

FULL STACK DEVELOPMENT (21CS62)

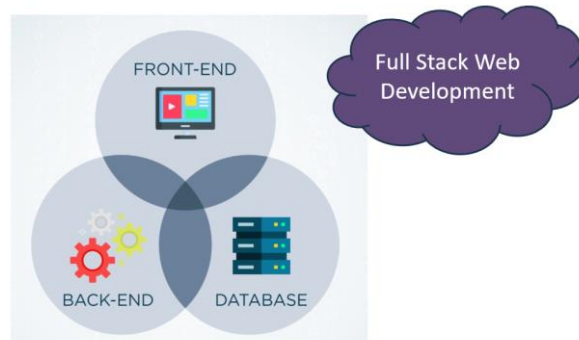
MODULE – 1

CONTENTS

1. Web framework
2. MVC Design Pattern
3. Django Evolution
4. Views
5. Mapping URL to Views
6. Working of Django URL Confs and Loose Coupling
7. Errors in Django
8. Wild Card patterns in URLs

Full Stack Development refers to the development of both frontend (client side) and backend (server side) portions of web application.

Full Stack Development with Django



The Key advantages of Full Stack Development are:

- Saves Time and Money
- Rounded Solution
- Great Exposure
- Complete Ownership
- Greater opportunity with more learning

Front End:

The front end is like the face of a website or web application that users interact with directly. It includes everything users see and interact with in their web browsers, such as buttons, forms, text, images, and animations. Front end development involves using languages like HTML, CSS, and JavaScript to create these user-facing elements and make them look good and work smoothly.

Example: When you visit a shopping website like Amazon, the front end is what you see on the homepage - the search bar, product listings, categories, and buttons to add items to your cart.

Back End:

The back end is like the behind-the-scenes of a website or web application. It consists of servers, databases, and applications that handle data processing, storage, and communication. Backend development involves using languages like Python, Ruby, PHP, or JavaScript (with Node.js) to create server-side applications and manage data stored in databases.

Example: When you log in to your email account, the back end is responsible for verifying your credentials, retrieving your emails from the database, and sending them to your browser to display.

Server:

A server is a computer or a software system that provides functionality or resources to other computers, known as clients, over a network. In web development, a server stores and manages data, processes requests from clients, and sends responses back. Servers can be physical machines or virtual machines hosted in data centers.

Web Server:

A web server is a specialized server software that handles HTTP requests and serves web pages and other web resources to clients over the internet. It processes requests from web browsers, retrieves the requested files from the server's file system, and sends them back to the client's browser for display.

Examples of web servers include Apache, Nginx, and Microsoft Internet Information Services (IIS).

Application Server:

An application server is a server software that hosts and manages applications and business logic on behalf of clients. It provides a runtime environment for running web applications and supports features like session management, security, and database connectivity. Application servers often work in conjunction with web servers to handle dynamic content generation and processing.

Example: When you use a social media platform like Facebook, the application server handles user authentication, stores and retrieves user data from the database, and generates the news feed based on your preferences.

1. WEB FRAMEWOK

Imagine you want to make a website where people can see the latest books. Back in the day, if you wanted to do this, you'd have to do everything yourself, kind of like baking a cake from scratch. You'd need to write code to connect to a database, get the latest books, and then show them on a web page. It might look something like this:

```
print "Content-Type: text/html"

print

print "<html><head><title>Books</title></head>"

print "<body>"

print "<h1>Books</h1>"

print "<ul>"

# code to connect to the database and get the latest books

print "</ul>"

print "</body></html>"
```

it's a python code generates a simple HTML page with a title ("Books") and potentially a list of books retrieved from a database.

Now, this might be okay for a simple page. But as your website gets bigger, things start getting messy:

What if you want to connect to the database from different pages? You need to write the database connection code again and again.

You want to bother with all the setup and cleanup stuff every time you write a new page? It's tedious and leads to mistakes.

Configuration management became complex in different environments.

Collaboration issues between developers and designers

And what if someone who's good at designing web pages but not so good at coding wants to change how the page looks? You'd want to keep the code for getting the books separate from the code for showing them, so they can change the look without messing up the functionality.

That's where a web framework like Django comes in. It's like a toolbox full of helpful stuff that you can use to build your website without starting from scratch every time. It handles all the boring, repetitive stuff like connecting to

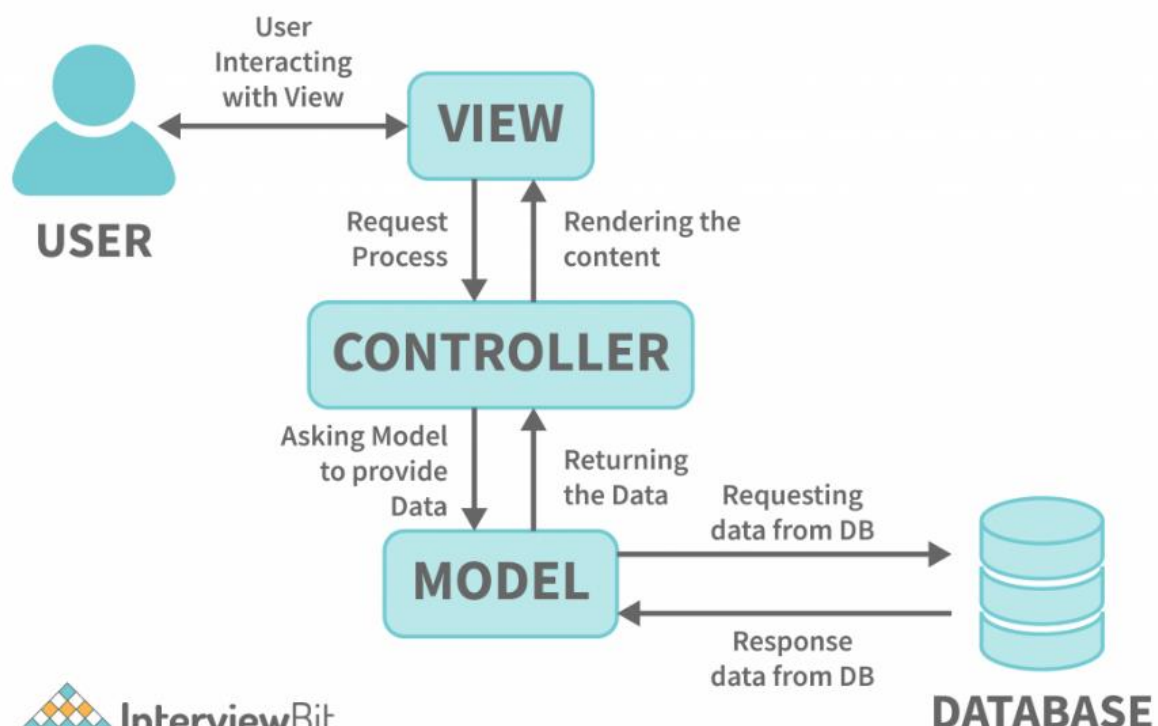
databases and managing web requests, so you can focus on writing the fun and interesting parts of your website. It makes your code cleaner, easier to maintain, and saves you a ton of time!

Django is a web framework that solves these problems, allowing developers to focus on writing clean, maintainable code without reinventing the wheel.

In summary, web frameworks like Django provide tools and structures to make web development easier by handling common tasks and promoting code organization and maintainability.

2. MVC Design Pattern

The MVC (Model-View-Controller) design pattern is a software architectural pattern commonly used in web development to separate an application into three interconnected components: the model, the view, and the controller.



Model:

The model represents the data and business logic of the application.

It manages the data, logic, and rules of the application's domain.

For example, in a web application, the model might represent database tables and manage interactions with the database.

View:

The view represents the presentation layer of the application.

It is responsible for displaying the data to the user in a particular format, such as HTML, XML, or JSON.

Views are often templates that contain the structure and layout of the user interface.

They receive data from the controller and render it to the user.

Controller:

The controller acts as an intermediary between the model and the view.

It receives user input (e.g., HTTP requests) from the view, processes it, and interacts with the model to retrieve or update data.

It decides which model methods to invoke and which view to render in response to a user action.

Controllers handle the flow of control and orchestrate the application's behavior.

How MVC Works:

A user interacts with the view (e.g., by clicking a button or submitting a form).

The view sends the user's input to the controller.

The controller processes the input, interacts with the model to retrieve or update data, and determines the appropriate response.

Based on the controller's decision, the model may be updated.

The controller selects the appropriate view and passes data to it for rendering.

The view receives the data from the controller and renders it to the user.

This separation of concerns makes the application easier to understand, maintain, and scale, as changes to one component typically do not require changes to the others.

3. Django Evolution

Django, a web development framework, was created to solve problems faced by developers building web applications. The typical process for developers used to be:

- Build a web app from scratch.
- Realize similarities between different apps and refactor code to reuse it.
- Repeat the process multiple times until realizing they've essentially created a framework.

This is exactly how Django came to be!

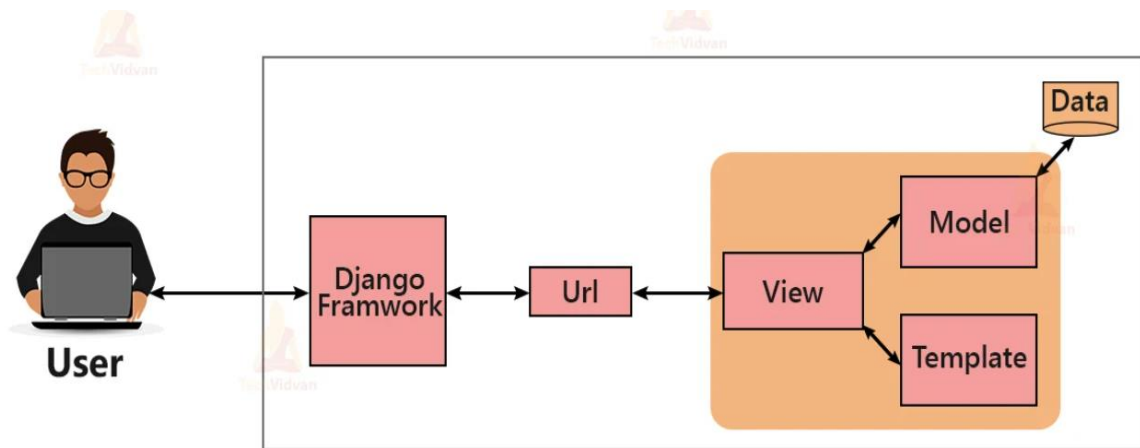
Django originated from real-world applications developed by a team at Lawrence Journal-World newspaper in Kansas. Adrian Holovaty and Simon Willison started using Python for web development in 2003. They needed to build and maintain several news sites quickly, often with tight deadlines. To cope with this pressure, they created Django as a time-saving web development framework.

In 2005, after successfully using Django to power their sites, they released it as open-source software. They named it Django after the jazz guitarist Django Reinhardt.

Django's origins in the news industry influenced its focus. It's particularly well-suited for content-driven sites like news portals. However, it's versatile and can be used for various types of dynamic websites.

The fact that Django was born from real-world projects shapes its community and development. Developers are motivated to continually improve Django based on their own experiences and challenges. They aim to make it easier for developers to save time, build maintainable applications, and handle high traffic. In simple terms, Django was created to solve real-world problems in web development and continues to evolve to meet the needs of developers.

How Django Works with MVT Achitecture



Model:

What it does:

Represents the data structure of the application.

Defines the database schema and manages interactions with the database.

View:

What it does:

Handles the business logic and request-response cycle of the application.

Retrieves data from the model, processes it, and passes it to the template for rendering.

Template:

What it does:

Defines the presentation layer of the application.

Contains HTML markup with placeholders for dynamic content.

Renders data received from the view to generate the final output sent to the client's browser.

Workflow of Django with MVT

In Django, when a user sends a request to the server, Django processes the request and generates a response. Here's a simple overview of how Django works from user request to response:

User Sends Request:

The user interacts with the application by sending a request to the server.

This request can be for accessing a webpage, submitting a form, or any other action supported by the application.

URL Routing:

Django's URL dispatcher receives the incoming request and matches it to a corresponding view function based on the URL pattern defined in the URL configuration.

View Function Execution:

Once a matching URL pattern is found, Django calls the associated view function.

The view function contains the business logic for processing the request.

It may interact with the database (through models) to retrieve or update data, process user input, or perform other tasks required to fulfill the request.

Template Rendering:

After processing the request, the view function typically renders a template.

Templates are HTML files with embedded template tags and filters that allow dynamic content rendering.

The view passes data to the template to populate placeholders, generating the final HTML response.

Response Sent to User:

Once the template is rendered, Django generates an HTTP response containing the HTML content.

This response is sent back to the user's browser, where it is displayed as a webpage.

User Receives Response:

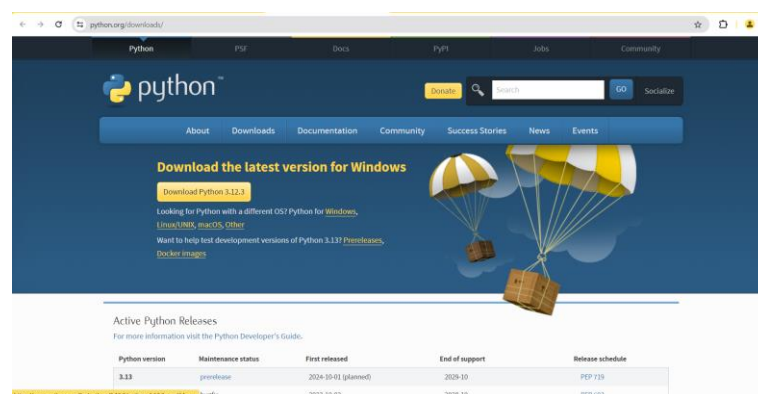
The user's browser receives the HTTP response and renders the webpage based on the provided HTML content.

The user can then interact with the webpage, submitting forms, clicking links, or performing other actions, which initiates a new cycle of requests and responses.

In summary, Django handles user requests by routing them to the appropriate view function, which processes the request, renders a template to generate HTML content, and sends the response back to the user's browser for display. This process allows Django to dynamically generate webpages and provide interactive web applications.

Installation, Django Setup and File structure

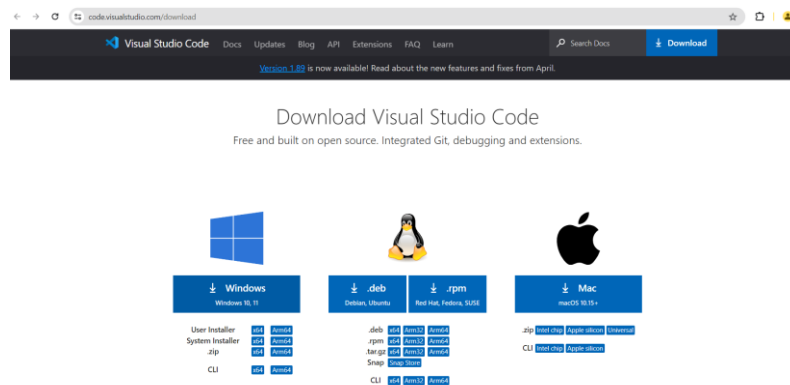
STEP 1: Install python type “python download” click the first link and download the python for windows



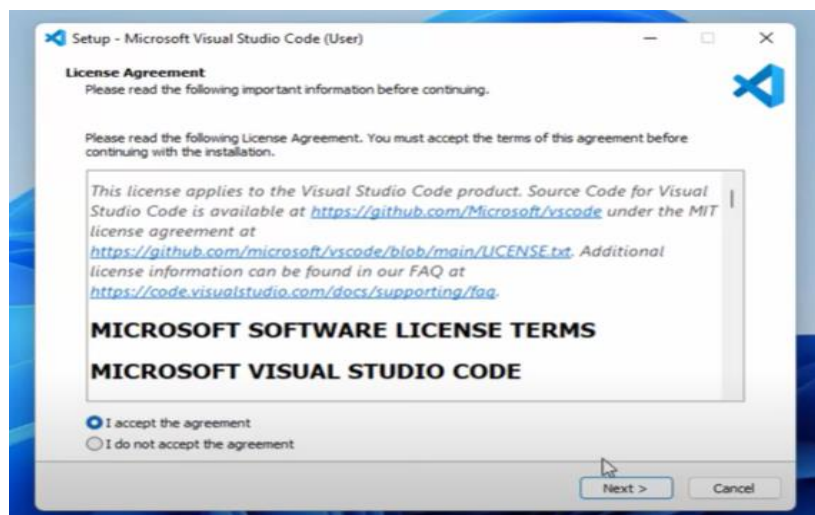
- After download check the below checkboxes then click on install now

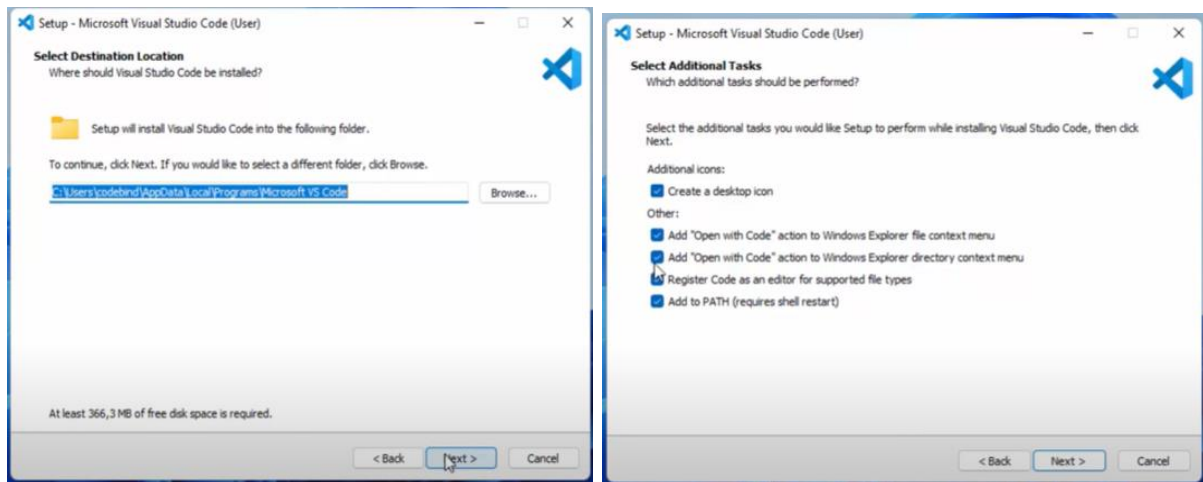


STEP 2: Install vscode by giving “visual studio code download” in chrome click the first link then download for windows



- Click the radio button I accept the agreement

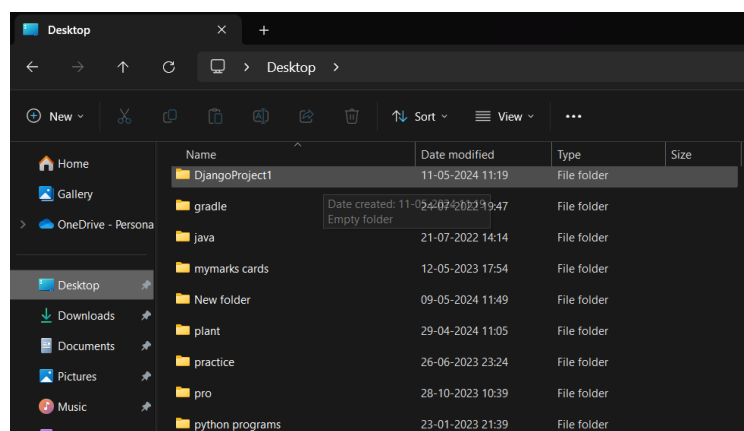




- Check all the check boxes and click next

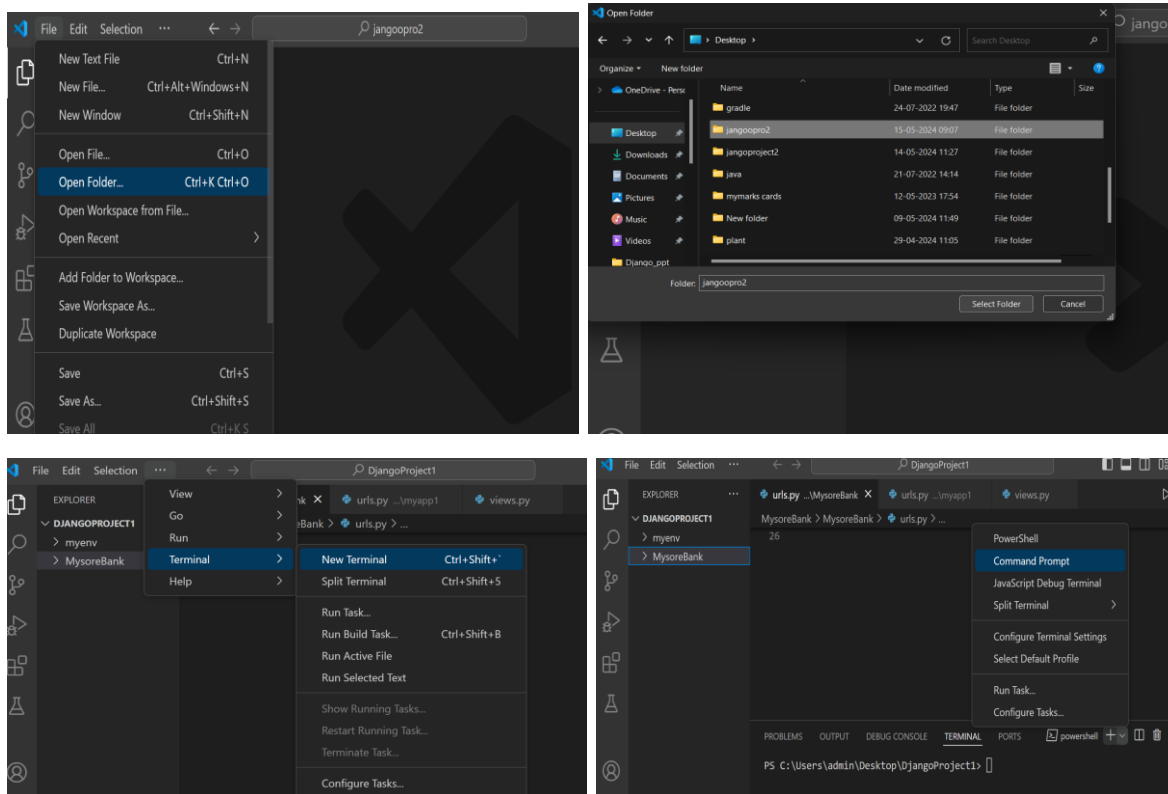
STEP 3: Create folder for Django Projects

DjangoProject1 is my folder



STEP 4: Open Vscode and click on open folder and select the Djangoproject folder which you have created

- Open new terminal
- Click on dropdown beside powershell and select “command prompt”



- Change path to desktop and folder which you have created

```

Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>cd desktop

C:\Users\admin\Desktop>cd DjangoProject1

C:\Users\admin\Desktop\DjangoProject1>

```

STEP 5: Now path is changed to your Projectfolder, next you have to created the Virtual Environment for the Django Projects

- Creating a virtual environment for a Django project keeps dependencies separate, making it easier to manage and ensuring that the project works consistently across different

computers. It also enhances security by preventing conflicts with other installed software.

- Command to create Virtual environment is “**python -m venv myenv**”
- **myenv** is the virtual environment name.

```
Command Prompt
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>cd desktop

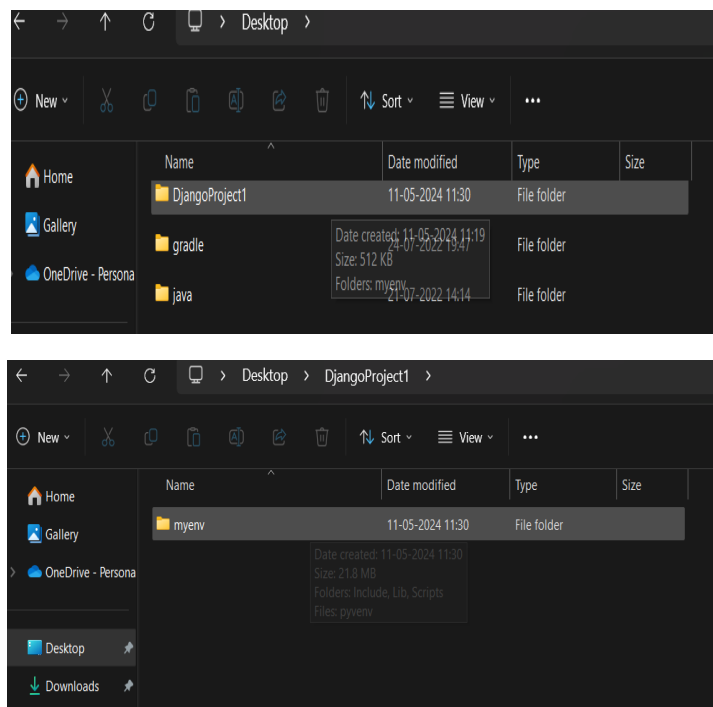
C:\Users\admin\Desktop>cd DjangoProject1

C:\Users\admin\Desktop\DjangoProject1>python -m venv myenv

C:\Users\admin\Desktop\DjangoProject1>|
```

STEP 6: Check whether myenv is created or not

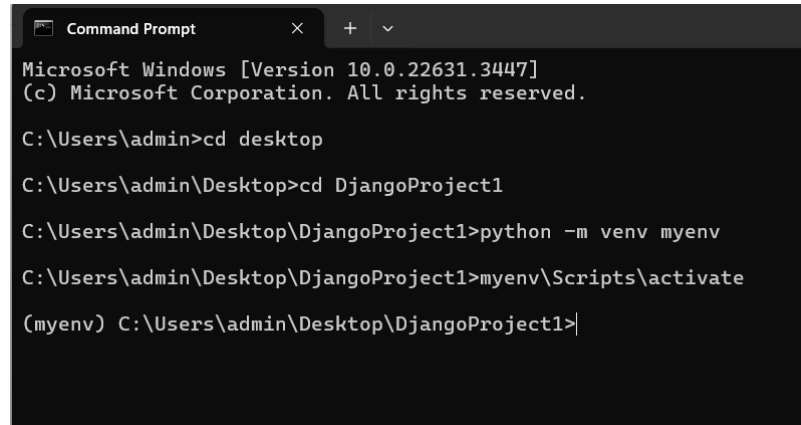
This is my project folder



myenv is created successfully created inside the DjangoProject1 folder

- Now we need to activate the myenv by giving the below command

myenv\Scripts\activate

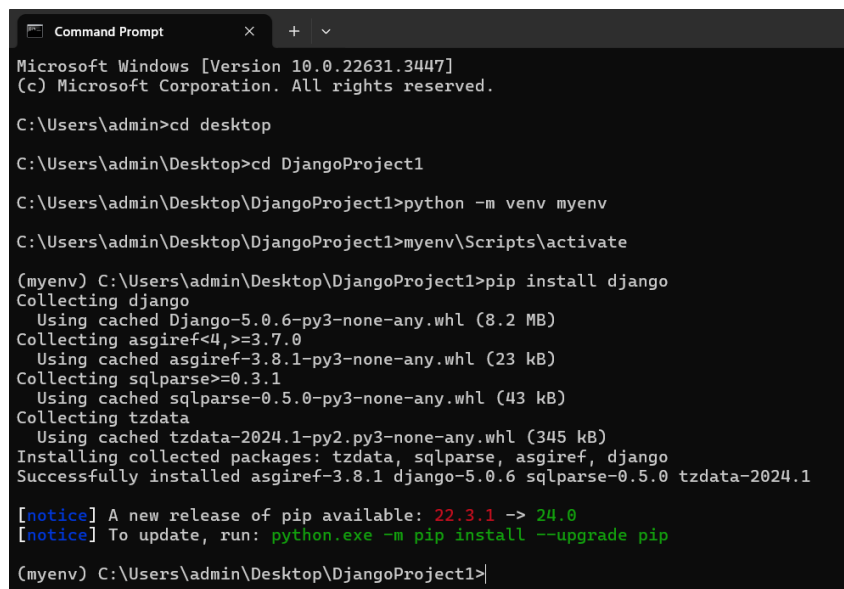


```
Command Prompt
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>cd desktop
C:\Users\admin\Desktop>cd DjangoProject1
C:\Users\admin\Desktop\DjangoProject1>python -m venv myenv
C:\Users\admin\Desktop\DjangoProject1>myenv\Scripts\activate
(myenv) C:\Users\admin\Desktop\DjangoProject1>
```

STEP 6: Install Django by giving below command

pip install django



```
Command Prompt
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

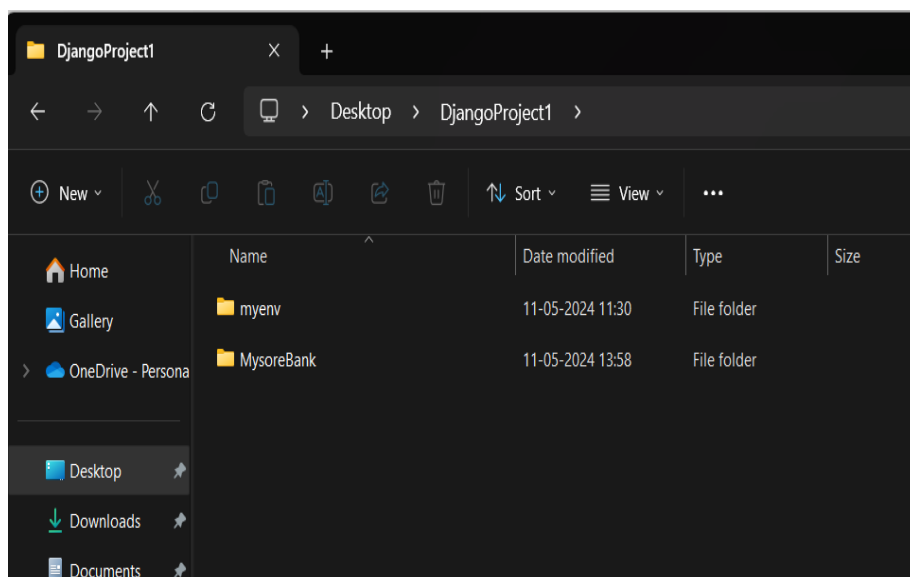
C:\Users\admin>cd desktop
C:\Users\admin\Desktop>cd DjangoProject1
C:\Users\admin\Desktop\DjangoProject1>python -m venv myenv
C:\Users\admin\Desktop\DjangoProject1>myenv\Scripts\activate
(myenv) C:\Users\admin\Desktop\DjangoProject1>pip install django
Collecting django
  Using cached Django-5.0.6-py3-none-any.whl (8.2 MB)
Collecting asgiref<4,>=3.7.0
  Using cached asgiref-3.8.1-py3-none-any.whl (23 kB)
Collecting sqlparse>=0.3.1
  Using cached sqlparse-0.5.0-py3-none-any.whl (43 kB)
Collecting tzdata
  Using cached tzdata-2024.1-py2.py3-none-any.whl (345 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.0.6 sqlparse-0.5.0 tzdata-2024.1

[notice] A new release of pip available: 22.3.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(myenv) C:\Users\admin\Desktop\DjangoProject1>
```

STEP 7: Create a project inside the folder by giving the command **“django-admin startproject MysoreBank”**

```
C:\Users\admin\Desktop>cd DjangoProject1
C:\Users\admin\Desktop\DjangoProject1>python -m venv myenv
C:\Users\admin\Desktop\DjangoProject1>myenv\Scripts\activate
(myenv) C:\Users\admin\Desktop\DjangoProject1>pip install django
Collecting django
  Using cached Django-5.0.6-py3-none-any.whl (8.2 MB)
Collecting asgiref<4,>=3.7.0
  Using cached asgiref-3.8.1-py3-none-any.whl (23 kB)
Collecting sqlparse>=0.3.1
  Using cached sqlparse-0.5.0-py3-none-any.whl (43 kB)
Collecting tzdata
  Using cached tzdata-2024.1-py2.py3-none-any.whl (345 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.0.6 sqlparse-0.5.0 tzdata-2024.1
[notice] A new release of pip available: 22.3.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(myenv) C:\Users\admin\Desktop\DjangoProject1>django-admin startproject MysoreBank
(myenv) C:\Users\admin\Desktop\DjangoProject1>
```

- Check whether project created in folder or not

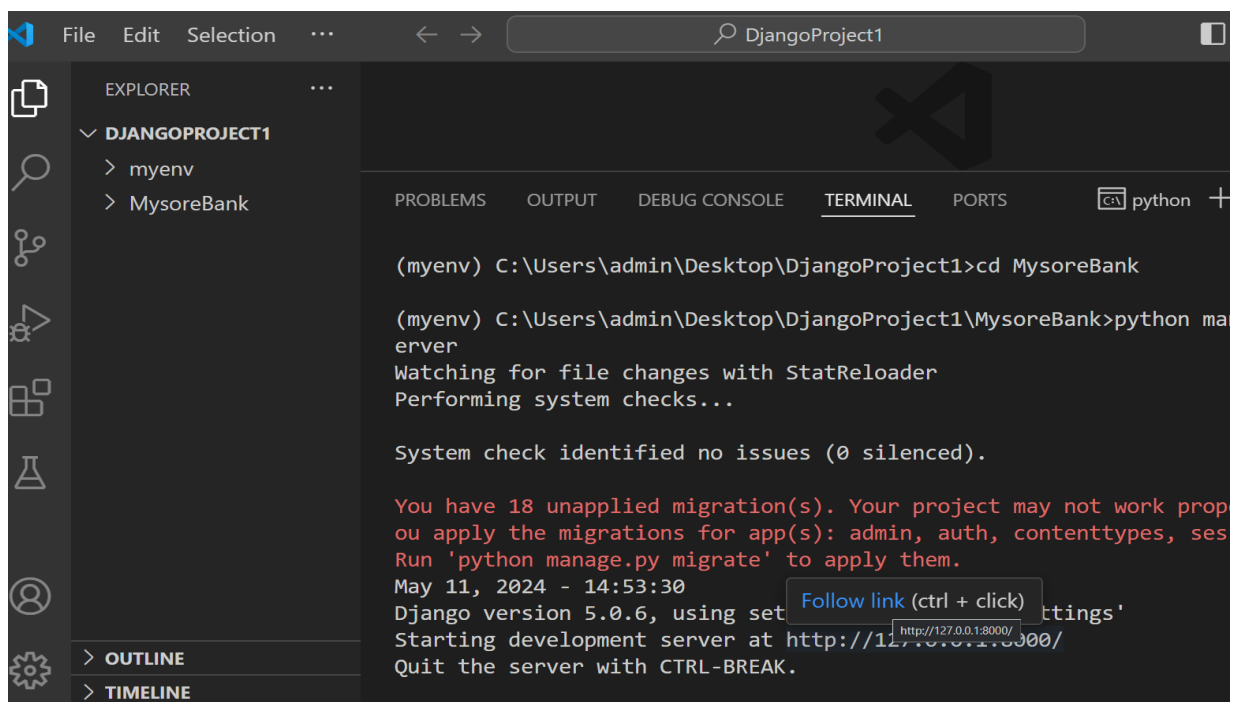


Project “MysoreBank” is created inside the DjangoProject1 folder

STEP 9: Change the directory to your current project by giving “cd MysoreBank” Django provides lightweight server to run your project if you want to run your developed project you need to give the below command

python manage.py runserver

In the MysoreBank project, you can run this command without having developed anything, and the default page will be displayed. From there, you can proceed to develop your project according to your specific requirements.

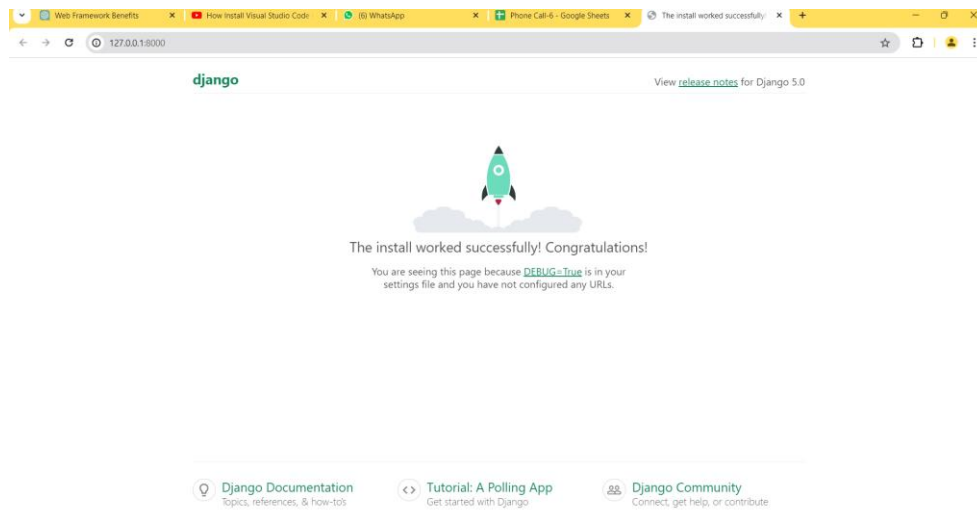


```
File Edit Selection ... < > DjangoProject1
EXPLORER
  DJANGOPROJECT1
    > myenv
    > MysoreBank
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS python +
(myenv) C:\Users\admin\Desktop\DjangoProject1>cd MysoreBank
(myenv) C:\Users\admin\Desktop\DjangoProject1\MysoreBank>python ma
server
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

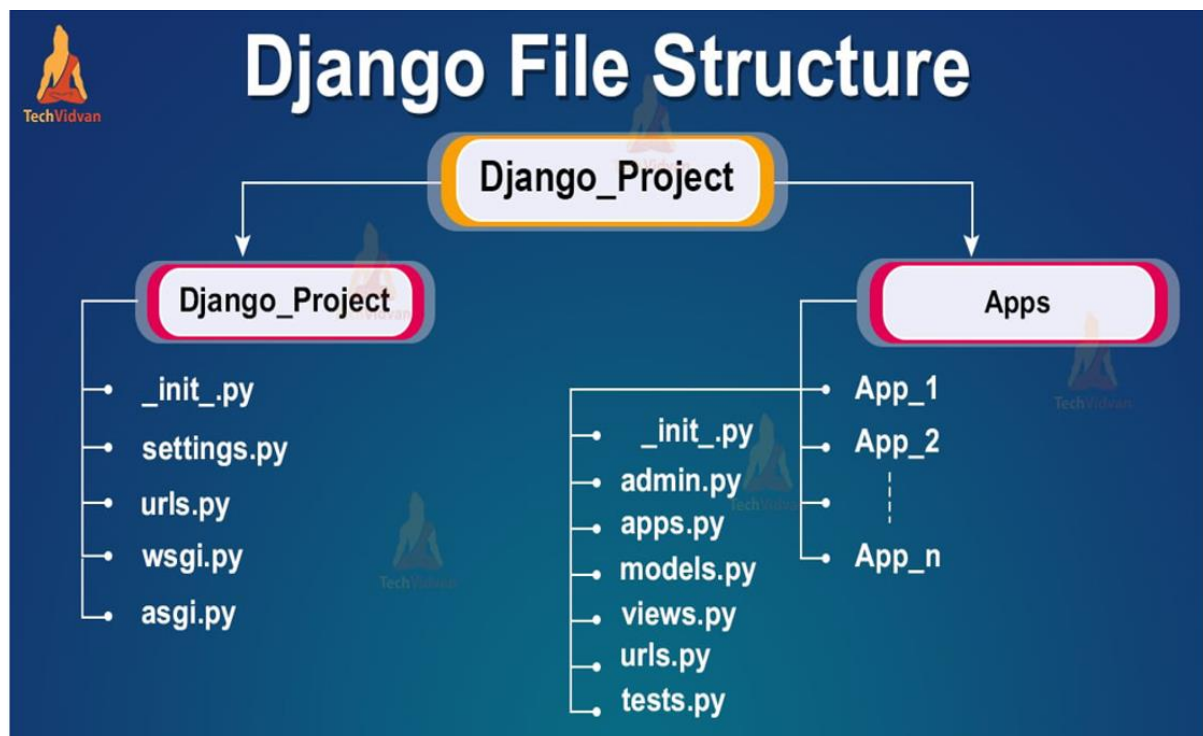
You have 18 unapplied migration(s). Your project may not work prop
ou apply the migrations for app(s): admin, auth, contenttypes, ses
Run 'python manage.py migrate' to apply them.
May 11, 2024 - 14:53:30
Django version 5.0.6, using set Follow link (ctrl + click) ttings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Click on the ip address and you will get this output in browser



Now you have successfully installed, made a setup and run the project now you can customize your project according to your requirement.

File Structure:



Project Folder (MysoreBank)

- Contains the main project settings and configurations.
- Consists of the project-wide URLs and the WSGI/ASGI configuration files for deployment.

Project Files:

- **__init__.py**: Empty file that signifies the folder as a Python package.
- **settings.py**: Configuration file for Django project settings (e.g., database settings, installed apps).
- **urls.py**: Main URL configuration for the project, mapping URLs to views.
- **wsgi.py**: WSGI application entry point for deployment using WSGI-compatible servers.
- **asgi.py**: ASGI application entry point for deployment using ASGI-compatible servers.

Apps (e.g., deposit, withdraw, check_balance, etc.)

- Represents a modular component of the project, containing specific functionality.
- Each app can have its own models, views, templates, and URLs.

App Files:

- **__init__.py**: Empty file signifying the folder as a Python package.
- **admin.py**: Configuration of models for Django Admin interface.
- **apps.py**: Configuration of the app (e.g., display name).
- **models.py**: Definition of database models (entities) for the app.
- **views.py**: Contains view functions or classes that handle requests and return responses.
- **urls.py**: URL configuration specific to the app, mapping URLs to views.
- **tests.py**: Unit tests for testing app functionality.

4. Views

Views are Python functions that receive web requests and return web responses.

They contain the logic to process user requests and generate appropriate responses.

Purpose:

Views handle the business logic of a web application, such as fetching data from a database, processing form submissions, or rendering templates.

Request Handling:

Views take a request object as input, which contains information about the user's request (e.g., URL, HTTP method, parameters).

The view processes the request and generates a response object to send back to the user.

Response Generation:

Views generate responses using response objects, typically instances of `HttpResponse` or its subclasses.

Responses can be HTML content, JSON data, redirect instructions, or error messages, depending on the application's requirements.

Routing:

URLs are mapped to views using URL patterns defined in the project's `urls.py` module.

When a URL matches a defined pattern, Django calls the corresponding view function to handle the request.

Templates and Context:

Views often render HTML templates to generate dynamic content for web pages.

They pass data to templates using a context dictionary, allowing templates to access and display dynamic content.

Testing:

Views can be unit tested to ensure they handle different types of requests correctly and produce the expected responses.

Django provides testing utilities to simulate HTTP requests and verify view behavior.

views in Django are essential components that process user requests, execute application logic, and generate appropriate responses, ultimately driving the behavior and functionality of web applications.

5. Mapping URL to Views and creating first App in Django

After successful installation and setup next thing is to create first app in Django which display “hello world” in web browser

STEP 1: Open Vscode select your project folder open terminal and select the command prompt

First change your directory to the project created in this case MysoreBank is the project name

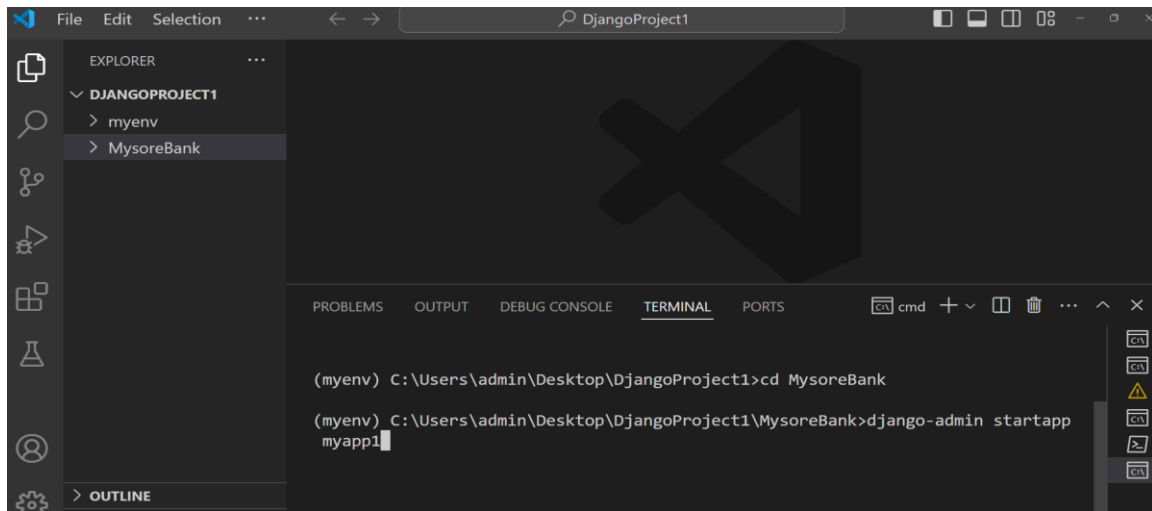
cd MysoreBank

Then create the app by giving below command

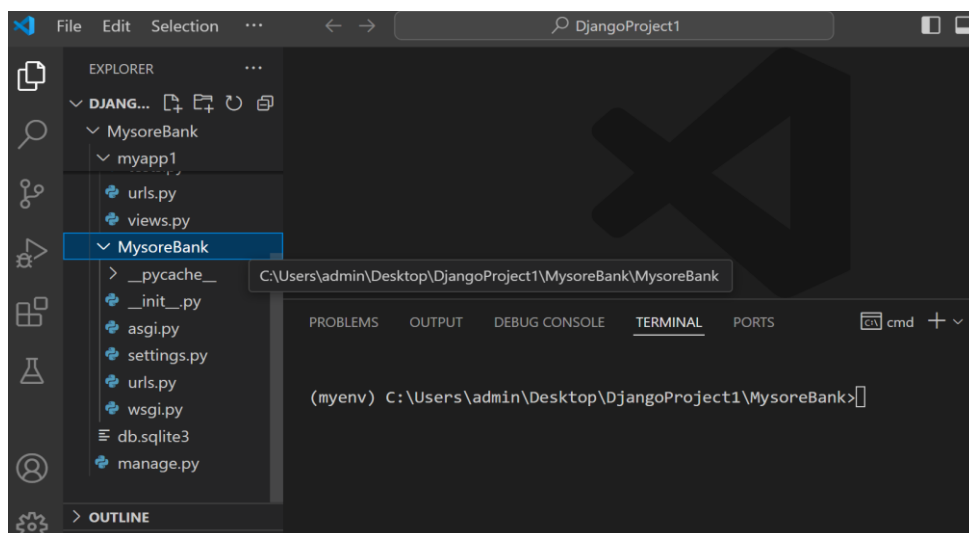
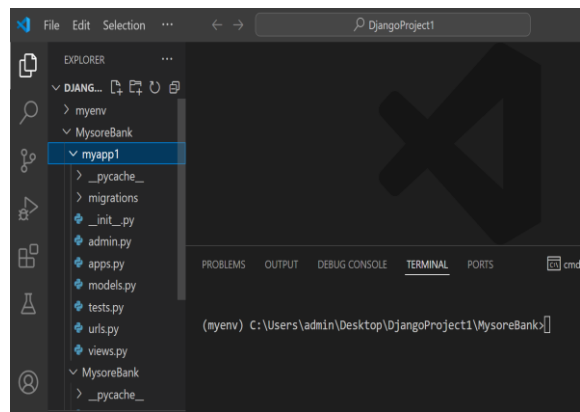
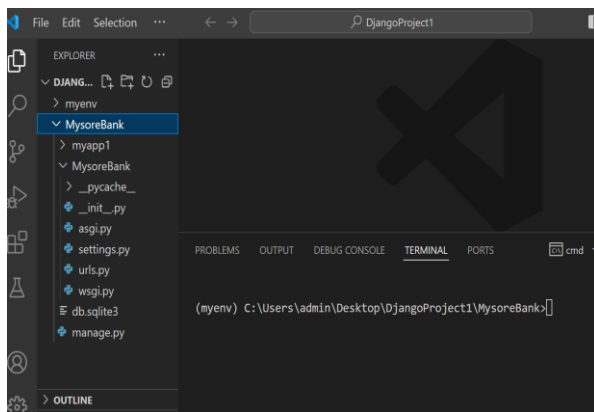
django-admin startapp myapp1

myapp1 represents the app name

Then check whether app is present inside the MysoreBank folder

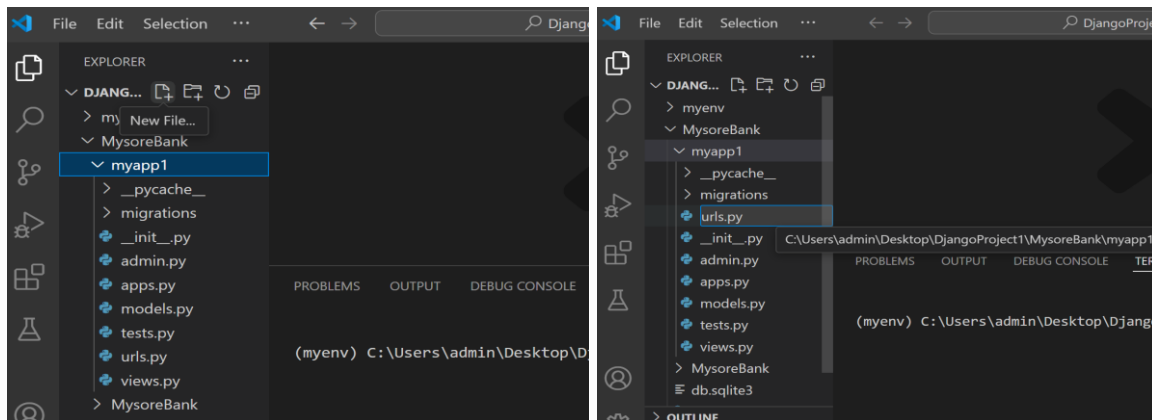


STEP 2: click on MysoreBank → myapp1 → MysoreBank



STEP 4: click on myapp1 → New File Icon

You will find one Textbox in myapp1 give name as “urls.py”



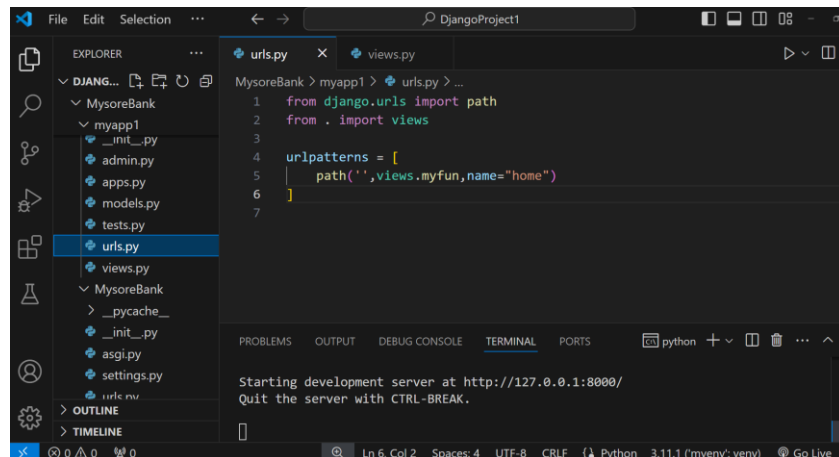
STEP 5: urls.py file is created in myapp1 App now type the below code inside the urls.py file in myapp1

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

```
# Create your views here.
def myfun(request):
    return HttpResponse("hello world")
```

urls.py (myapp1):

- This file defines the URL patterns for the myapp1 App.
- **from django.urls import path**: Importing the path function from django.urls.
- **from . import views**: Importing the views module from the current directory.
- **urlpatterns**: A list of URL patterns.
- **path("", views.myfun, name="home")**: Maps an empty URL to the myfun view function from the views.py file within the myapp app, and assigns the name "home" to this URL pattern.



STEP 6: Now open views.py file in myapp1 and type the below code

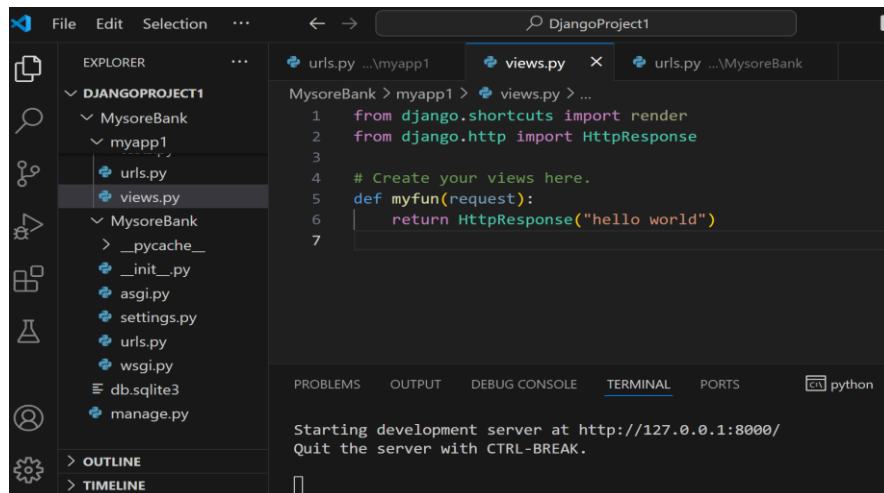
```
from django.shortcuts import render
from django.http import HttpResponse
```

Create your views here.

```
def myfun(request):
    return HttpResponse("hello world")
```

views.py (myapp1):

- This file contains the view function myfun.
- **from django.http import HttpResponse:** Importing HttpResponse from django.http.
- **def myfun(request):** Defines the view function myfun, which takes an HttpRequest object as a parameter.
- **return HttpResponse("hello world"):** Returns an HTTP response with the content "hello world".



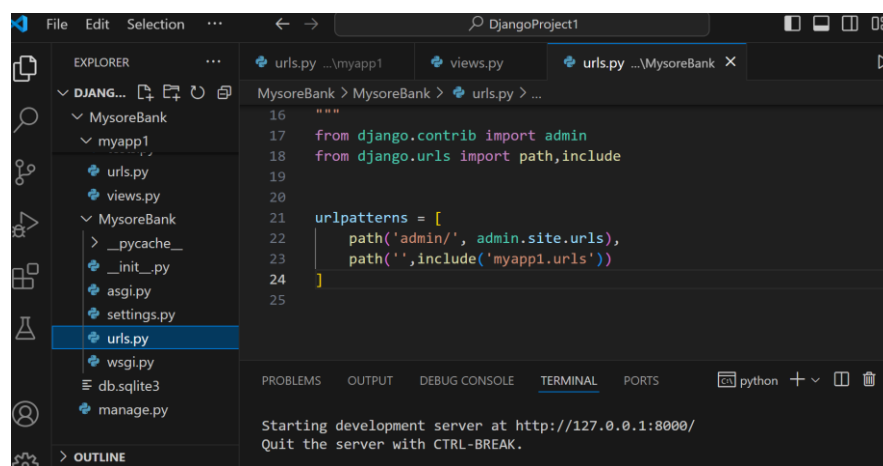
STEP 7: Now open urls.py file in MysoreBank project and type the below code

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp1.urls'))
]
```

urls.py (Mysorebank):

- This file defines the URL patterns for the entire project, including the myapp1 app.
- **from django.urls import path, include:** Importing the path and include functions from django.urls.
- **from django.contrib import admin:** Importing the admin module.

- **urlpatterns:** A list of URL patterns.
- **path('', include('myapp1.urls')):** Includes the URL patterns defined in the urls.py file of the myapp app. This means that URLs matching those patterns will be handled by the views defined in myapp.
- **path('admin/', admin.site.urls):** Maps the URL /admin/ to the Django admin interface.

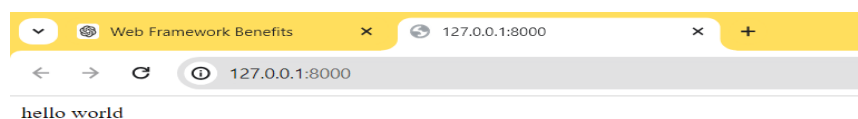


```
16 """
17 from django.contrib import admin
18 from django.urls import path, include
19
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('', include('myapp1.urls'))
24 ]
25
```

Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

Now run below command in terminal to execute the project

Python manage.py runserver



In Mysorebank/urls.py, the empty path(' ') is linked to the URL patterns defined in myapp1.urls, meaning that when a user visits the root URL of the website, Django will look for further URL patterns in myapp1.urls.

In myapp1/urls.py, the empty path(' ') is mapped to the myfun view function. Therefore, when a user visits the root URL, they will see the "hello world" response generated by the myfun view function in views.py.

This is how the mapping of URLs to view functions is done in a Django application.

6. Working of Django URL Confs and Loose Coupling

Loose Coupling Principle:

Loose coupling is a software design approach that keeps components of a system independent from each other, Changes made to one component have minimal or no impact on other components.

In Django, URLconfs (URL configurations) and view functions are loosely coupled, URL definitions are kept separate from the view functions they call.

In some other frameworks, the URL is tightly coupled with the file location or method name (e.g., early CherryPy or typical PHP applications).

This tight coupling can become unmanageable as the project grows.

Advantages of Loose Coupling in Django:

Interchangeability: You can change the URL for a view function without modifying the function itself.

Flexibility: You can change the logic inside a view function without affecting the URLs mapped to it.

Maintainability: You can expose the same functionality at multiple URLs by modifying the URLconf, without touching the view code.

Ease of Updates: Modify URLs or view functions independently.

Code Reusability: Use the same view function for different URLs.

Simplified Refactoring: Update or refactor view logic without breaking the URL structure.

Example in Django:

Suppose you have a view function that displays the “hello world”.

If you want to change the URL from ‘ ‘ to /root/ you only need to update the URLconf.

If you want to change the logic of the view function, you can do so without altering the URLconf.

Urls.py(myapp)

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

```
urlpatterns = [
    path('', views.myfun, name="home") → path('home/', views.myfun, name="home")
]
```

Views.py(myapp)

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

```
def myfun(request):
    return HttpResponse("hello")
```

urls.py(mysite) → mainproject urls.py file

```
from django.contrib import admin
from django.urls import path, include
```

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

This is the code that displays 'hello' on the webpage. Here, I gave the URL path as empty. What if I change it to /root/ in the main project's urls.py and to /home/ in the urls.py file of myapp? Will it affect the views? The answer is no, it will not affect the views because urls.py and views.py are loosely coupled. We can change the view function response from 'hello' to 'hi' without affecting the URLs."

In summary, loose coupling in Django, especially with URLconfs and view functions, allows for greater flexibility, easier maintenance, and better scalability by keeping the URL mappings and the logic of the views separate and independent.

7. Errors in Django

404 Error in Django:

A 404 error occurs when a requested URL is not defined in the URLconf.

Run the Django development server and try to access a URL that is not defined, such as <http://127.0.0.1:8000/hello/> or <http://127.0.0.1:8000/does-not-exist/>.

When a non-existent URL is accessed, Django displays a "Page not found" message (404 error page).

Information Provided:

The 404 error page shows which URLconf Django used.

It lists every pattern in that URLconf.

This helps developers understand why the URL is not found.

Development Mode:

This detailed 404 error page is displayed only when the Django project is in debug mode “Debug = True”

Debug mode is active by default when you first create a Django project.

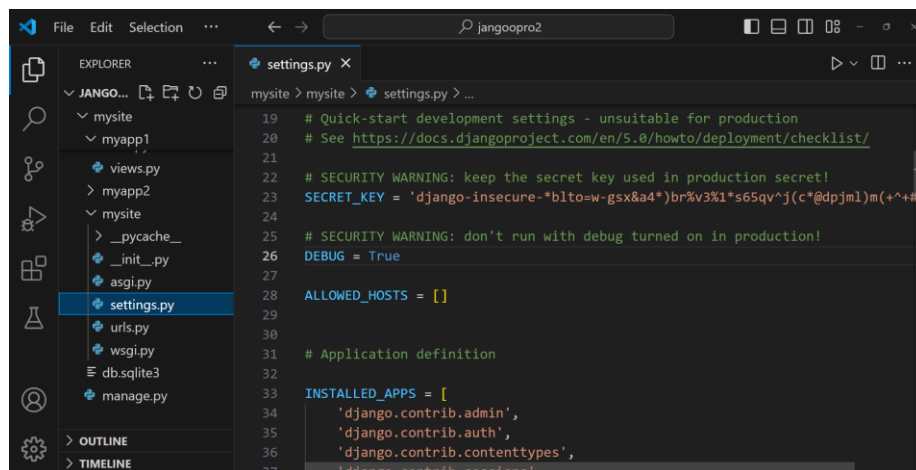
In a live production site, you wouldn't want to expose detailed information about your URLconf to users.

When debug mode is off, a different, less informative 404 error page is displayed to protect sensitive information.

Debug Mode Setting:

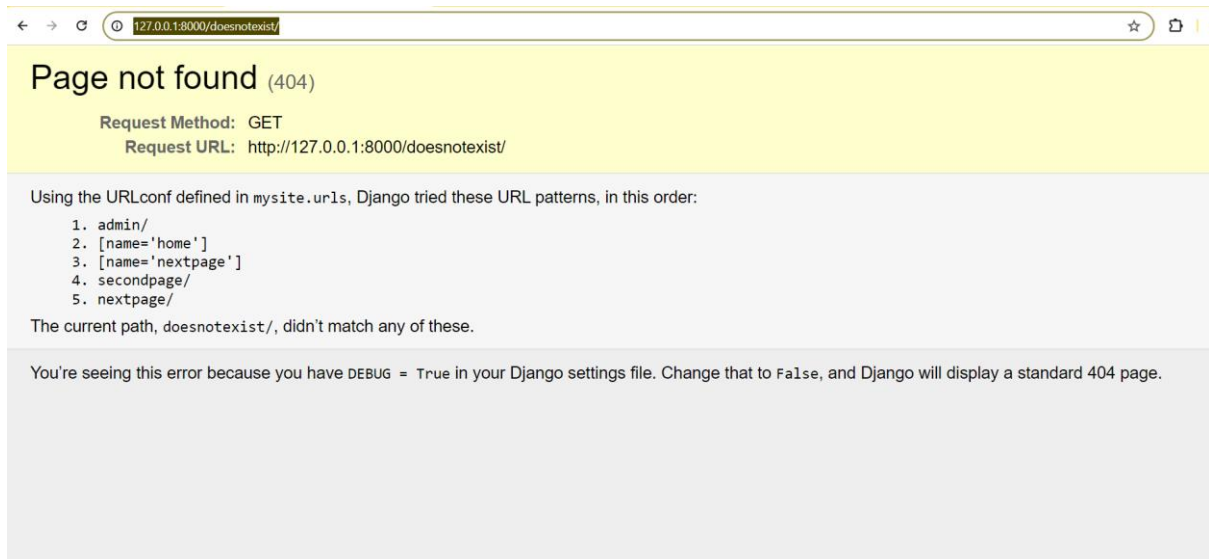
Debug mode can be turned off by setting `DEBUG = False` in the Django settings file.

Turning off debug mode is recommended for production environments to enhance security.



In summary, Django's 404 error page in debug mode provides helpful information for developers by showing which URLconf and patterns were used, but this detailed information is hidden in production to protect sensitive data.

- Django Error page with “DEBUG = **True**”



- Django Error with “DEBUG = **False**”

In terminal you will get the error message like this

```
[17/May/2024 13:24:00] "GET / HTTP/1.1" 200 17
Not Found: /doesnotexist/
[17/May/2024 13:24:17] "GET /doesnotexist/ HTTP/1.1" 404 2660
```

Static vs Dynamic URLs

- **Static URL:** A fixed URL like /time/ that always shows the same page.
- **Dynamic URL:** A URL that changes based on parameters, influencing the page content.

Example Scenario:

You want to create a view that shows the current date and time but offset by a certain number of hours.

For example, /time/plus/1/ shows the time one hour into the future, /time/plus/2/ shows two hours into the future, and so on.

Incorrect Approach

A novice might think to write separate view functions for each offset and create a URLconf like this:

```
urlpatterns = [  
    path('time/', current_datetime),  
    path('time/plus/1/', one_hour_ahead),  
    path('time/plus/2/', two_hours_ahead),  
    path('time/plus/3/', three_hours_ahead),  
]
```

This approach is flawed because it leads to redundant view functions.

It's limited to predefined hour ranges.

Adding new ranges means adding more view functions and URL patterns.

Correct Approach: Abstraction with Dynamic Parameters

Instead, use a single view function with a dynamic parameter:

```
from django.urls import path  
from . import views  
  
urlpatterns = [  
    path('time/', views.current_datetime),  
    path('time/plus/<int:hours>/', views.hours_ahead),  
]
```

This way, you only need one view function to handle different offsets.

Pretty URLs

Pretty URLs: Clean, simple, and readable URLs like `/time/plus/3/` instead of `/time/plus?hours=3`.

Benefits:

- Easier to read and remember.
- Django encourages pretty URLs by making it easy to use them.

Implementing the Example

View Function:

```
from django.http import HttpResponse
from django.utils.timezone import now, timedelta

def current_datetime(request):
    now_time = now()
    return HttpResponse(f"Current date and time: {now_time}")

def hours_ahead(request, hours):
    offset_time = now() + timedelta(hours=hours)
    return HttpResponse(f"Date and time {hours} hours ahead: {offset_time}")
```

URL Configuration:

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

urlpatterns = [
    path('time/', views.current_datetime),
    path('time/plus/<int:hours>/', views.hours_ahead),
]
```

Use dynamic URLs with parameters to avoid redundancy, Pretty URLs are more user-friendly and are encouraged by Django, Abstraction helps in creating flexible and maintainable code.

8. Wild Card Patterns in URLs

Introduction

In this example, we aim to create a view in Django that displays the current date and time, offset by a specified number of hours, using dynamic URLs.

1. Regular Expressions in URL Patterns

Regular Expressions: We use regular expressions to match URL patterns dynamically.

Example: `\d+` matches one or more digits.

Explanation of Regular Expression Elements

- . Matches any single character except newline.
- ^ Matches the start of the string.
- \$ Matches the end of the string.
- * Matches zero or more repetitions of the preceding element.
- + Matches one or more repetitions of the preceding element.
- ? Matches zero or one repetition of the preceding element.
- [] Matches any one character within the brackets.
- \d Matches any digit (equivalent to [0-9]).
- \w Matches any word character (alphanumeric plus underscore).
- \s Matches any whitespace character (space, tab, newline).

Example URL Patterns

```
from django.conf.urls.defaults import *
from mysite.views import current_datetime, hours_ahead
```

```
urlpatterns = [
    path('time/', current_datetime),
    path('time/plus/\d+/', hours_ahead),
]
```

The above URL matches URLs like /time/plus/2/, /time/plus/25/, etc.

Limiting the Offset

Limit the offset to a maximum of 99 hours using `\d{1,2}`.

```
(r'^time/plus/\d{1,2}/$', hours_ahead),
```

Capture Data

Use parentheses in the regex to capture the dynamic part of the URL

```
(r'^time/plus/(\d{1,2})/$', hours_ahead),
```

Final url conf

```
from django.conf.urls.defaults import *
from mysite.views import current_datetime, hours_ahead
```

```
urlpatterns = [
    path(r'^time/$', current_datetime),
    path(r'^time/plus/(\d{1,2})/$', hours_ahead)
]
```

Complete views.py

```
from django.http import HttpResponse
from django.utils.timezone import now, timedelta

def current_datetime(request):
    now_time = now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)

def hours_ahead(request, hours):
    offset_time = now() + timedelta(hours=hours)
    html = "<html><body>In %s hour(s), it will be %s.</body></html>" % (offset,
dt)
    return HttpResponse(html)
```

Steps:

- **Convert Offset:** Convert offset to an integer.
- **Calculate Future Time:** Add the offset to the current time.
- **Generate HTML:** Create an HTML response displaying the future time.
- **Return Response:** Return the HTML response.

Testing the View

Start Server: Ensure the Django development server is running.

Test URLs:

`http://127.0.0.1:8000/time/plus/3/`

`http://127.0.0.1:8000/time/plus/5/`

`http://127.0.0.1:8000/time/plus/24/`

Invalid URL like `http://127.0.0.1:8000/time/plus/100/` should return a 404 error.

We've so far been producing views by hard-coding HTML into the Python code. Unfortunately, this is nearly always a bad idea. Luckily, Django ships with a simple yet powerful template engine that allows you to separate the design of the page from the underlying code. We'll dive into Django's template engine in the next chapter.

