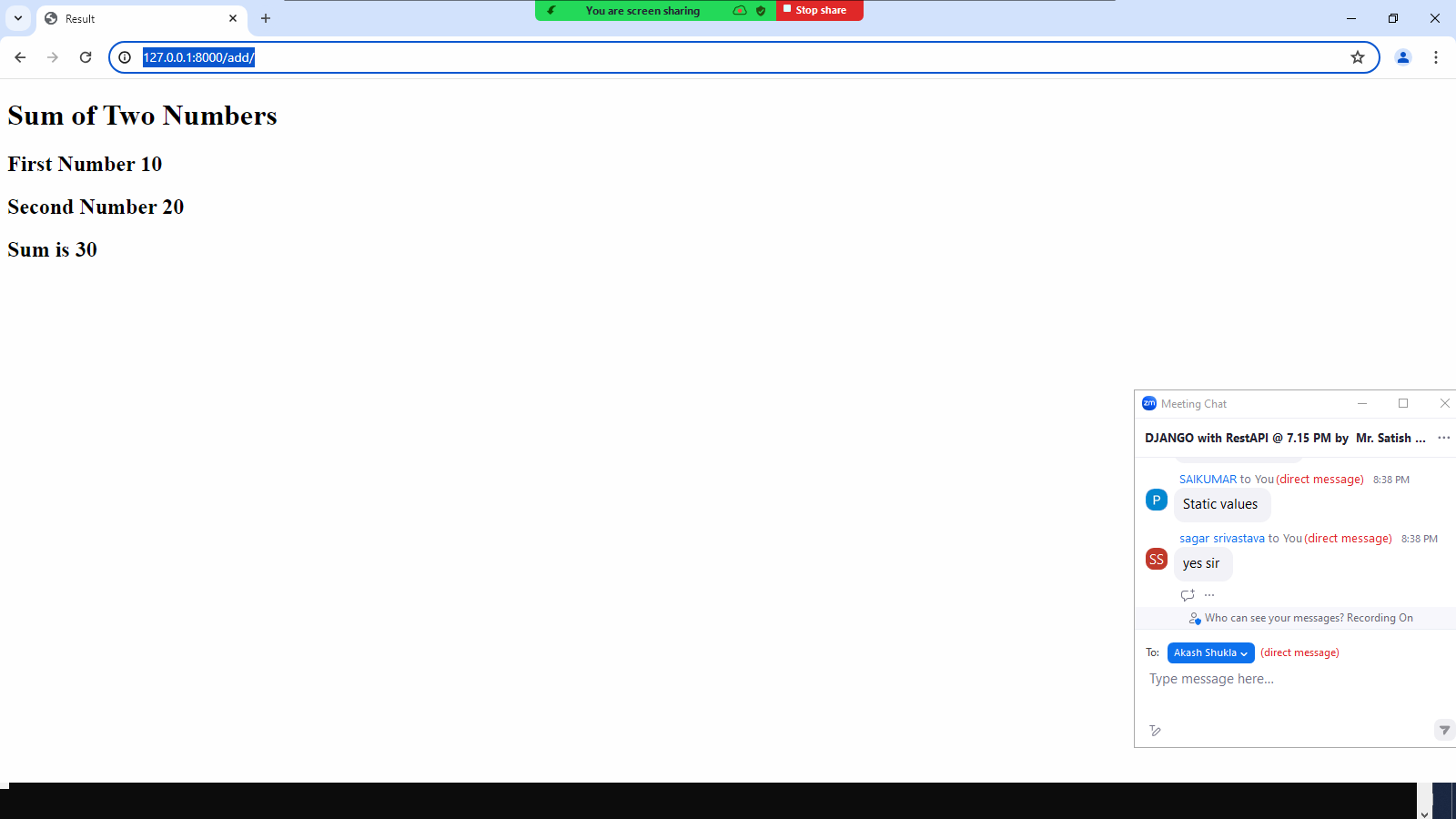
**Do following changes in settings.py**

TEMPLATE\_DIR=BASE\_DIR/'templates'

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'app1'  
]

TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [TEMPLATE\_DIR],

<http://127.0.0.1:8000/add/>



**add.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Result</title>  
 <style>  
 h1  
 {  
 color:red;  
 }  
 h2  
 {  
 color:blue;  
 }  
 body  
 {  
 background-color:cyan;  
 }  
  
 </style>  
</head>  
<body>  
 <h1> Sum of Two Numbers</h1>  
 <h2> First Number {{num1}}</h2>  
 <h2> Second Number {{num2}}</h2>  
 <h2> Sum is {{result}}</h2>  
</body>  
</html>

**Sending values to view from browser**

**Views.py**

from django.shortcuts import render  
  
*# Create your views here.*def add(request,num1,num2):  
 result=eval(num1)+eval(num2)  
 d={'num1':num1,'num2':num2,'result':result}  
 return render(request,'app1/result.html',context=d)

**urls.py**

from app1.views import add  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('add/<num1>/<num2>/',add)  
]