Django

1. Django was developed by two people: **Adrian Holovaty** and **Simon Willison**.
2. It was created in **2003** and released as **open source in 2005**.
3. The current **standard versions** are 4.2 and 5.1, though earlier versions like 2.0 and 3.0 were standard at the time of their release.
4. The name **"Django"** was chosen as a tribute to **Django Reinhardt**, a famous Belgian-born Romani-French jazz guitarist. The developers were fans of jazz and admired Django Reinhardt's **skill, speed, and creativity**. They wanted the web framework to reflect those qualities.
5. Django is a **Python web framework** that makes it easier to create websites using Python.
6. We use Django primarily for building web applications, including managing the connection between different components.
7. Django acts as a **bridge between the frontend, backend, and databases**.
8. Other major Python web frameworks used for this purpose include **Flask** and **FastAPI**.
9. In **Flask**, almost every feature must be defined manually by the developer, whereas **Django provides many built-in functions and features**.
10. Flask is often preferred for **small or mini projects**, while Django is typically used for **large-scale projects**.
11. Although Django can be used for small projects, it is **less common** due to its numerous built-in features, which may result in **higher space and time complexity** for simpler applications.
12. **Real-time examples** of applications built with Django include **Instagram** and **Pinterest**.
13. Django follows the **Model-View-Template (MVT)** design pattern and adheres to the **DRY (Don't Repeat Yourself)** principle.
14. Django provides Architecture, API's and Admin Panel.

Model:

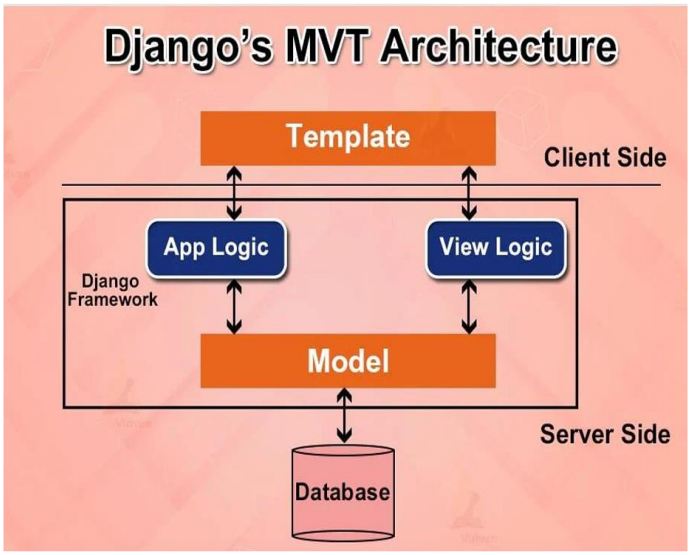
- Represents the data you want to display, typically sourced from a database.
- In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.
- The models are usually located in a file called models.py.

View:

- A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.
- The views are usually located in a file called views.py.

Template:

- A text file (usually HTML) that defines the structure of the web page and includes logic for displaying the data.

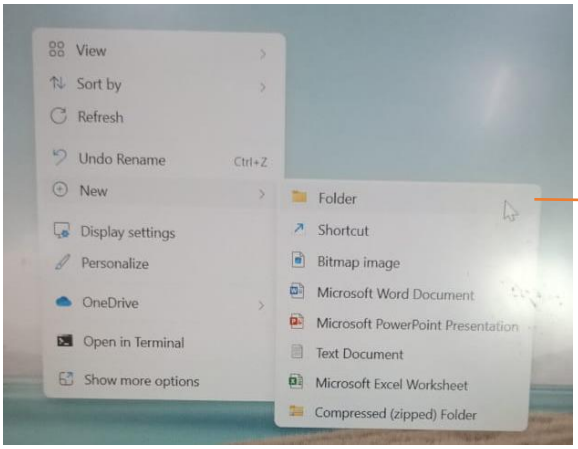


Django

- Introduction
- Getting Started
- Virtual Environment
- Install Django
- Create Project
- Create App
- Views
- Urls
- Templates
- Models
- Insert Data
- Update Data
- Delete Data
- Update Model

Process to create and installation of django:

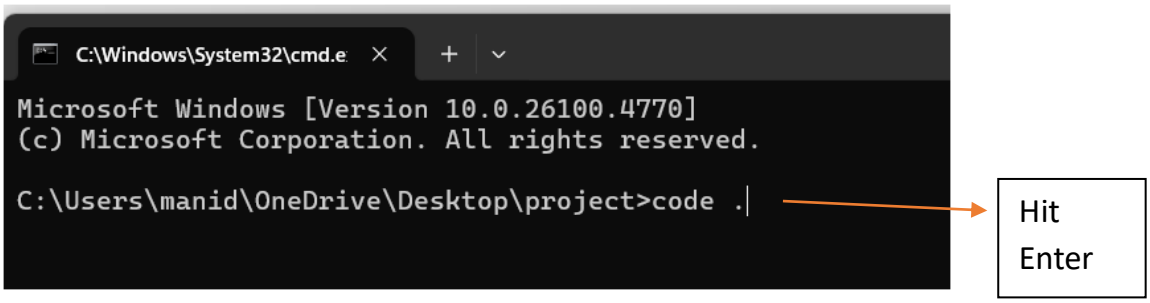
1) Create a new folder on desktop by clicking on new and name it



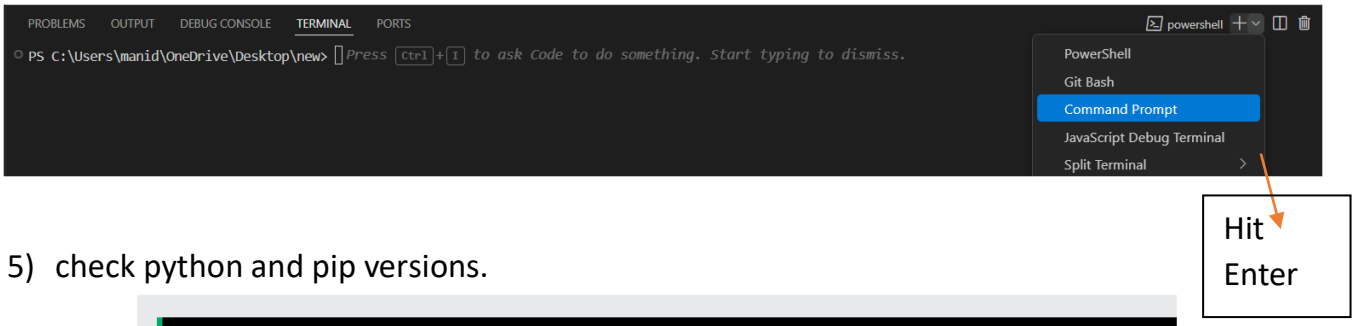
Create a new folder by clicking on folder and name it

2) Open the folder and in search bar you will see like this.

3) Then we will get this interface and then type “code .” and hit enter. This will redirect you to visual code studio.



4) In visual code studio initially the terminal is in power shell then switch it to command prompt.



5) check python and pip versions.

```
python --version
```

You will get the version of python

```
pip --version
```

You will get the version of pip

6) Then create an environment for our code.
7) Here comes one question **why we need environment for creating projects? We use environments mainly for security purposes, so that our code remains secure and others cannot access it without permission.**

```
>python -m venv environment
```

This will set up a virtual environment, and create a folder named "environment" with subfolders and files, like in fig-1 at initially the scripts folder will contain only two files after when it gets activated it contain activate, activate.bat files etc like in fig-2.

Terms in that are python means programming language, m means module upto this environment and in github also there is a letter m which stands for (master and main), v stands for virtual and env stands for environment, and environment is the name of the environment we are creating and it is user choice. There will be two stages in the environment

1) Activate
2) De-Activate

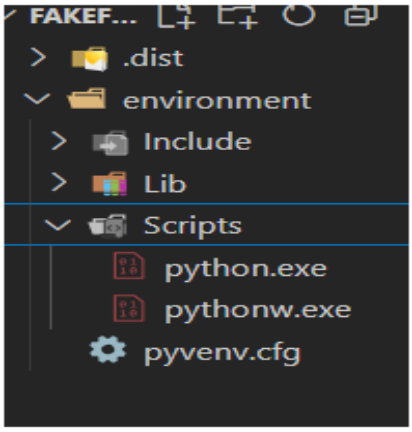


Fig-1

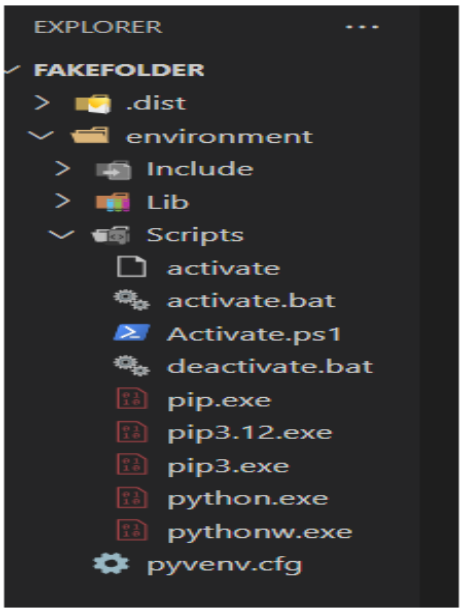


Fig-2

8) After creating an environment we have to activate it, with the following commands in windows:

```
C:\Users\manid\OneDrive\Desktop\fakefolder>environment\Scripts\activate.bat
```

Then for unix/MacOS:

Unix/MacOS:

```
source myworld/bin/activate
```

9) Once the environment is activated, you will see this result in the command prompt:

```
(environment) C:\Users\manid\OneDrive\Desktop\fakefolder>
```

10) After activation install Django. There are two ways in installing

1)

```
python -m pip install Django
```

Which will give a result that looks like this

```
Collecting django
  Using cached django-5.2.4-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref>=3.8.1 (from django)
  Using cached asgiref-3.9.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from django)
  Using cached sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Collecting tzdata (from django)
  Using cached tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Using cached django-5.2.4-py3-none-any.whl (8.3 MB)
Using cached asgiref-3.9.1-py3-none-any.whl (23 kB)
Using cached sqlparse-0.5.3-py3-none-any.whl (44 kB)
Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.9.1 django-5.2.4 sqlparse-0.5.3 tzdata-2025.2

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

2) Another way is through pip command

```
(environment) C:\Users\manid\OneDrive\Desktop\fakefolder>pip install django
Collecting django
  Using cached django-5.2.4-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref>=3.8.1 (from django)
  Using cached asgiref-3.9.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from django)
  Using cached sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Collecting tzdata (from django)
  Using cached tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Using cached django-5.2.4-py3-none-any.whl (8.3 MB)
Using cached asgiref-3.9.1-py3-none-any.whl (23 kB)
Using cached sqlparse-0.5.3-py3-none-any.whl (44 kB)
Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.9.1 django-5.2.4 sqlparse-0.5.3 tzdata-2025.2

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

11) To Check Django version

```
>django-admin --version
```

django-admin → This is the **Django command-line tool** used to perform various administrative tasks in a Django project.

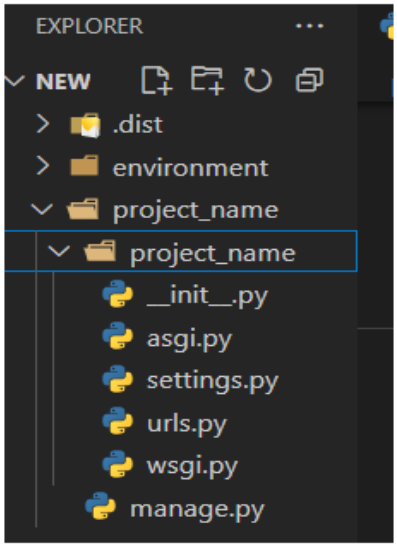
The output of this command result like this:

```
(environment) C:\Users\manid\OneDrive\Desktop\new>django-admin --version
5.2.4
```

12) And then start a project

```
>django-admin startproject project_name
```

It will create a folder named project_name with the files in it:



13) Run the Django project

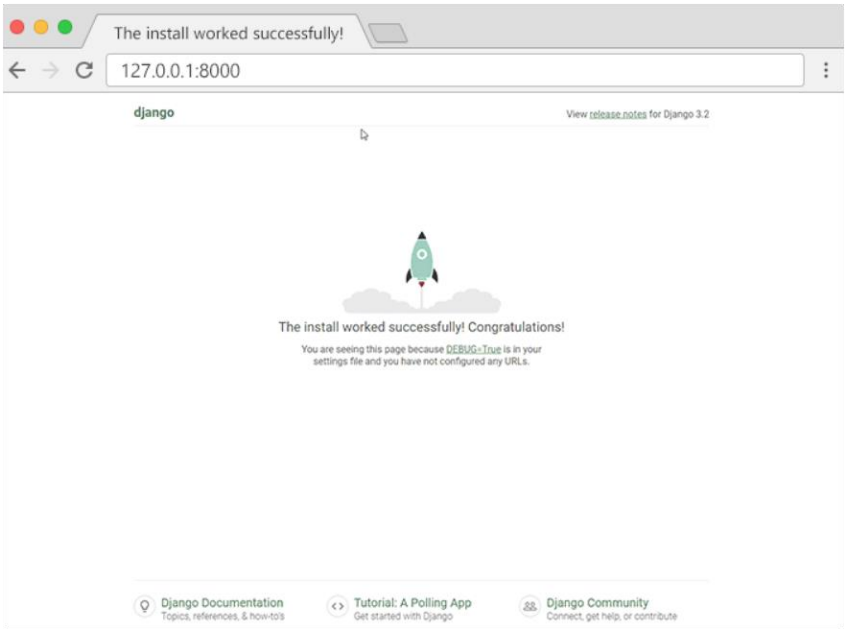
```
python manage.py runserver
```

When you run this you will see the following screen:

```
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, content
pes, sessions.
Run 'python manage.py migrate' to apply them.
August 02, 2025 - 16:36:20
Django version 5.2.4, using settings 'project_name.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

Move the cursor and click on follow link on the url "http://127.0.0.1:8000"

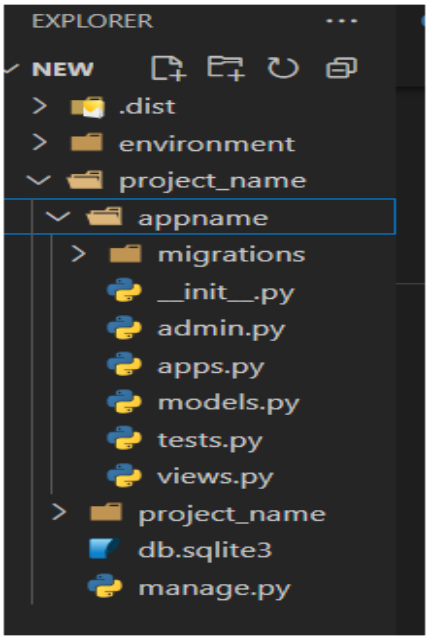
The Result is:



14) And then create an app. **What is an App? You cannot have a web page created with Django without an app. An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.** This is where we gather the information we need to send back a proper response.Django views are Python functions that take http requests and return http response, like HTML documents.

```
(environment) C:\Users\manid\OneDrive\Desktop\new\project_name>python manage.py startapp appname
```

Then the appname contains the following folders



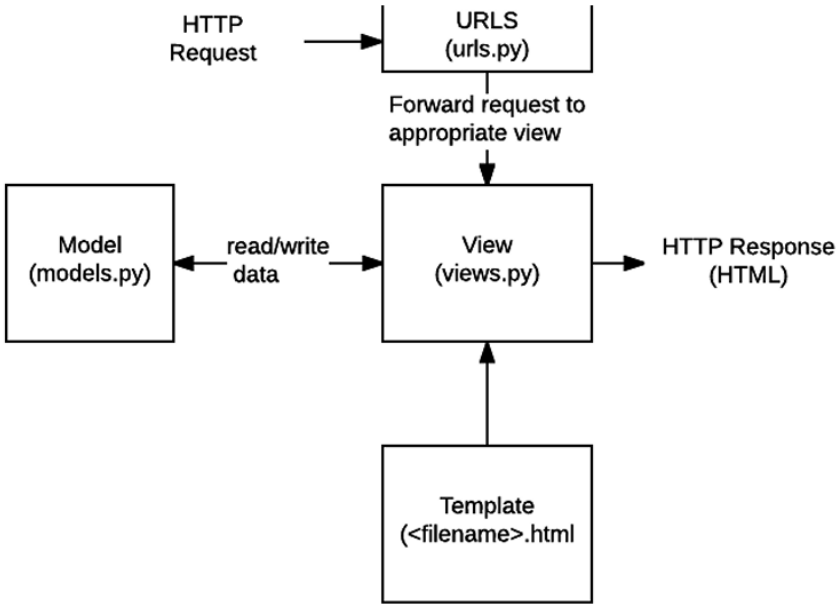
Note:

- Django-admin startproject project_name
- If we give space in between start and project it will through an error below is the image for that

```
(satya_env) C:\Users\manid\OneDrive\Desktop\Django files>django-admin start project project_name
No Django settings specified.
Unknown command: 'start'. Did you mean startapp?
Type 'django-admin help' for usage.
```

- If we want to execute the command then no need to give space between the start and project.
- To run the server first switch your directory to your project by typing the command **cd project_name** and then runserver using the command **python manage.py runserver**
- **We need to run the server inside the project directory because the manage.py file, which is used to manage the Django project, is located there.**
- **ASGI** stands for **Asynchronous Server Gateway Interface**.
  It allows Django to handle **multiple requests simultaneously**, making it suitable for **asynchronous tasks**, such as Web Sockets and real-time updates.
- **WSGI** stands for **Web Server Gateway Interface**.
  It is a standard interface between **web servers** and **Python web applications** used for communication and handling HTTP requests.

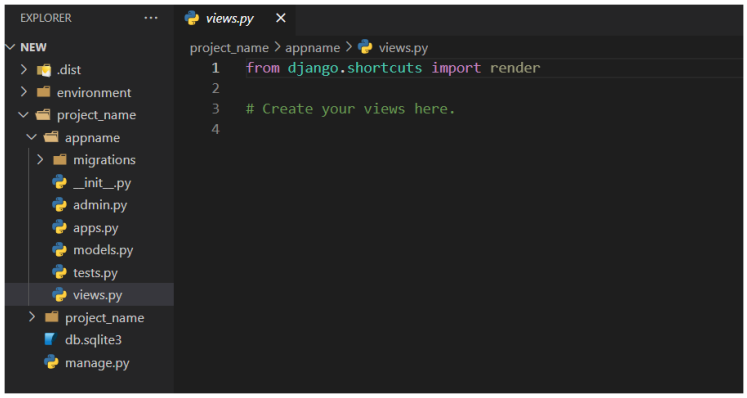
- When the **virtual environment** is activated and the Django project is created, the **default database added is sqlite3**, which is a lightweight, file-based database.
- http request→ urls→ view checks what it wants→ model.py→templetes



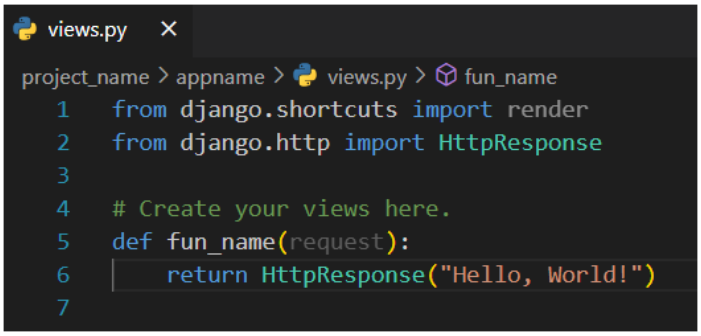
Views:

Django views are Python functions that take http requests and return http response, like HTML documents. A web page that uses Django is full of views with different tasks and missions. Views are usually put in a file called views.py located on your app's folder.

In the appname folder of vs code there is a file called view.py there we have to write the code and the basic file will look like this.



Replace it with the content:



Line by Line Explanation:

from django.shortcuts import render

- **Purpose**: Imports the render function, which is used to render HTML templates.
- **Used when**: You want to return a full HTML page using a .html file.

from django.http import HttpResponse

- **Purpose**: Imports HttpResponse, which is used to send raw text (or any content) back to the browser.
- **In this code**: Used to return the simple "Hello, World!" message.

def fun_name(request):

- **Defines a view function**.
- fun_name is just a placeholder — you can name this anything (e.g., home, index, etc.).
- request: An object representing the HTTP request received from the browser (contains data like method, headers, user info, etc.).

return HttpResponse("Hello, World!")

- This line sends a simple **plain-text response** to the browser.

URL's: **There are many ways to create url's file**

1) Create a file named urls.py in the same folder as the views.py file, and type this code in it:

views.py | urls.py

project_name 〉 appname 〉 urls.py 〉 ...

```
1  from django.urls import path
2  from . import views
3  urlpatterns=[
4      path('u/', views.fun_name, name="app_name") ]
```

There some other alternative ways to create urls inside our App:

🔹 2. Use `from views import ...` instead of `from . import views`

```
python

from django.urls import path
from .views import fun_name

urlpatterns = [
    path('u/', fun_name, name="app_name")
]
```

Here we are directly importing the function name from views.py file

2) Second way is to directly creating the url inside the projectname folder without the need for url's file in app folder

views.py | settings.py | urls.py | models.py | index.html

project_name 〉 project_name 〉 urls.py 〉 ...

```
1  from django.contrib import admin
2  from django.urls import path
3  from appname import views  # Import directly from app
4
5  urlpatterns = [
6      path('admin/', admin.site.urls),
7      path('u/', views.fun_name),  # Directly define the URL here
8  ]
9
```

3. Use a different URL path

You can customize the route to something more meaningful:

```python
path('greeting/', views.fun_name, name="greeting_view")
```

Now, accessing `http://localhost:8000/greeting/` will show "Hello, World!"

When some one visits the above url it will redirect you to fun_name which is present in the views.py file

from django.urls import path

- **Purpose**: Imports the path() function from Django.
- **Use**: This function is used to define **URL patterns**.
- You use it to map a specific URL to a **view function**.

from . import views

- **. (dot)** means "import from the **current directory**".
- This imports your **views.py** file, where you defined your view functions (like fun_name).
- This allows you to **call those view functions** from the urls.py file.

urlpatterns = [...]

- urlpatterns is a **list of URL routes**.
- Django looks inside this list to know **what view to show** for each URL.
- Each item inside is a path(...) function.

Inside the path(...) function:

path('u/', views.fun_name, name="app_name")

1. 'u/'

- This is the **URL path** that the user types in the browser.
- Example: http://localhost:8000/u/
- The 'u/' URL is now mapped to your view function.

2. name="app_name"

- This gives the URL pattern a **name** so it can be referred to elsewhere in Django (especially in templates and redirects).

➔ (Do this step when you create url file in app) Then the next step is in project file switch to urls.py file and add the paths to it.

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

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

from django.contrib import admin

- This **imports Django's built-in admin module**.
- It allows you to use the **admin site** at /admin/.
- Admin lets you manage models (add/edit/delete) through a web interface.

✅ **from django.urls import path**

- Imports the path() function.
- Used to define URL patterns (i.e., which view should be shown for which path).

✅ **from django.urls import include**

- Imports the include() function.
- include() is used to **include another URL configuration file**, usually from an app.

✅ **urlpatterns = [...]**

- A list of all your **URL routes**.
- Django checks this list to figure out **which view to call** for an incoming request.

🔷 **Inside urlpatterns:**

🔹 path('', include('app_name.urls'))

- The **empty string ''** means this is the **root URL** (i.e., http://localhost:8000/).
- include('app_name.urls') tells Django to **look inside the app named app_name** for another urls.py file.
- That file (app_name/urls.py) will contain more detailed routes.

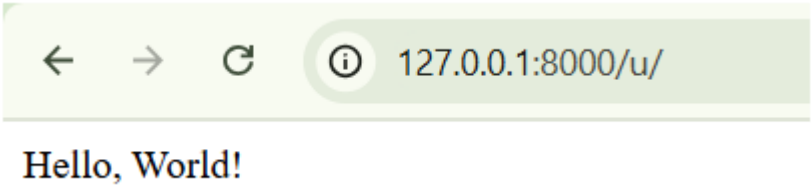
🔹 path('admin/', admin.site.urls)

- This enables the Django **admin panel** at http://localhost:8000/admin/.
- Used by superusers and staff to manage database content via web UI.

Run this command

```
python manage.py runserver
```

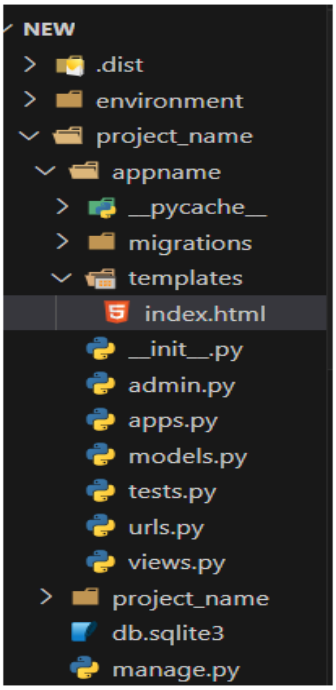
You will see the following output. This is the basic hello world program



Note: For small mini project like printing hello world or names of some thing we need not to edit the settings in the projects meanwhile for others we have to add the whatever apps we are created to the installed apps in settings.

Templates:

- ➔ To create and use templates we have to create a folder in our app called templates inside that we can create 'n' number of html files.
- ➔ And for creating css files we have to create another folder called static and then we can create 'n' number of css files in it.



- ➔ Open the html file and write the code:

➔ In this html code just focus on the h1 tag there we will see two flower braces. If we give only one flower brace then in output it will show like “This is {name}”. if we give two flower braces then it will take the value dynamically and in output it will show like “This is satya”.

How to take values dynamically in django using html , render and html, loader:

➔ We have two types to get the output one is through render method and another one is through loader method.

1) Taking the values dynamically using the render method:

- Create folder -> open it -> type cmd -> open it-> type code . -> visual studio will open -> python -m venv environment for creating virtual environment -> then pip install Django -> Django-admin startproject projectname-> cd projectname -> Django-admin startapp appname -> inside the app create a folder called migrations and templates. -> inside the templates folder create a file called index.html
- You will see some default folders in the migrations file which are pycache and init.py we didn’t do any work on them.
- In the html file for the default structure of it click the buttons shift+1 and hit tab which gives you default structure.
- There type the following:

index.html | models.py | views.py

app1 > templates > index.html > ...

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1> This is {{name}},{{roll_no}}</h1>
</body>
</html>
<!-- ekkada name ni dynamic ga declare chastunam -->
```

Explanation:

1. For creating dynamic values we give 2 flower braces. If we give only one flower brace the above code will show the output such as “This is {name},{rollno}”.
2. To take dynamic values we have to give 2 flower braces.

- After writing html code in templates folder then go to views.py file and write the following code.

1) This is one way through render method

views.py X index.html

project_name > appname > views.py > ...

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

# Create your views her.
def fun_name(request):
    return render(request,"index.html",{'name':'Satya','roll_no':1234})

```

2) This is through loader method:

```
from django.http import HttpResponse
from django.template import loader

def fun_name(request):
    template = loader.get_template('index.html')
    context = {'name': 'Satya', 'roll_no': 1234}
    return HttpResponse(template.render(context, request))
```

Create urls.py in your app folder

app1 > urls.py > ...

```python
from django.urls import path
from . import views
urlpatterns = [
    path('', views.fun_name, name='index'),
]
```

Go to urls.py file in project_name project_name

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

urlpatterns = [
    path('', include('app1.urls')),

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

OUTPUT:

This is Satya,1234

Create urls.py in your app folder

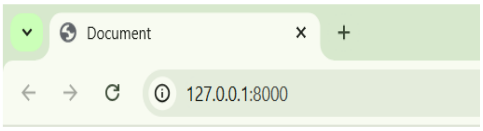
app1 > urls.py > ...

```python
from django.urls import path
from . import views
urlpatterns = [
    path('', views.fun_name, name='index'),
]
```

Go to urls.py file in

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

urlpatterns = [
    path('', include('app1.urls')),

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

OUTPUT:



This is Satya,1234

Explanation:

This is one method of getting response render(request,"index.html",{'name':'satya','roll_no':1234}). Here we create a dictionary to give values dynamically to display , "index.html" is the file name where we write our html code. After writing code in views.py then create urls.py file in app folder

Then go for urls.py file in projects and include the path of your app Here app1.urls is my file name you have to write the app name of your app. After go to terminal → switch to command prompt and type python manage.py runserver to run our application, you will see the output.

Another way is through loader method all the above code is same for that except, we make changes in only one file that is in views.py file

🔍 **Line-by-Line Explanation:**

1. `from django.http import HttpResponse`
   → Imports the `HttpResponse` class to send a response back to the browser.
2. `from django.template import loader`
   → Imports Django's `loader` to manually load a template.
3. `def fun_name(request):`
   → Defines a view function called `fun_name` that handles an HTTP request.
4. `template = loader.get_template('index.html')`
   → Loads the HTML template named `index.html` from your templates directory.
5. `context = {'name': 'Satya', 'roll_no': 1234}`
   → Prepares data to pass into the template.
6. `return HttpResponse(template.render(context, request))`
   → Renders the template with the provided data and returns the final HTML as a response.

How to create models and add data to databases, update, delete, save the data in databases:

➔ To create models go to models.py in app folder and write the following code.

Create a model:

These models provides all the field types (like CharField, IntegerField, etc.).

```
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    roll_no = models.IntegerField()
```

CharField creates a text field in the database. max_length=100 means the name can be up to 100 characters long. There is also another field called text-field which takes infinite length of characters.

IntegerField creates a field for whole numbers (integers). Used here to store a student's roll number.

class Student(models.Model) Declares a model class named Student. It inherits from models.Model, which tells Django this is a database model.

NOTE: when you install and create a new Django project, the **default database is SQLite**

Other fields in Django:

✅ 1. String/Text Fields

Field	Description
`CharField(max_length=...)`	Short strings (e.g., names, titles)
`TextField()`	Long text (e.g., descriptions, articles)
`SlugField()`	URL-friendly text (e.g., `my-blog-post`)
`EmailField()`	Validates email addresses
`URLField()`	Validates URLs
`FilePathField(path=...)`	File path on the system

✅ 2. Numeric Fields

Field	Description
`IntegerField()`	Whole numbers
`BigIntegerField()`	Very large integers
`PositiveIntegerField()`	Only positive integers
`FloatField()`	Decimal numbers (float)
`DecimalField(max_digits=..., decimal_places=...)`	Fixed precision decimals (good for money)
`SmallIntegerField()`	Smaller integers

✅ 3. Boolean and Null Fields

Field	Description
`BooleanField()`	`True` or `False`
`NullBooleanField()`	`True`, `False`, or `None` *(Deprecated in Django 4.0+)*

✅ 4. Date and Time Fields

Field	Description
`DateField()`	Stores only the date
`TimeField()`	Stores only the time
`DateTimeField()`	Stores both date and time
`DurationField()`	Stores time duration (like difference between two times)
`AutoField()`	Auto-incrementing primary key (used by default)

Migrate

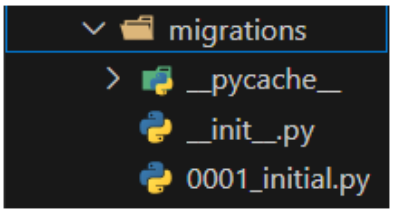
Now when we have described a Model in the models.py file, we must run a command to actually create the table in the database.

```
>python manage.py makemigrations appname
```

Which will result in:

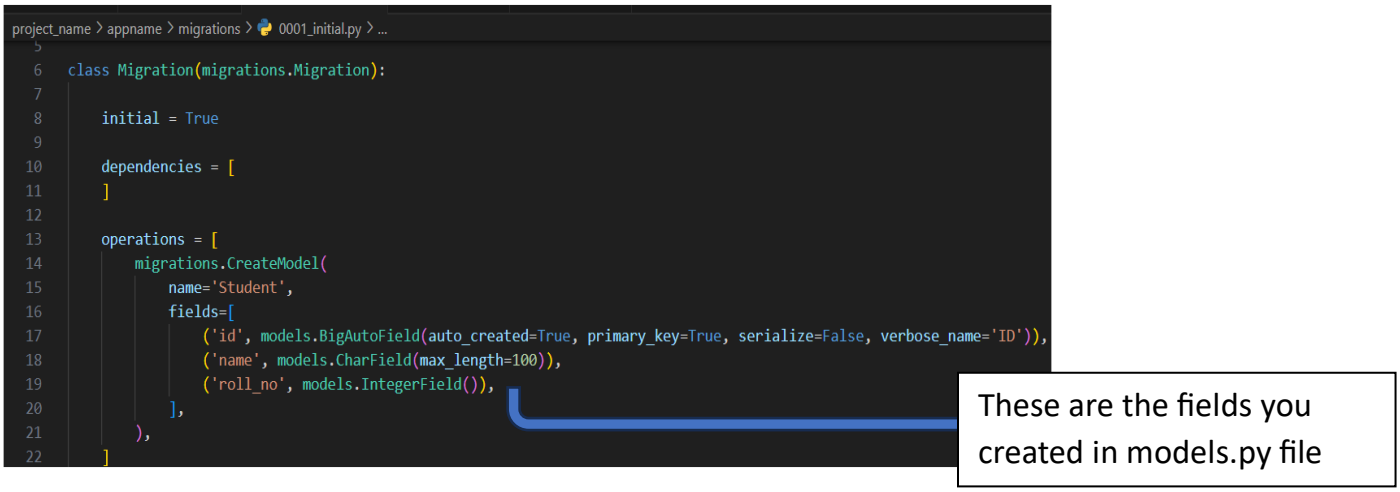
```
(environment) C:\Users\manid\OneDrive\Desktop\new\project_name>python manage.py makemigrations appname
Migrations for 'appname':
  appname\migrations\0001_initial.py
    + Create model Student
```

Now Django creates the file and store it in migrations folder like this:



Intially you wont see this 0001_initial.py file in migrations folder. This only visible when you run makemigrations command

You will see the following picture when you click on 000_1initial.py file in migrations



These are the fields you created in models.py file

The table is not created yet, you will have to run one more command, then Django will create and execute an SQL statement

```
>python manage.py migrate
```

Which will result in this output:

```
Operations to perform:
  Apply all migrations: admin, appname, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying appname.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Now you have Student table in your database

Add Records

We will use the Python interpreter (Python shell) to add some members to it.

To open a Python shell, type this command:

```
python manage.py shell
```

Basic operations with Student database:

1. Create a Student object and save it to the database

```
python

student = Student(name='Satya', roll_no=1234)
student.save()
```

This will create an object called student and store the data name and rollno to the database. And save it

📋 Student Retrieval Techniques (based on the record: `name='Satya', roll_no=1234`)

S.No	Query	Purpose	Returns
1	`Student.objects.all()`	Get **all students**	QuerySet of all records
2	`Student.objects.values()`	Get all students as **dicts**	QuerySet of dicts
3	`Student.objects.get(id=1)`	Get by **ID = 1**	Single object
4	`Student.objects.get(name='Satya')`	Get by **name** (must be unique or 1 record only)	Single object
5	`Student.objects.get(roll_no=1234)`	Get by **roll number**	Single object
6	`Student.objects.filter(name='Satya')`	Filter by name (safe for multiple)	QuerySet
7	`Student.objects.filter(roll_no=1234)`	Filter by roll number	QuerySet
8	`Student.objects.filter(id=1).first()`	Get first matching object safely	Single object or None
9	`Student.objects.filter(name='Satya').values()`	Filter + values as dicts	QuerySet of dicts
10	`Student.objects.filter(name__icontains='sat')`	Case-insensitive match	QuerySet
11	`Student.objects.filter(roll_no__gte=1000)`	Roll number ≥ 1000	QuerySet

To update the data:

🟦 4. Update data

```python
a1 = Student.objects.get(id=1)
a1.name = 'Updated Name'
a1.save()  # Save the changes
```

To delete the data:

🟦 5. Delete data

```python
a1 = Student.objects.get(id=1)
a1.delete()  # Deletes the object from the database
```

BlockQuotes:

✅ In the `base.html` (parent template):

```django
{% block block_name %}
    Default content (optional)
{% endblock %}
```

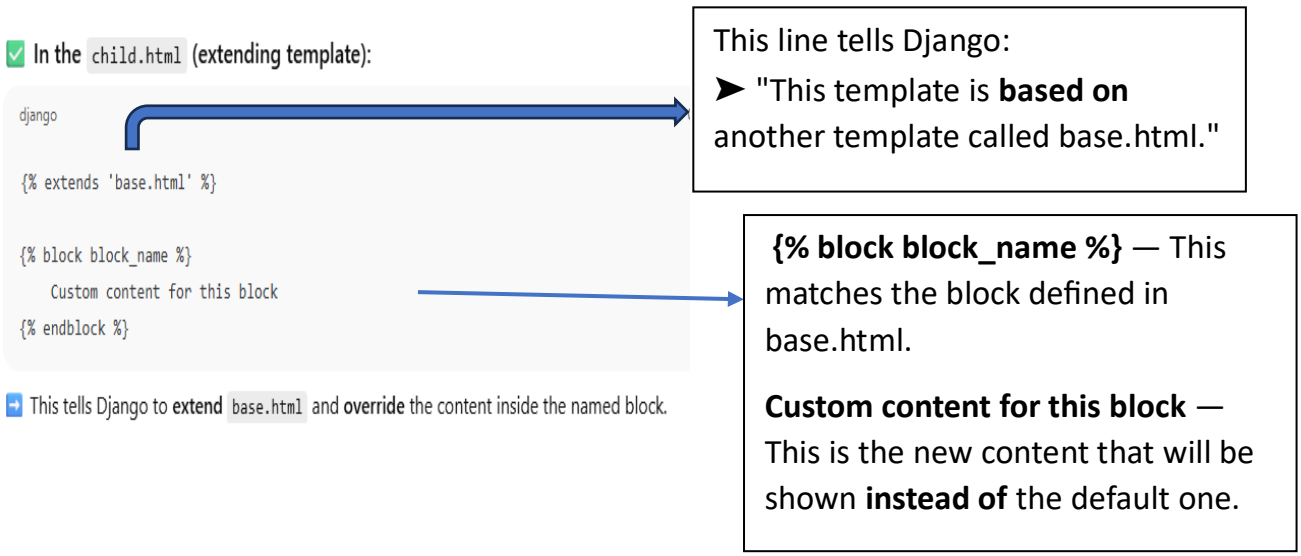
➡️ This defines a **block** that can be replaced in child templates.

This starts a named block section in Django template. To define a **replaceable area** in your base template that can be **overridden** by child templates using {% extends %}.

It's the **name/ID** of the block. It can be anything like content, title, header, footer, etc. Used by child templates to **match** and override the block.

Default is optional the code which you write inside this in base.html file will only display the code when there is no child extensions for this .

➔ {% endblock%} tells that the block quote has ended for this Django template.



Example code for BlockQuotes: create a file called base.html in inside templates folder which is inside the appname folder.

base.html | child.html | urls.py | views.py

project_name > appname > templates > base.html > html > body > div > h1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        {% block content %}
        <h1> You will see this when there is no child class</h1>
        <!-- Content will be injected here -->
        {% endblock %}
    </div>

</body>
</html>
```

Create another file called child.html in the same place where base.html is created

base.html | child.html | urls.py | views.py

project_name > appname > templates > child.html > ...

```
<!-- templates/child.html -->
{% extends 'base.html' %}

{% block content %}
    <h2>Hi, this is Visual Studio Code</h2>
{% endblock %}
```

Django – admin interface:

The Django Admin Interface is one of the most powerful features of the Django framework. It provides a ready-to-use interface for managing project data through models, allowing developers and site administrators to perform Create, Read, Update, and Delete (CRUD) operations with ease.

When we start a new Django project, Django automatically includes a built-in admin app (**django.contrib.admin**).

It offers:

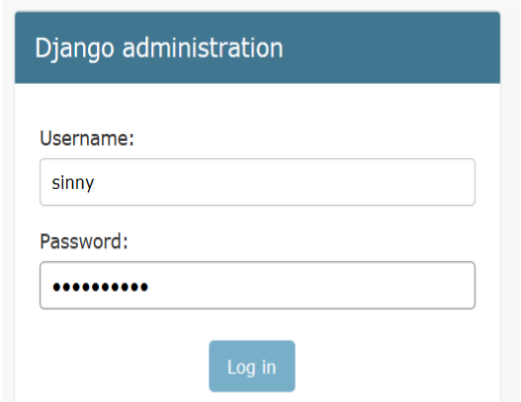
- CRUD operations on models
- User and permission management
- Model-level customizations
- A clean, responsive interface

Run the server using the command "python "

```
C:\Users\manid\OneDrive\Desktop\new\project_name>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 06, 2025 - 09:52:39
Django version 5.0.6, using settings 'project_name.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Click on this link and in the url type admin

Type the "\admin" in the above url and hit enter this will redirect you to the below image



The following page will be displayed. Generally there are two ways for creating this admin panel one is

1) By creating superuser
2) Staff user with permissions

1) By creating superuser:

- Has full access to the admin panel and all models.
- Can create, update, and delete any record.
- Created using createsuperuser.

2)Staff user with permissions:

- You can mark a user as a **staff member (is_staff=True)** and assign them specific permissions (e.g., **view**, **change**, **delete** for a **model**).
- Such users will also be able to log in to the admin panel but only see what they're allowed to manage.

Creating superuser for Django admin panel:

The above steps are from starting how we did it

```
(environment) C:\Users\manid\OneDrive\Desktop\new>django-admin startproject projectName
```

→ This will create a new project in your folder

```
(environment) C:\Users\manid\OneDrive\Desktop\new>cd projectName
```

→ Switch to your project

```
(environment) C:\Users\manid\OneDrive\Desktop\new\projectName>django-admin startapp appName
```

→ Create an app inside the projectName

Go to settings.py folder and in the installed apps add your name

settings.py

projectName > projectName > settings.py > ...

```
29
30
31  # Application definition
32
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'appName'
41  ]
```

→ Here the name of the app which I created is

In the models.py add the model

models.py

projectName > appName > models.py > Student

```python
from django.db import models

# Create your models here.
class Student(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    email = models.EmailField()
```

I am creating a class called student which will take name, age and email address of a student

Create database and table run the following commands:

1) Python manage.py makemigrations
2) Python manage.py migrate

```
(environment) C:\Users\manid\OneDrive\Desktop\new\projectName>python manage.py makemigrations
Migrations for 'appName':
  appName\migrations\0001_initial.py
    + Create model Student
```

The resultant of the command python manage.py

```
(environment) C:\Users\manid\OneDrive\Desktop\new\projectName>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, appName, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying appName.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

The resultant of the command python manage.py

Run the command python manage.py createsuperuser

```
(environment) C:\Users\manid\OneDrive\Desktop\new\projectName>python manage.py createsuperuser
Username (leave blank to use 'manid'): pikachu
Email address: pokemon@cartoon.com
Password:
Password (again):
The password is too similar to the email address.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

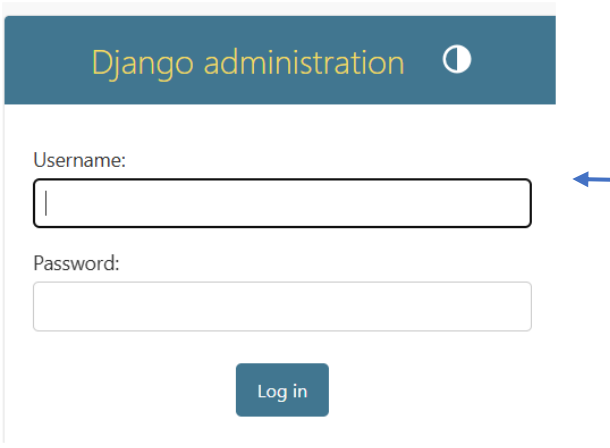
This will create a new user called pikachu and provide the necessary fields asked by the

```
(environment) C:\Users\manid\OneDrive\Desktop\new\projectName>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 06, 2025 - 11:25:32
Django version 5.2.4, using settings 'projectName.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

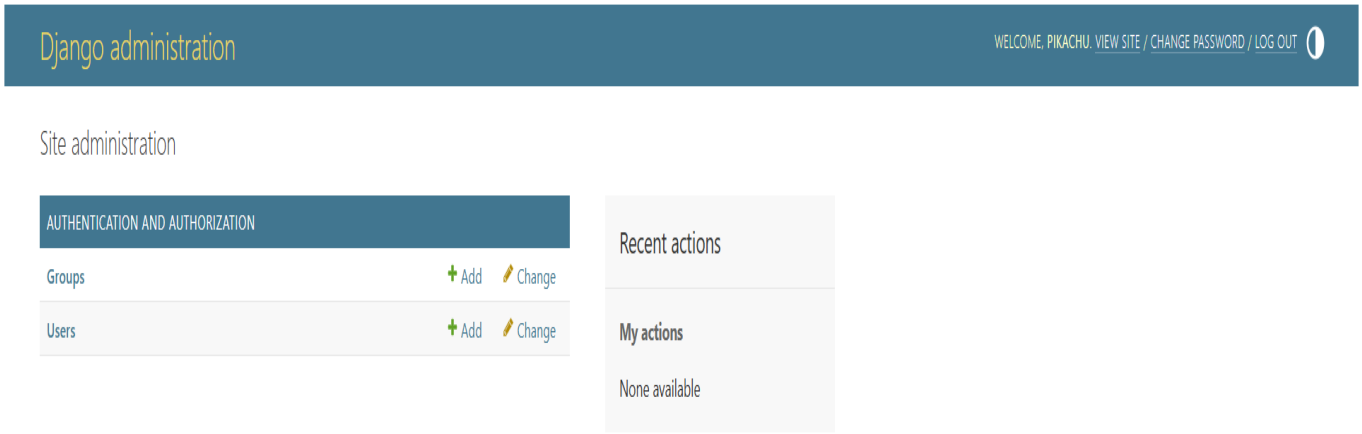
Run this command and click on the

127.0.0.1:8000\admin

Type admin at the end of the url this will redirect you to the below image



Login with details where you entered in the superuser



Register the created model in admin.py file of appName

models.py admin.py

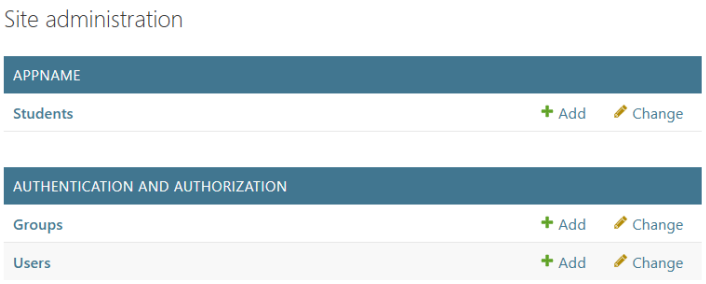
projectName > appName > admin.py

```
1  from django.contrib import admin
2  from .models import Student
3  admin.site.register(Student)
```

This will import the built-in admin

This imports the Student model from the current app's models.py

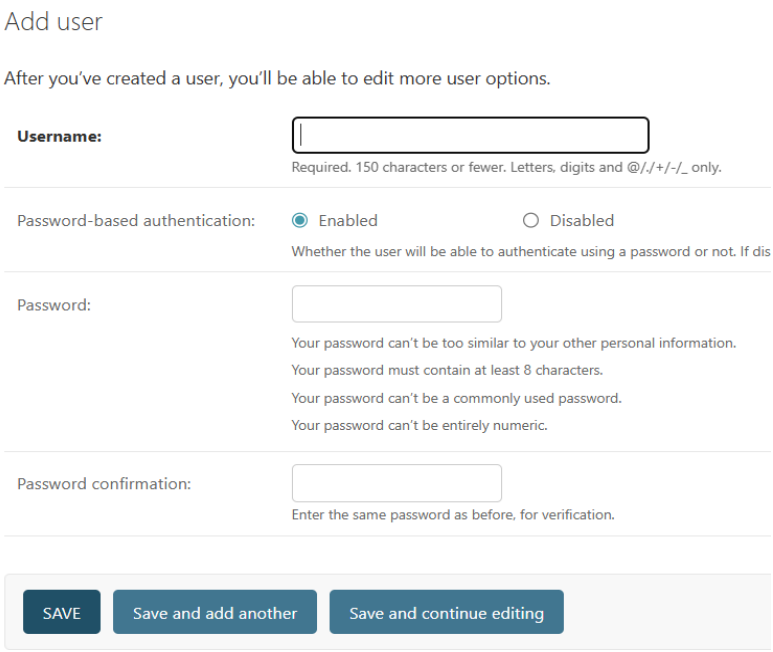
This **registers the Student model** with the Django admin site.After registering, you can view, add, update, and delete Student records from the admin panel.Without this line, your model **won't appear** in the admin interface.



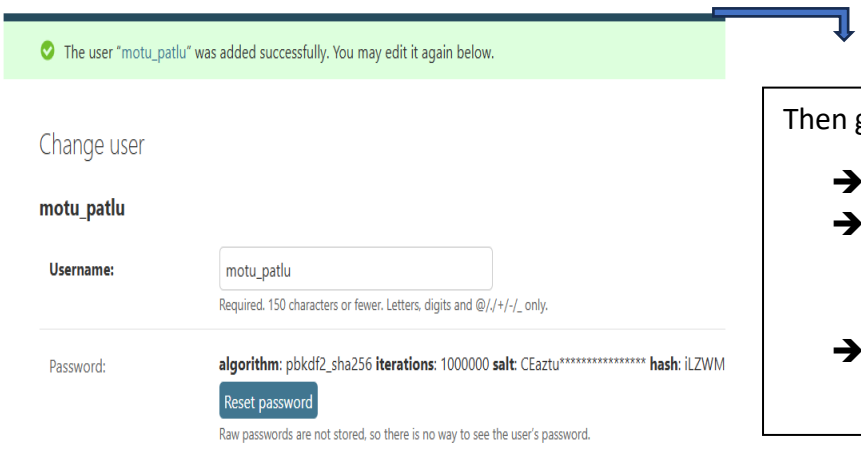
You will see this after you register your model in admin.py file

Inside the admin panel we can also create a new user who are called as **staff Users** for your models

For creating staff users → Inside the admin panel → go to Users column in the Authentication and Authorization section → There we can create a new user and groups → click on add User option → It will take you to this pane



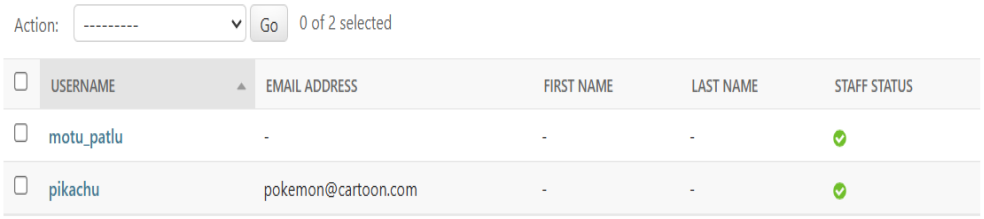
Give the necessary details and save it. It will result in following page after you saved it



Then go to permission section and

- ➔ Click the tick mark on the staff status
- ➔ Go to user permissions field (this will tells what are the permissions you want to give your staff user)
- ➔ Select that he or she can add, delete, update and view student and click save

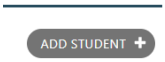
It will result in following page. We can click on the user and update the fields when ever you want



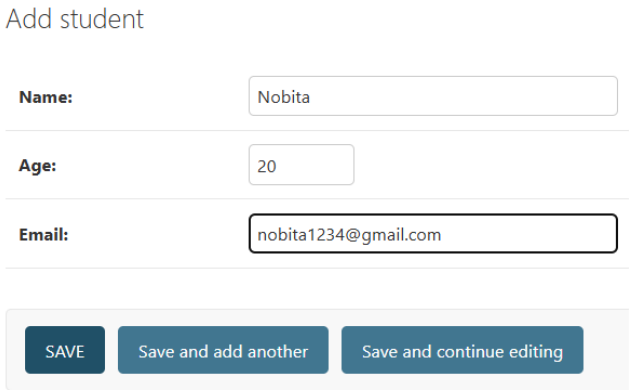
Click on motu_patlu to change the data if

➔ There are two ways in adding data to the students Table

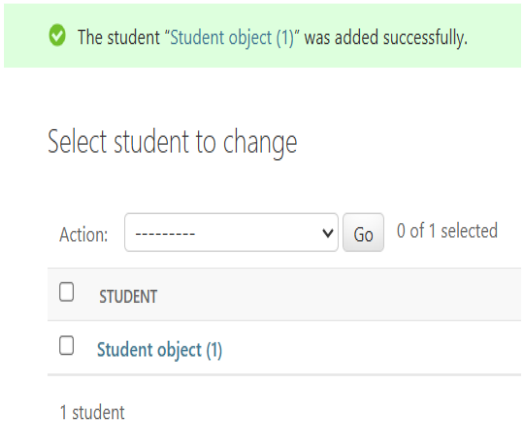
1.Adding through admin panel prompt



You will see this add student field in the students site in admin panel click on it



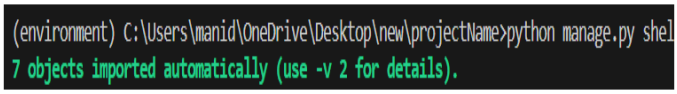
Fill the necessary fields and save the data. These are the Fields you are given in the model.py file. After saving it
The result is:



In this way we can add n number of students to the Database. we can also perform delete operation on that.

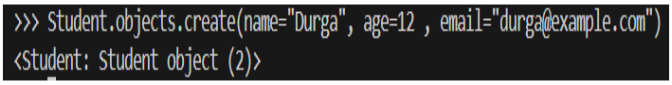
2. Adding through shell in command

In the command prompt type the command "python manage.py shell" to open the python's interactive shell and create an object for student and add details of that

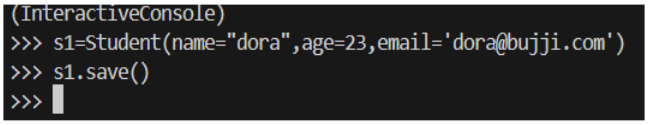


This will import the 7 helpful objects to run code
Here you can add student data using 2 methods:

1)

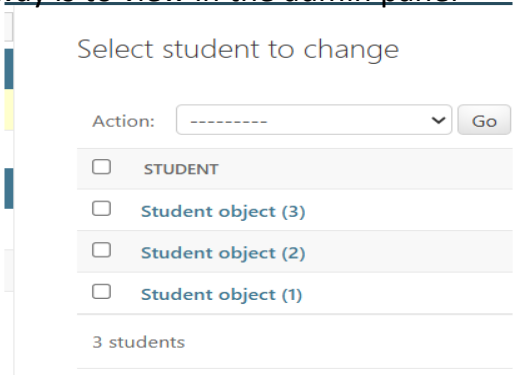


2)



Both the above methods will save the data to the Students class. To display the data that is stored in the student class. Again there are two ways to view the data stored in Database.

1st way is to view in the admin panel



2)second way to view is this in the command prompt:

```
>>> Student.objects.values()
<QuerySet [{'id': 1, 'name': 'Nobita', 'age': 20, 'email': 'nobita1234@gmail.com'}, {'id': 2, 'name': 'Durga', 'age': 12, 'email': 'durg
a@example.com'}, {'id': 3, 'name': 'dora', 'age': 23, 'email': 'dora@bujji.com'}]>
```

How to save data:

```
>> s4=Student(name='Vijay',age=30,email='vijaydevarakonda@gmail.com')
>> s5=Student(name='Rashmika',age=30,email='rashmika@gmail.com')
>> s4.save()
>> s5.save()
```

This is one way of saving data

```
>> s6=Student(name='Hasini',age=25,email='hasini@email.com')
>> s7=Student(name='Kingkom',age=50,email='forest@gmail.com')
>> s=[s6,s7]
>> for i in s:
..  i.save()
```

This is one way using for loop, we declare the values in separate variables and keep them in a list, and then using iteration through for loop we save the data in database table

Basic Operations like update, delete, find data in admin page:

From Shell:

```
>>> Student.objects.all()
<QuerySet [<Student: Student object (1)>, <Student: Student object (2)>, <Student: Student object (3)>, <Student: Student object (4)>, <
Student: Student object (5)>, <Student: Student object (6)>, <Student: Student object (7)>]>
>>> Student.objects.values()
<QuerySet [{'id': 1, 'name': 'Nobita', 'age': 20, 'email': 'nobita1234@gmail.com'}, {'id': 2, 'name': 'Durga', 'age': 12, 'email': 'durg
a@example.com'}, {'id': 3, 'name': 'dora', 'age': 23, 'email': 'dora@bujji.com'}, {'id': 4, 'name': 'Vijay', 'age': 30, 'email': 'vijayd
evarakonda@gmail.com'}, {'id': 5, 'name': 'Rashmika', 'age': 30, 'email': 'rashmika@gmail.com'}, {'id': 6, 'name': 'Hasini', 'age': 25,
'email': 'hasini@email.com'}, {'id': 7, 'name': 'Kingkom', 'age': 50, 'email': 'forest@gmail.com'}]>
```

In the above diagram shows how to retrieve id's and values in a database using the commands: Student.objects.all(), Students.objects.values()

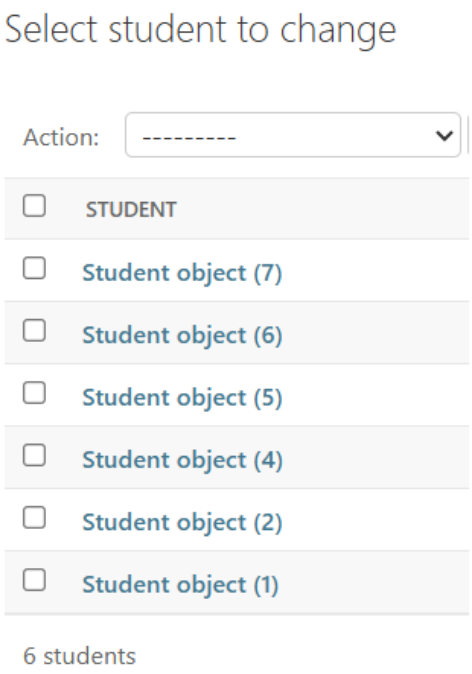
```
>>> Student.objects.get(id=2)
<Student: Student object (2)>
>>> alpha=Student.objects.get(id=2)
>>> alpha.name
'Durga'
>>> alpha.email
'durga@example.com'
```

This is the way we can retrive objects. In place of Id we can place any field and their respective value.

```
>>> s=Student.objects.get(id=3)
>>> s.delete()
(1, {'appName.Student': 1})
```

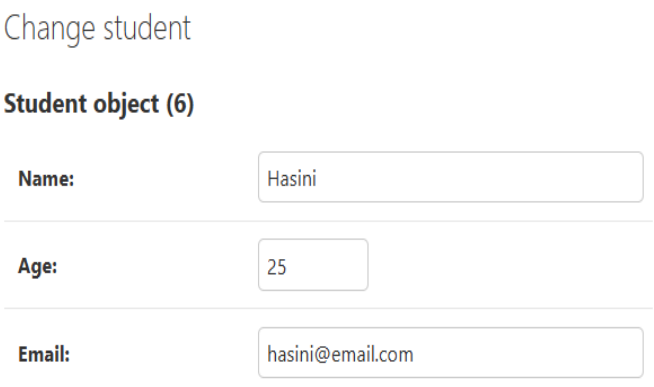
This is the way we can delete objects or un necessary data in the table.

After all these observations the result of the above operations are like this:

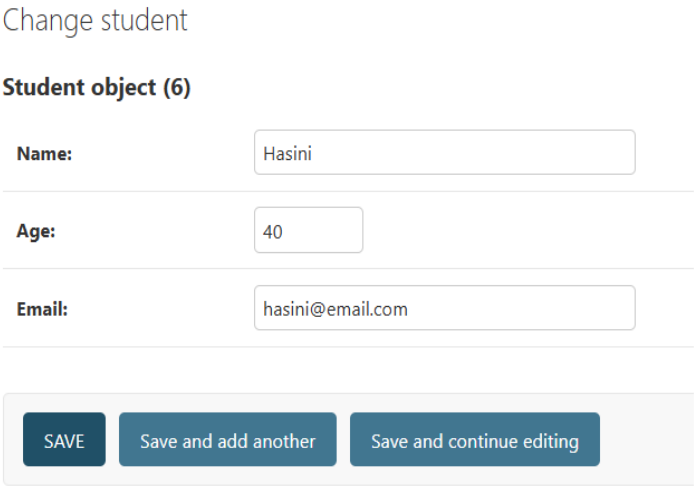


BASIC operations from admin platform:

1)Update operation (updating the value of age of student 6):

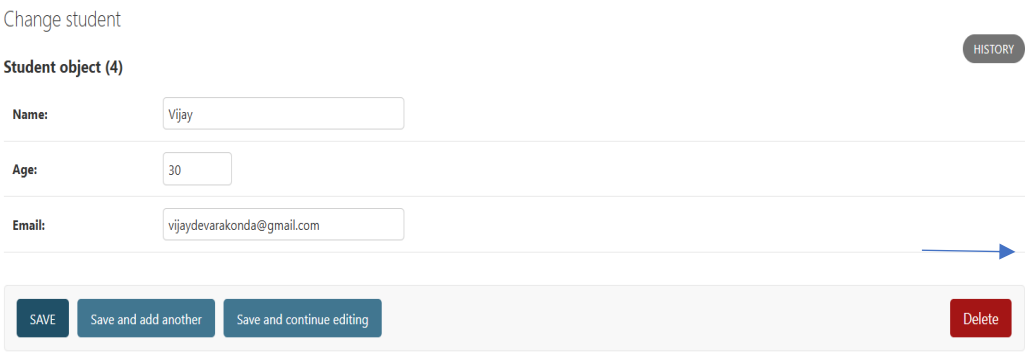


Before Updating the value of age



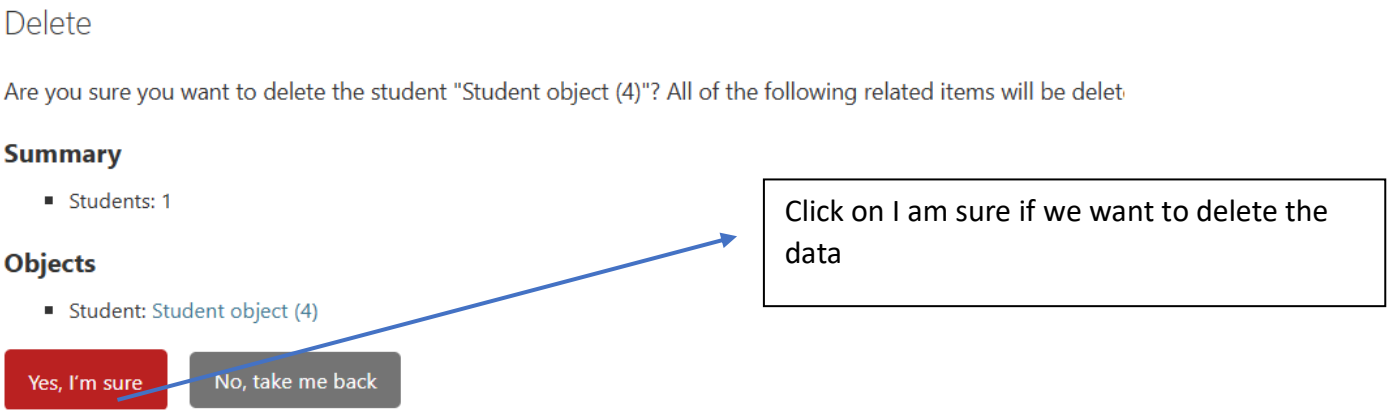
After updating the value click on save button

It’s Time for delete operation:



Click on delete option, it will show you a message for confirmation

When you click on delete the following box will appear:



Click on I am sure if we want to delete the data