Django Tutorial For Beginners

Made By : Mariam Tamer

Table Of Contents :

1. General Info……..1

2. Setting Up & Installing Django …. . 5

3. Django Files And How They Work....12

4. Create A New App ……...…..... 18

5. Database Model Using PostgreSQL….24

6. Database Form ……40

7. Registration Form…… 53

8. Login & Logout System …… 71

9. Class Based Views
(Create, Update, and Delete) …..78

0 General Info

Created	@October 8, 2023 9:48 AM
Made By	Mariam Tamer

Difference between package and framework

- **package :** set up programs/code that provides specific functionality
- **framework :**

  - predefined architecture that provides abstract functionality

  -speeds up process of creation

  -focus is not on ‘how’ but on ‘what’

---

django

provides full stack + rest APIs

---

package manager importance :

- install /uninstall packages
- check dependencies
- check package version
- control package behaviour

our package manager in python —> **pip**

Difference between package manager in a specific path and a global one

-npm i —>

- specific path
- used only once

-npm g —>

- global on system
- used many times
- one version and if it becomes old, u have to edit it for every project

How to install your package manager in a specifiv path/location in Django?

Using **virtualenv**

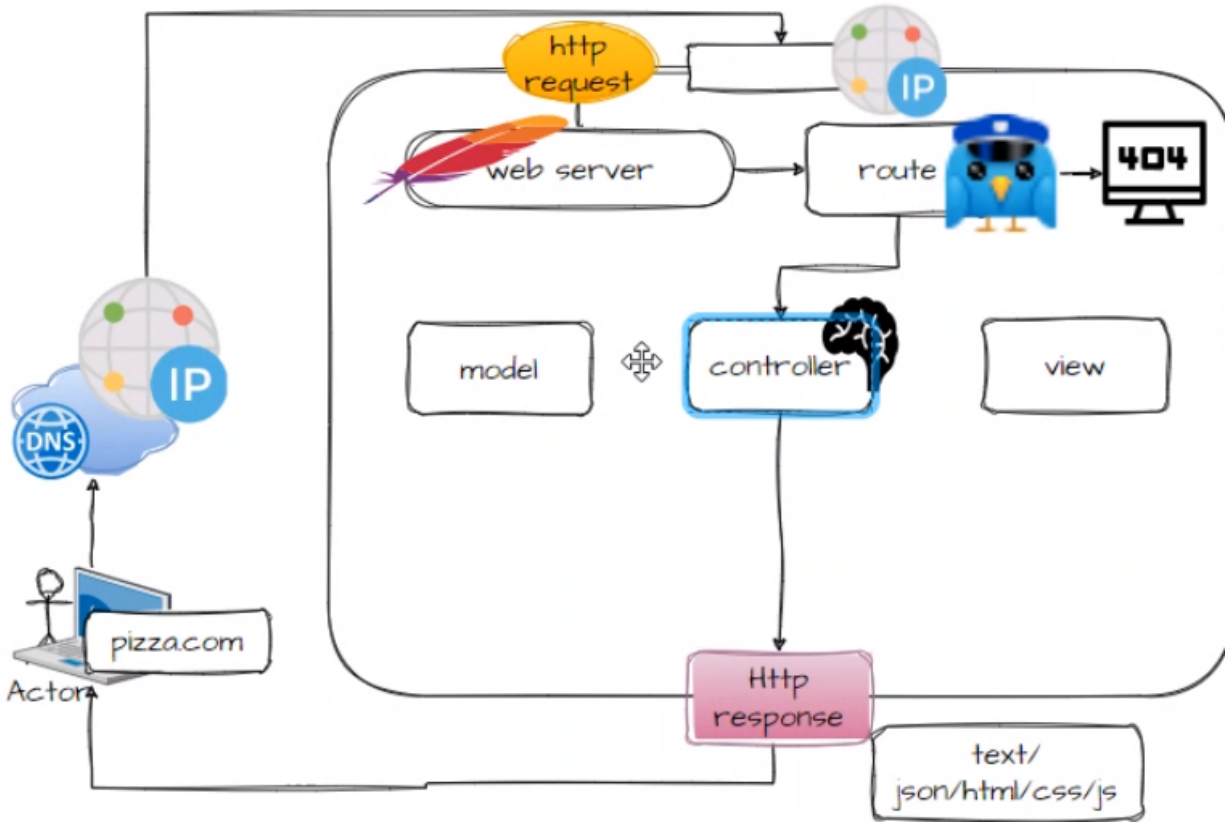
Note :

pip freeze —> tells you which dependencies/packages you have and their versions.

Design Pattern

-Biggest one is **MVC** which consists of :

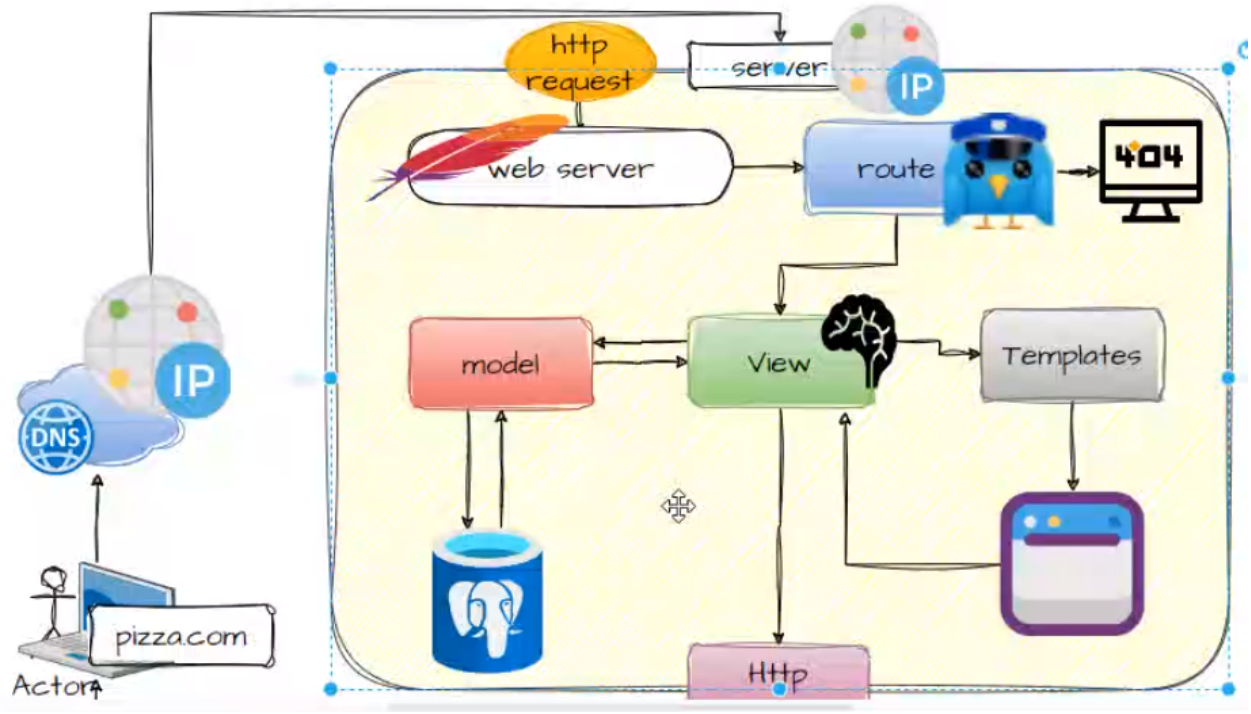
- **model :** talks to DB
- **controller** : Brain of framework
- **view :**



-If my sub ip is not correct, for ex : linkedin.com/dkenondndjkfndkjfn , the route won't let me pass to the controller, and instead would give an error of a HTTP Status Code : 404

-But if it's correct, the the route would take me to the controller, and the controller would let me pass to the HTTP response to give it back to the user.

In django, the concepts will be :



-The controller is called View (as it's the brain)

-The View is called Templates (as it's where your display will be)

-Model is the same , database of django

Setting Up & installing Django

Created	@October 6, 2023 9:03 PM
Made By	Mariam Tamer

********** Django Tutorial **********

Steps to create any Django project and run it on server:

1) Install Pythonv3.8 or newer

2) Install Virtual Environment

3) Create Virtual environment

4) Activate Virtual Environment

5) Install Django

6) Create Project

7) Run the project on server

Let's take you step by step :)

Install & Set Up Django

First: Install Pythonv3.8 or newer

- you need to have a **Python Version not older than 3.8**
- Because it contains **pip** and this is where your whole work is …

Second: Install Virtual Environment

- You'll need to work on a **virtual environment** to separate your work on your code editor from the outside hardware so when you import libraries or anything else, no conflicts happen, and so on ..

to install your virtual environment:

after you install a *Python Version NOT older than v3.8*, open your terminal and write this :

For windows :

pip install virtualenv

For Mac / Linux :

pip3 install virtualenv

To start a new project —> create a new virtual environment

Step 3 : Create Virtual environment

Method 1 :

virtualenv name

EX : virtualenv test

Method 2 :

On WIndows :

python -m venv name

EX : python -m venv test

On Ubuntu :

install the python3 venv package first —>

sudo apt install python3.10-venv

then create a new venv :

python3 -m venv project_name

EX : python3 -m venv test

Now you have created a new folder with the the name of the virtual environment, and this folder contains some packages like lib, pip, ….

Step 4 To activate your venv :

- **cd** to your created virtual environment folder (but if you're there, then it's gonna be **a dot "."**) , then type :

On Windows :

name\Scripts\activate … and S in Scripts is Capital

EX : test\Scripts\activate

On Linux :

source name/bin/activate

EX : source test/bin/activate

Step 5 : now you can install django …

pip install django

Step 6 : Now to start a new project

cd to your venv folder, then type :

django-admin startproject project_name

EX : django-admin startproject project

You'll realize a new folder is created with your project name, and if you open it, it contains two files, one of them is called **manage.py**

Step 7 : Now to run your django server,

cd to your newly created project name folder, and type :

python3 manage.py runserver (now yk python3 is for linux and for windows it's python)

this is gonna appear :

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
October 06, 2023 - 18:30:00
Django version 4.2.6, using settings 'project.settings'
```

Starting development server at **http://127.0.0.1:8000/**
Quit the server with CONTROL-C.

See the **http://127.0.0.1:8000/ ? This is your localhost server !**

However, your built in files haven't been installed yet because of this message : You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

This has something to do with the database you haven't created it. You're gonna learn about it later, but to ignore it, press :

ctrl + c

then type :

python3 manage.py migrate

now your files are installed, and you can open your project in Visual Studio Code by typing :

code .

now open your VSC terminal, either using : **ctrl +** ` or by choosing **Terminal** from the menu above, then **New Terminal**

and now you can run your server again : **python3 manage.py runserver**

Django Files And How They Work

Created	@October 6, 2023 9:38 PM
Made By	Mariam Tamer

What I'll discuss here :

1) Recap on last file, how to create any Django project.

2) Your created project built in files.

3) How Django works.

---

1) Let's recap our steps to create any Django project and run it on server:

1) Install Pythonv3.8 or newer

2) Install Virtual Environment

3) Create Virtual environment

4) Activate Virtual Environment

5) Install Django

6) Create Project

7) Run the project on server & migrate

On visual Studio Code, run your server :

python3 manage.py runserver

---

2) Now let's get to know the new installed built in files :

1) __init__.py

2) asgi.py

3) settings.py

4) urls.py

5) wsgi.py

6)db.sqlite3

1) __init__.py :

Contains your imported libraries, and you won't use it much.

---

2) asgi.py && 5) wsgi.py :

-Files related to server.

-You won't use them.

3) settings.py

The brain of the project. Contains everything.

-Contains your **SECRET_KEY** that you shouldn’t share with anyone.

-Contains your **installed apps** and any app you install ,you should write it here :

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

-Contains your **Frontend templates** :

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

-Contains your **used databases**. It has **sqlite3** so it could work, but you can use **PostgreSQL** for big projects.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

-Contains **password validation** :

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

-Contains **static url** or files such as **html/css/images**

STATIC_URL = 'static/'

4) urls.py

-Very important too, and you'll be using it for all your created paths.

-If you wanna create a path like :

facebook.com/groups

facebook.com/settings

facebook.com/profile

and so on.. you'll write them and create them there …

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

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

********** ((3)) How Django Works ? **********

-By creating **web applications** from your project.

-Project makes **many apps**. Each app is responsible for a path.

(Like I gave you the Facebook example before : groups - settings - profile) ,

your django project c**reates apps, one for each path** .. one for groups , another for settings, another for profile

-Each app contains the following :

- **Views**
- **Models**
- **Templates**
- **Urls**

Views :

-The leader of them all.

-Decides what is displayed, and what is not.

Models :

-Your **Database**

Templates :

-You create this.

-This is your created **Frontend.**

Urls :

-You create this too.

-This is different than the one in the project built in files.

Create A New App

Created	@October 6, 2023 10:59 PM
Made By	Mariam Tamer

1) Create a new App

-First : write the command in terminal

-Second : add the installed app in settings.py

-Third : create a new url.py (another one apart from the project's) in your app folder

-Fourth : Link your project urls.py to your app urls.py

-Fifth : add your functions in views.py and import them in urls.py

First :

In your VSC terminal, type :

python3 manage.py startapp app_name

FOR EX mine is : **python3 manage.py startapp pages**

(and remember that for **windows** it's **python not python3**)

you'll realize a new folder called app is created with the following files : **init**.py - admin.py - adds.py - tests.py - views.py and we've discussed what all of them does in the previous file.

Second :

in your **settings.py,** add the app.

Let’s say my app name is pages ,

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

-Where did this come from ? go to your pages app, you’ll find a file called **apps.py** and in there’ you’ll find a **class : class PagesConfig** …

-Also make sure to put on a **comma ‘,’** after every line

Third :

Create a new **urls.py i**n your Pages app (different from the one in your projects folder)

Fourth :

To **link** the **urls.py of your app to the urls.py of your projec**t, go to your **urls.py in project** and you’ll find this :

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

urlpatterns = [
```

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

It’s already explaining what you should do. Import the include from django.urls and add the path of your app down there like this :

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

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

-I wrote **2 empty quotaions : ‘ ‘** because I want my app path pages to be the **default** when I open the server, but if I didn’t want it to be the default, I’ve would written smth like : **‘/pages’** instead.

-Also make sure to put on a **comma ‘,’** after every line

Fifth :

-Now go to your views.py in your app

-To create a function there and let it be displayed on the server, you must send a request to the server, and the server replies with a response.

-To do so,

1. import HttpResponse from django.urls
2. create your index function that’ll send a request

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

# Create your views here.

def index(request):
    return HttpResponse("Hello from HttpResponse")
```

3. import that function in your app's urls.py by :

- importing it from views
- defining it in the urlpattern ..
- it's syntax is :

urlpatterns = [

path('name of page after slash' , path to function , func name),

]

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


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

]
```

You do so with every page/function you create …

for ex :

-in views.py :

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

```
# Create your views here.

def index(request):
    return HttpResponse("Hello from HttpResponse")


def about(request):
    return HttpResponse(" About Page ")
```

in urls.py :

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


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

Diff between project urls.py and app urls.py

-in my **project urls.py,** if I write the path as **‘pages’** :

path('pages' , include('pages.urls'))

-this means that my **whole project domain** is at : **127.0.0.1:8000/pages** ,

but if I go to **127.0.0.1:8000 only** **without the /pages** , it’ll give me an **error** bc it’s not the default domain.

-and now in my **app urls.py**, if I write the path as **‘index’** :

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

-this means I can find it at : **127.0.0.1/pages/index**

-but since I chose to make my **project urls.py** domain at **empty quotations ‘ ‘** ,

it means my **whole project domain** is found at : **127.0.0.1:8000**

-and since I chose my **index in my app urls.py** to be **empty quotations as well ‘ ‘ ,**

it means I can find it in the **same domain** at : **127.0.0.1:8000/**

Database Model Using PostgreSQL

Created	@October 9, 2023 9:42 AM
Made By	Mariam Tamer

to change the running port from 8000 to any number —>

python manage.py runserver 9000

Template Inheritance

-Components that become **repeated** in the whole page like **Navbar and Footer** are put **only once** in a **base.html file** so you don't have to write them in every page you make ..

-add a new folder called **Layouts** that'll contain your **base.html**

-Your new source will be : **app/templates/appName/Layouts/base.html**

Steps To Create A Database Model

-which dbms ?

-set it up in settings.py

-make sure it’s installed and db is connected

-install postgresql driver

to install postgresql driver :

-make sure your venv is activated

-then type : **pip install psycopg2**

Note

1. **Install `psycopg2` (for Python 3.x):**

   If you are using Python 3.x and Django, you should install `psycopg2` using `pip3` (the Python 3 version of `pip`):

   ```
   pip3 install psycopg2-binary
   ```

   `psycopg2-binary` is a standalone package that includes `psycopg2` and is often easier to install.

2. **Install `psycopg` (for Python 2.x):**

   If you are using Python 2.x (which is not recommended as it has reached its end of life), you can install `psycopg`:

   ```
   pip install psycopg
   ```

type : **pip freeze** to see what packages are installed

---

next : configure confidentiality :

username, password, dbname, portname, hostname

create user on postgresql:

create user mariam with password ‘000’;

note : you don’t use double quotations “ “ in postgresql unless in 2 cases : alias - arrays

to give user permission

alter user mariam superuser;

to give user permission to create database

create database itii owner mariam;

grant all privileges on database itii to mariam;

OR .. But the first one is the one that worked with me

alter user mariam createdb;

to display list of roles (aka list all users)

\du

to list all databaases

\l

If you exit and want to enter to this user again

psql -U username -d database_name

EX : psql -U mariam -d iti

If you wanna enter to the sudo postgres user

sudo -u postgres psql

create database

create database project;

---

then add credits in settings.py

```
DATABASES = {
"default": {
"ENGINE": "django.db.backends.postgresql",
"NAME": "project",
"USER": "mariam",
"PASSWORD": "0000",
"HOST": "localhost",
"PORT": "5432",
}
}
```

---

connect to database

\c iti

-Django connects to DB using ORM : Object Relational Manager.

-Any table in DB is seen as a CLASS.

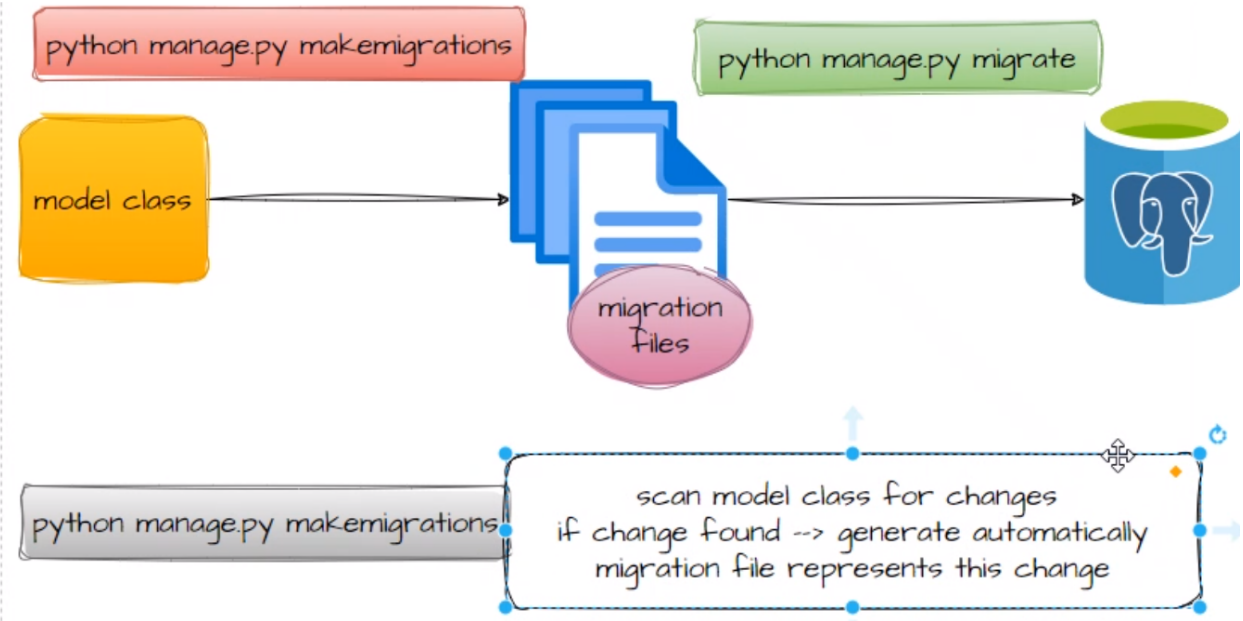
-Any row in DB is seen as an OBJECT.

-bc **class** : **field type + method(Type) type**

-**object** : **represents record**

How to transfer from classes&objects to PostgreSQL in Django so they’re applied in DB?

-Using Migration Files



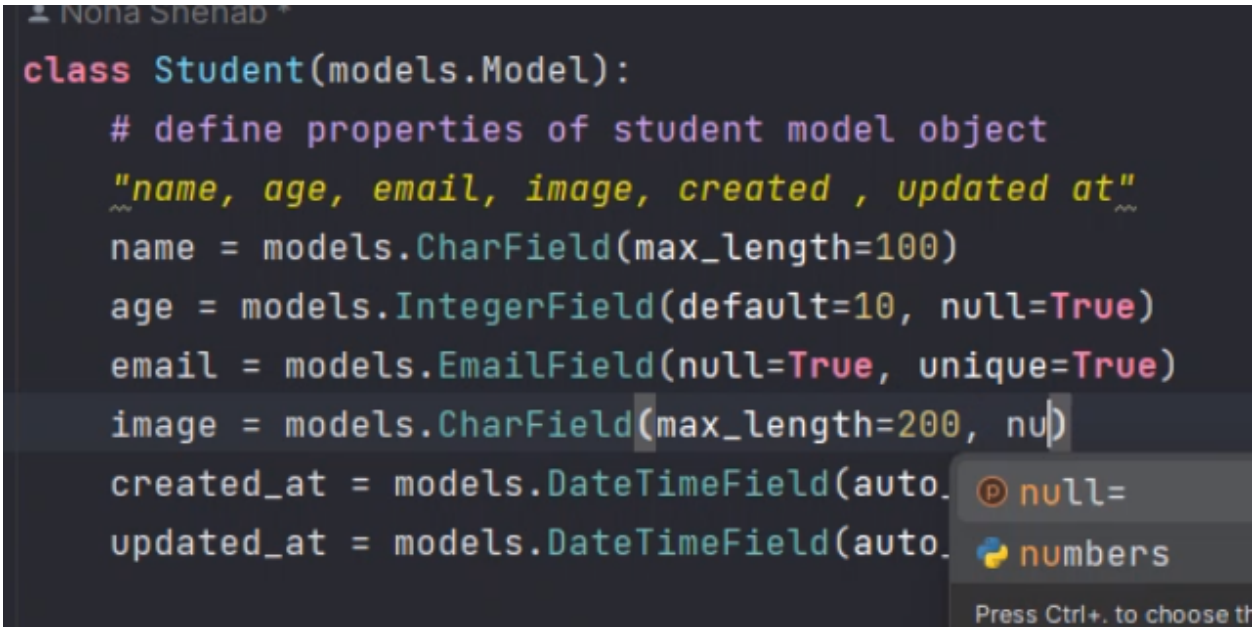
-so to migrate your files, you type :

python3 manage.py makemigrations

python3 manage.py migrate

Creating my own model

-Give attributes to your student table



-Here, **EmailField** is avaialable.

-There's an **auto_now_add=True** property for dates.

-After every edit in your database, type the migration 2 commands

-You can check for your databases in PostgreSQL to see what's added

-In the newly generated migration files, you can find this:

```
    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Student',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
                ('name', models.CharField(max_length=100)),
                ('age', models.IntegerField(default=10)),
                ('email', models.EmailField(max_length=254)),
                ('image', models.CharField(max_length=200)),
                ('created_at', models.DateTimeField(auto_now_add=True)),
                ('updated_at', models.DateTimeField(auto_now=True)),
            ],
        ),
    ]
```

to list description of a your models table in javascipt ,(its cols and data types)

\d appName_tableName

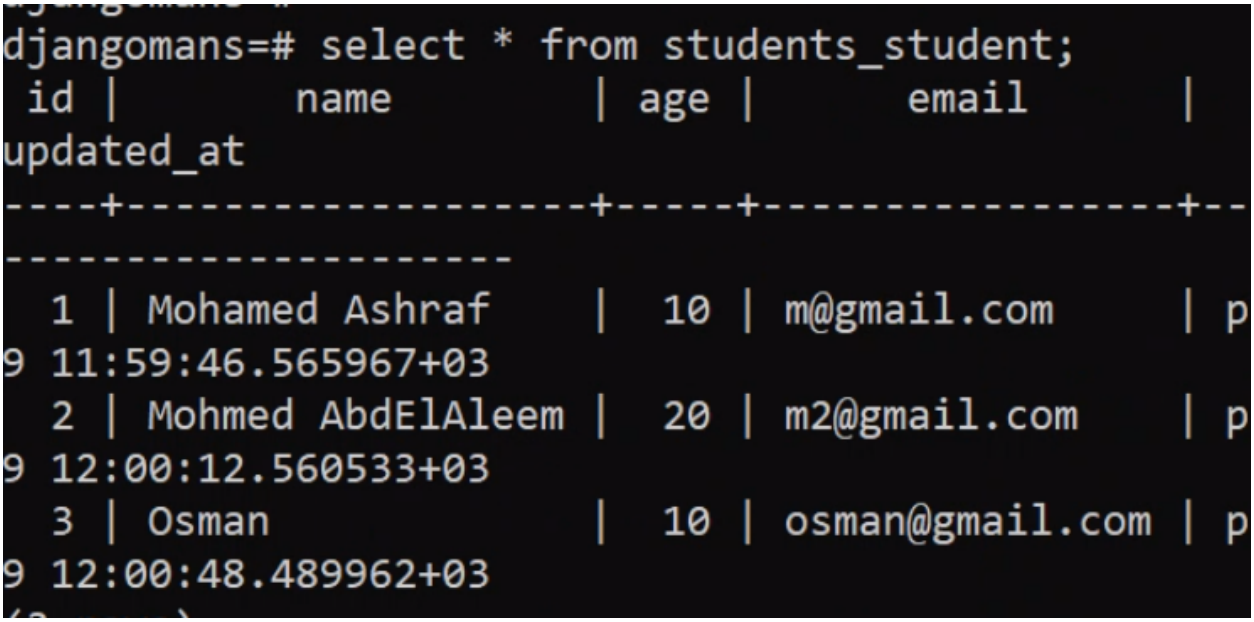
ex : \d project_student

```
djangomans=# \d students_student
                                Table "public.students_student"
   Column   |           Type           | Collation | Nullable |              Default
------------+--------------------------+-----------+----------+-----------------------------------
 id         | bigint                   |           | not null | generated by default as identity
 name       | character varying(100)   |           | not null |
 age        | integer                  |           | not null |
 email      | character varying(254)   |           | not null |
 image      | character varying(200)   |           | not null |
 created_at | timestamp with time zone |           | not null |
 updated_at | timestamp with time zone |           | not null |
Indexes:
    "students_student_pkey" PRIMARY KEY, btree (id)
```

-Any table you generate using **any Framework including models here**, your cols are **NOT Null by default.**

-So if you want a **column to be empty/or not** depending on the user's desire, you add **(Null = True)** property to the column.

To display the table, don't forget to write app_table when selecting :



How Does Interaction Happen?

-Since **model** is the one that **talks to DB,** Django sends the model object so it becomes **saved** in the DB (PostgreSQL here), or Django **requests** an **object** or **set of objects** from the DB

Register To Admin Panel

-In your VSC terminal, type :

python3 manage.py create superuser

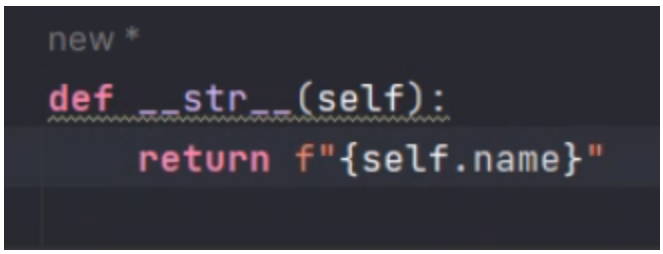
Add the model you created to Admin Panel

```
from students.models import Student

admin.site.register(Student)
```

To **stringify** your student table objects so they become **normal names** on your admin panel **instead of student.object1 , … —>**

def __str__(self):

return f”{self.name}”



To make your terminal colored :

python manage.py <command> --color <option>

QuerySets

To add new data to your table from a shell in your terminal **for testing** instead of from Admin Panel Site

1) python3 manage.py shell

then write your code like the following :

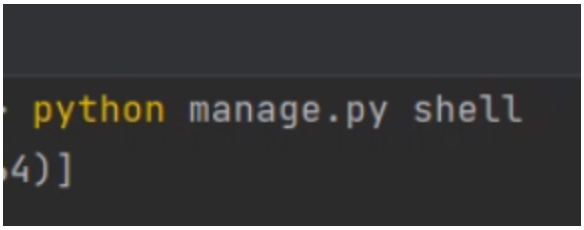
2) import your model table —> from app_name.models import TableName

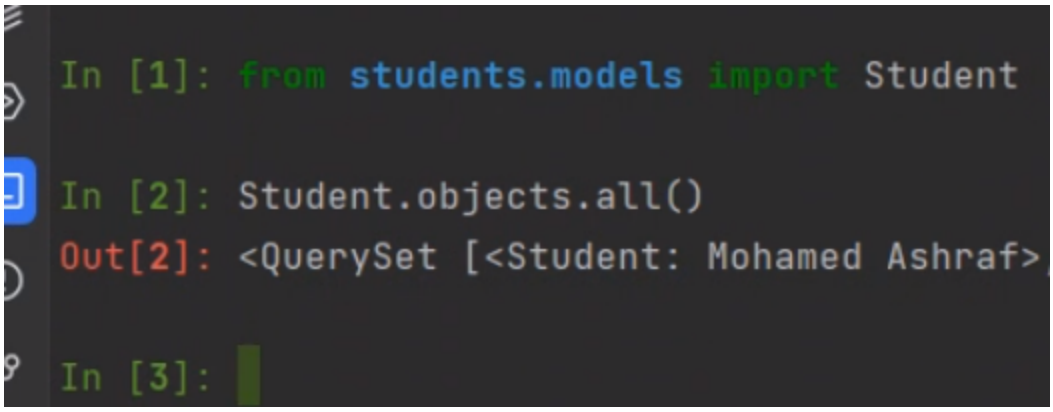
3) to access all data in table —> Student.objects.all()

4) create an object from your table so you could add data to it : s = Student()

5) then insert the data you want :

6) then s.save() to save all this and check it on your server





```
In [3]: s = Student()

In [4]: s.name='yahia2'

In [5]: s.email='yahia2@gmail.com'

In [6]: s.age = 23

In [7]: s.image = 'pic3.png'

In [8]: s
Out[8]: <Student: yahia2>

In [9]: s.age = 23
```

```
In [10]: s.save()
```

-To get a specific object :

Student.objects.filter(id=1)

```
In [9]: Student.objects.filter(id=1)
Out[9]: <QuerySet [<Student: Mohamed Ashraf>]>
```

-To get specific objects

for ex : get students id > 2

Student.objects.filter(id__gt=2)

id underscore underscore gt

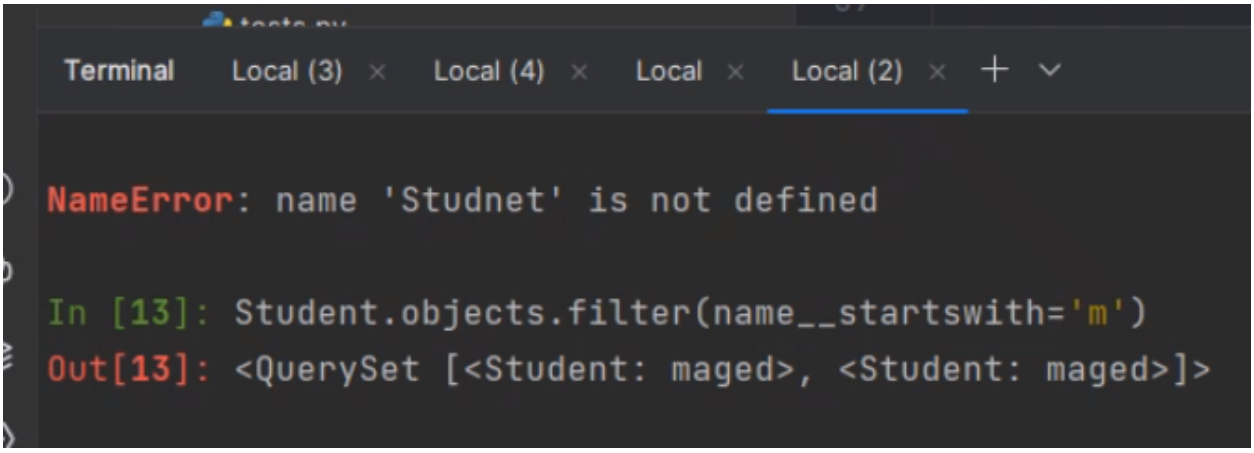
```
In [11]: Student.objects.filter(id__gt=2)
    ...:
Out[11]: <QuerySet [<Student: Osman>, <Student: Yahia>, <Student: yahia2>, <Student: maged>, <Student: maged>]>
```

-to do this :

select * from students where name like ‘m%’;

Student.objects.filter(name__startswith=’m’)

-In shell :



To use them in views.py :

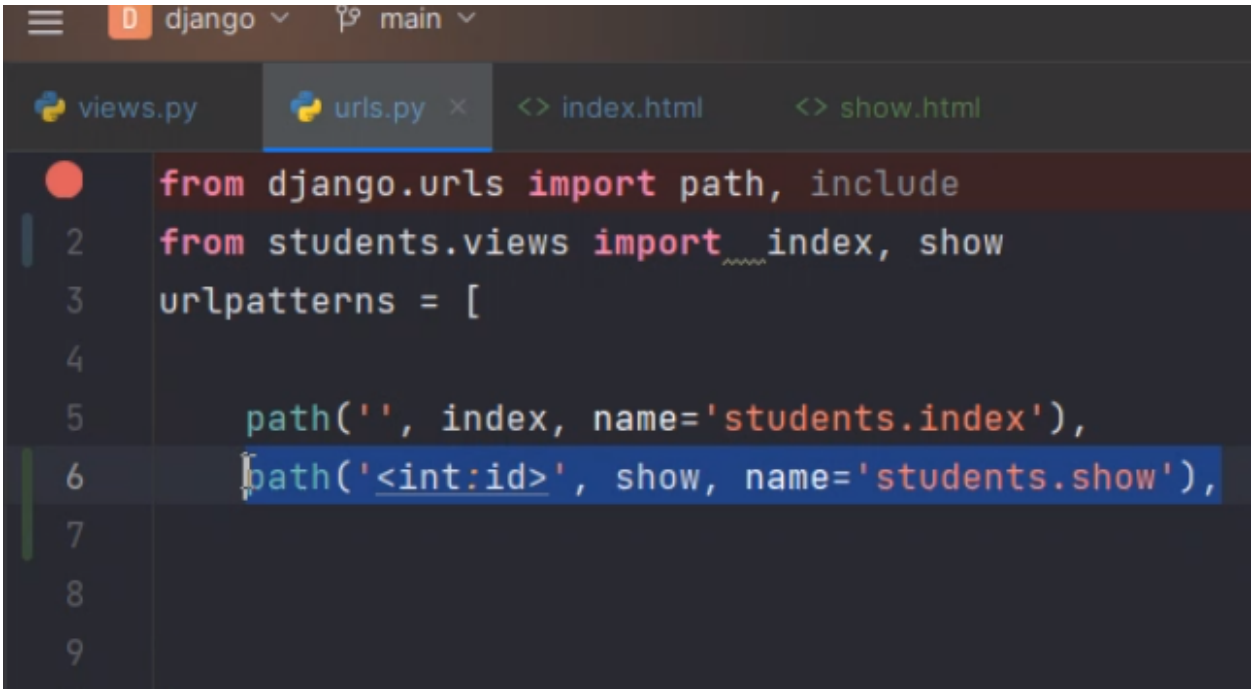
from .models import Student t2rebn

```
14  def show(request, id):
15      student = Student.objects.get(id=id)
16      return render(request, template_name: 'students/show.html', context={"student":student})
17
18
    new *
19  def delete(request, id ):
20      student = Student.objects.get(id=id)
21      student.delete()
```

-The (id=id) part means like : when id=’4’

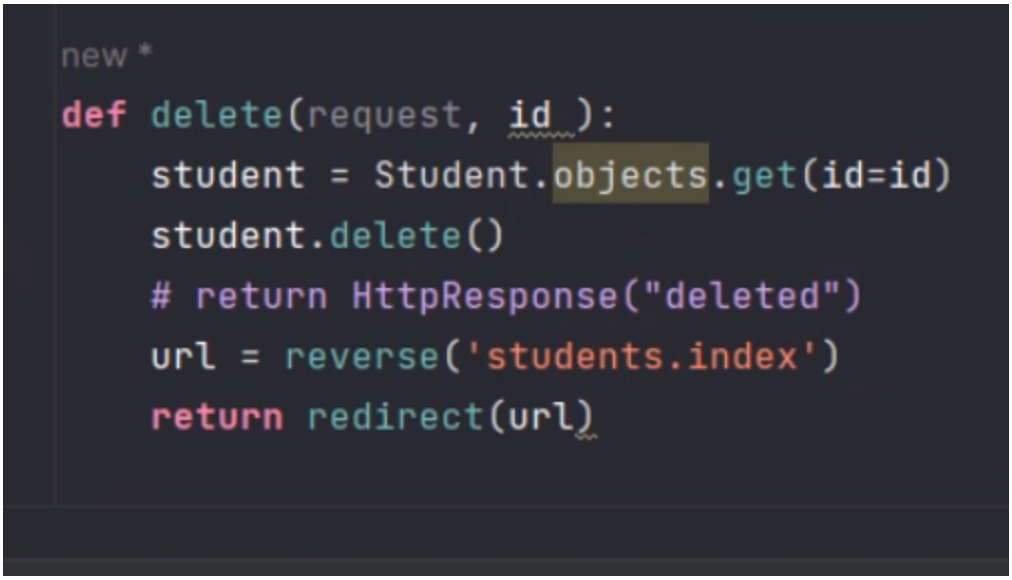
```
18
19
    new *
20  def delete(request, id ):
21      student = Student.objects.get(id=id)
22      student.delete()
23      # return HttpResponse("deleted")
24      return redirect('')
```

And in urls.py :



To redirect to a url back :

use the reverse function



-An advanced (more detailed) example of a DB model :

```
class Product(models.Model):

  x = [
      ('phones', 'phones'),
      ('computers', 'computers'),
      ('gym products' , 'gym products'),
  ]

  name = models.CharField( max_length=30, default='name', verbose_name='product name' )
  content = models.TextField(null=True, blank=True)
  price = models.DecimalField( max_digits=5, decimal_places=2, default=6.3)
  image = models.ImageField(upload_to='photos/%y/%m/%d', default='photos/20/12/2')
  active = models.BooleanField(default=True)
  category = models.CharField(max_length=30,null=True, blank=True, choices=x)

  def __str__(self):
    return self.name   #so they could be called iphone6,.. not product.object

class Meta:
#     verbose_name = ('hamada')
    ordering = ['-price']
```

-**Verbose** i**s the name of the column that appears on server.**

-In **DecimalFIeld**, **max_digit=5** is the **num of unit digits before the dot …..**
decimal_places =3 is the **num of decimal digits after the dot EX : 12200.111**

Database Form

Created	@October 11, 2023 10:41 AM
Made By	Mariam Tamer

Steps to add new objects into your DB table using form:

1. **Make a button and make its href to send you to the form page**
2. **In your html file, make a bootstrap form**
    - **make it's method="post"**
    - **the type="" depending on your cols**
    - **add the csrf token**
    - **add name attributes in your html inputs so the data can be posted on the network Payload**
3. **Add the view function for the html page**
    - **check if the request is post or get**
    - **get data from the request body**
    - **add data to your DB**
4. **Add the view function in urls.py of your app**
5. **Add its url in your html form href**

Let's discuss them step by step :

To add students objects into DB using form

1. **Make a button and make its href to send you to the form page**
2. **Make a bootstrap form and edit the type="" depending on your cols**

```
<div class="mb-3">
  <label class="form-label">Name </label>
  <input type="text" class="form-control"  aria-descr
</div>
<div class="mb-3">
  <label class="form-label">Email </label>
  <input type="email" class="form-control"  aria-descr
</div>
<div class="mb-3">
  <label class="form-label">Image </label>
  <input type="text" class="form-control"  aria-descr
</div>
  <div class="mb-3">
  <label class="form-label">Image </label>
  <input type="text" class="form-control"  aria-descr
</div>
<button type="submit" class="btn btn-primary">Submit</
```

3. **Add the view function for the html page**

```
new *
def create(request):
    return render(request, template_name: 'students/create.html')
```

4. **add it’s urls.py in pages**

```
rlpatterns = [

    path('', index, name='students.index'),
    path('<int:id>', show, name='students.show'),
    path('<int:id>/delete', delete, name='students.delete'),
    path('create', create, name='students.create')
```

5. **give the href this url in your html page**

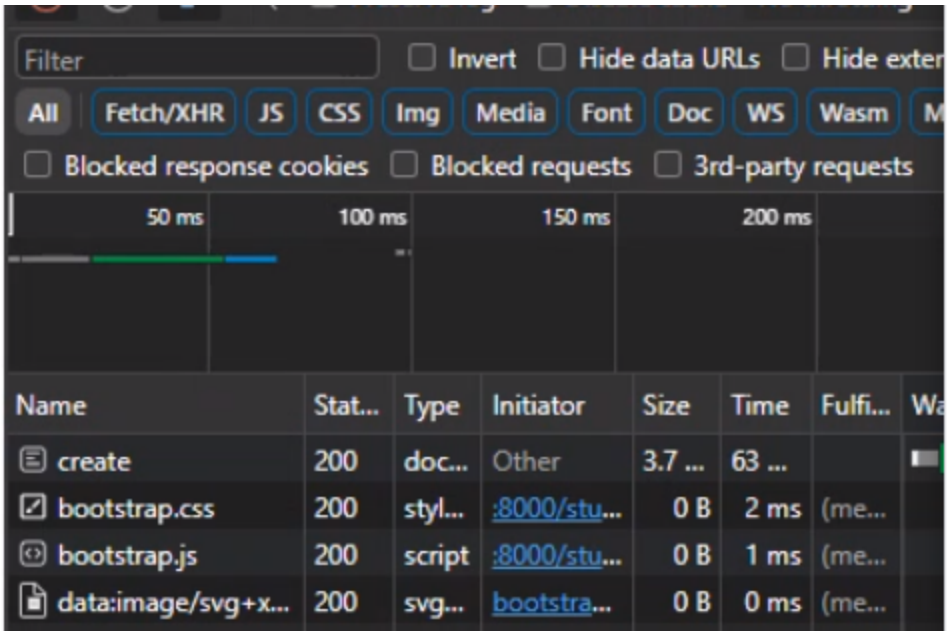
```
{% block title %}Students Index {% endblock %}
{% block main %}
    <h1> Welcome to students index </h1>
    <a href="{% url 'students.create' %}" class="btn btn-dark">
    <table class="table">
            <tr> <th> ID</th>  <th> Name</th> <th>Image </th>
                <th> Show </th> <th> Delete</th>
            </tr>
```

-Now if you go to your form and press Submit then open your network to check your function status, you’ll find it’s a get method and your data is sent in the **same page** .. so you want your method to be post so your data is posted on **another page** ..

method=’get’ —> data posted in the **same page**

method = ‘post’ —> data posted to **another page**

-You can find all this in your **inspect—network ..** (the create function you created in your views/whatever its name is)



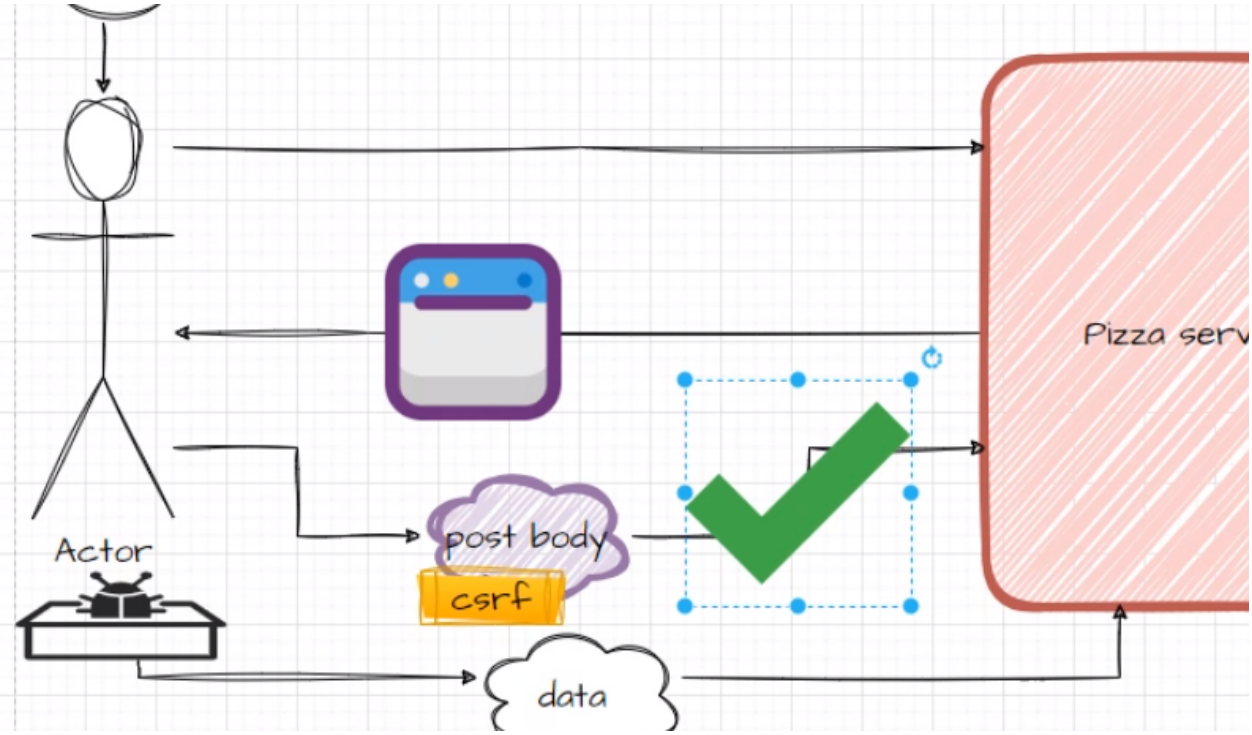
-After changing your method to post and pressing submit, you'll get an error called : csrf

csrf error —> *cross site forgery request*

(related to certificate and authentication)

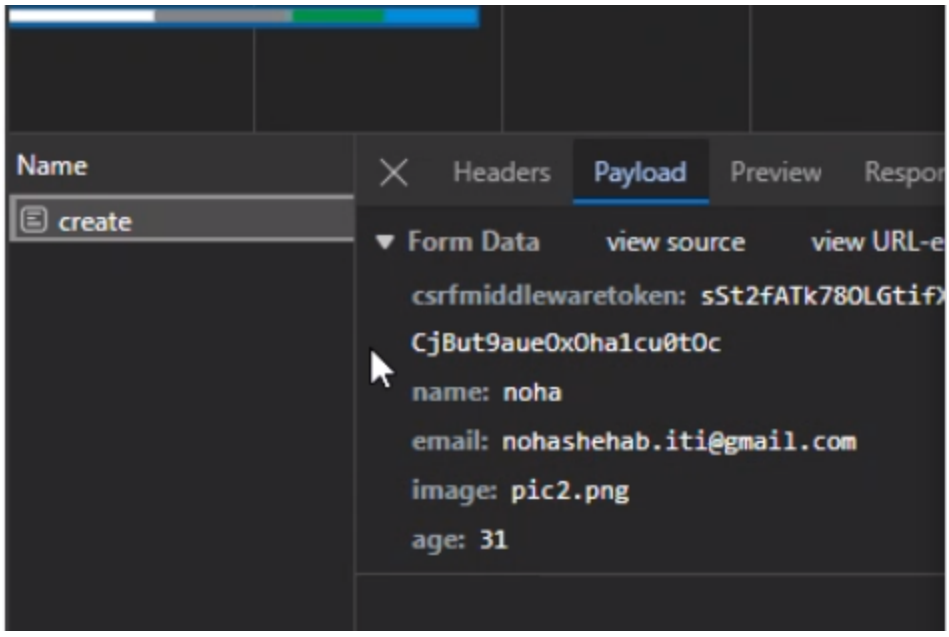
-When you post a form to server, you connect to a network. This network downloads mulware on your system. (A mulware is like a bad software or a software with virus for EX). This mulware takes the data in your post body and sends data to the server from their own data without checking if any changes happened to the data in the process which is not secure, so you get that csrf certification error.

-Only those who have the **crsf token** will pass their post body data and go to the server.

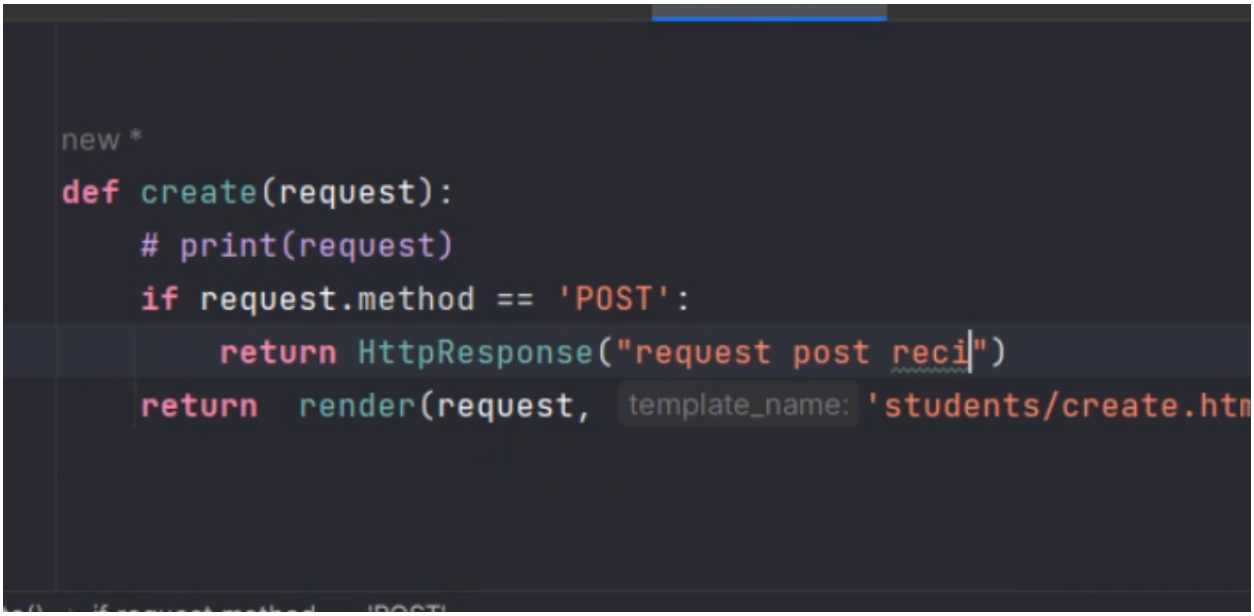


6. **So add the csrf token to your form like this , and your post method will be sent to server.**

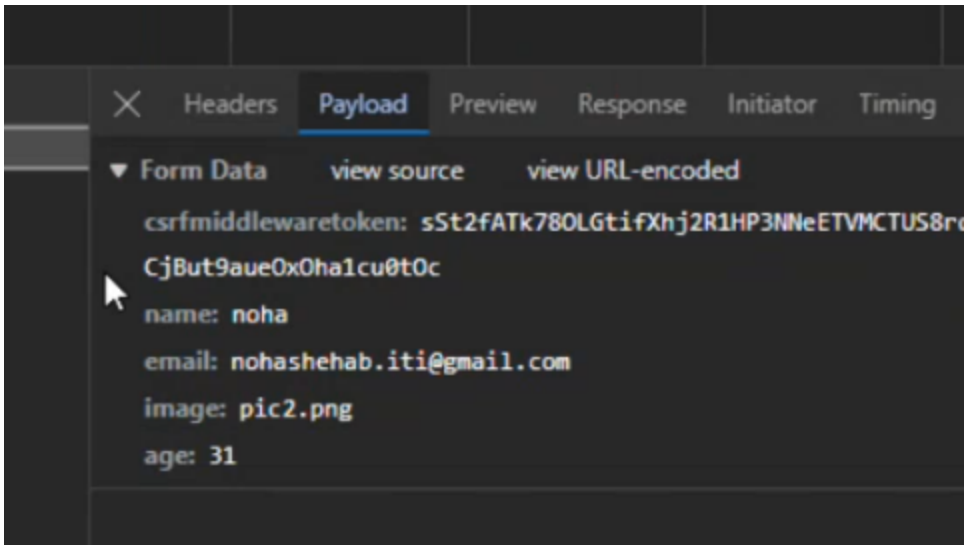
```
{% block main %}
    <h1> Add new Student  </h1>
    <form method="post" >
        {% csrf_to%}
  <div class="mb-3">
    <label class="form-label">Name </label>
    <input type="text" class="form-control"
  </div>
 <div class="mb-3">
```



7. **To know if your method=get or post**



8. **add name attributes in your html inputs so the data can be posted on the network Payload**



9. **Now, to post them to DB, get data from request body, then add it to DB**

- **get it from request body**

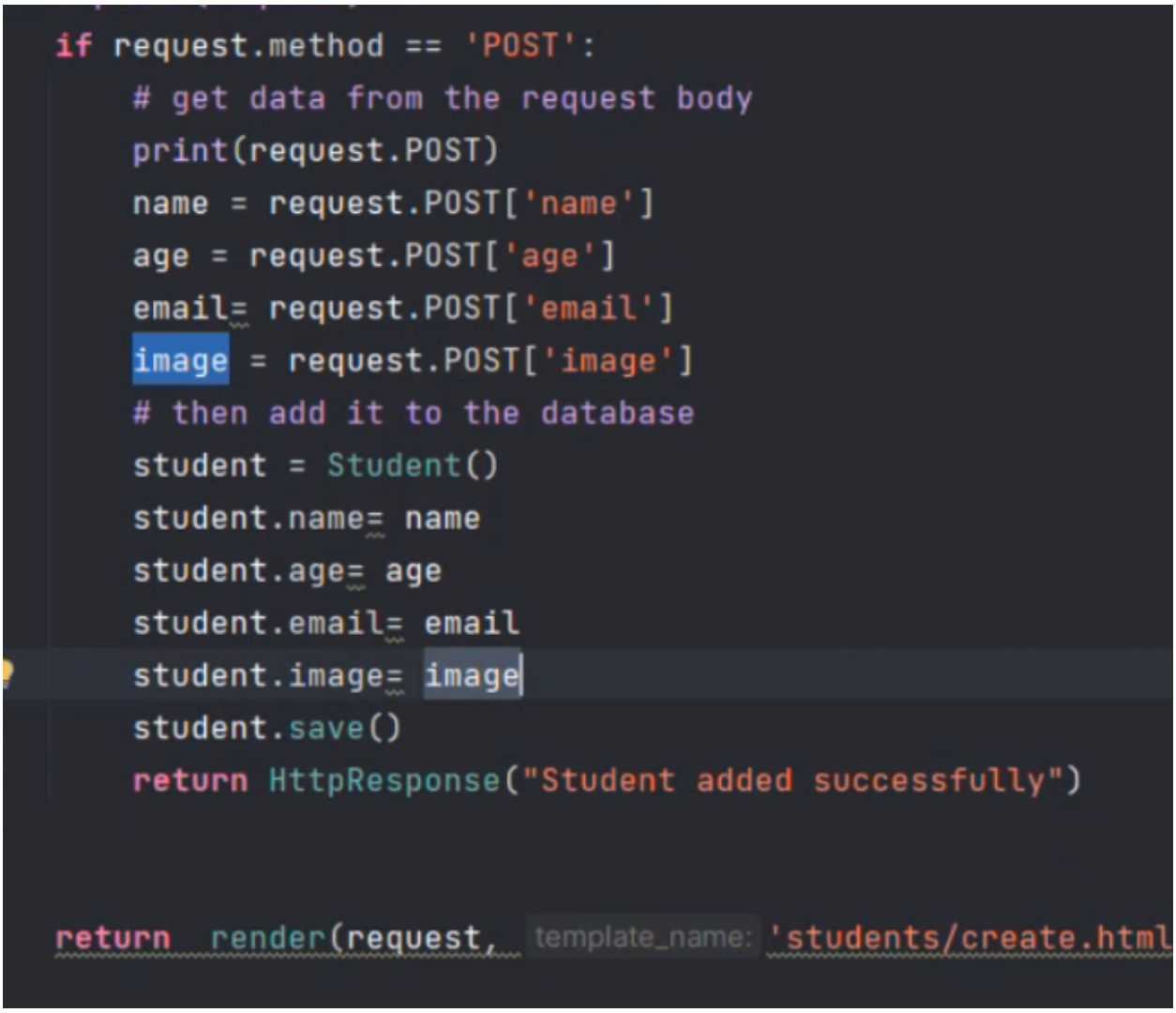
(**request.POST**) .. POST is capitalized

- **add it to DB**

```
email= request.POST['email']
image = request.POST['image']

student = Student()
student.name= name
student.age= age
student.email= email
student
# then add it to the database
```

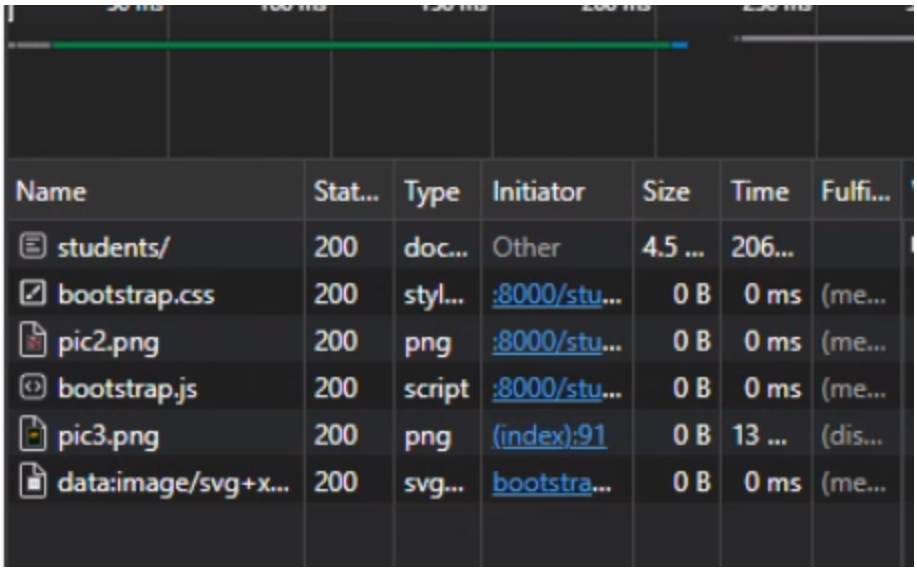
then student.save()

```
student.name= name
student.age= age
student.email= email
student.image= image

student.save()


return HttpResponse("Student added successfully
```



Now if u submit data :

a new student is added successfully to your DB and appears on your site page



Name	Stat...	Type	Initiator	Size	Time	Fulfi...
students/	200	doc...	Other	4.5 ...	206...	
bootstrap.css	200	styl...	:8000/stu...	0 B	0 ms	(me...
pic2.png	200	png	:8000/stu...	0 B	0 ms	(me...
bootstrap.js	200	script	:8000/stu...	0 B	0 ms	(me...
pic3.png	200	png	(index):91	0 B	13 ...	(dis...
data:image/svg+x...	200	svg...	bootstra...	0 B	0 ms	(me...

```
student.email= email
student.image= image
student.save()
# return HttpResponse("Student added successfully")
# url = reverse('students.index')
# return redirect(url)
url = reverse( viewname: 'students.show', args=[student.id])
return redirect(url)
```

return redirect 3la el track

after break :

form validation

-Django made smth called **“Django Forms”**

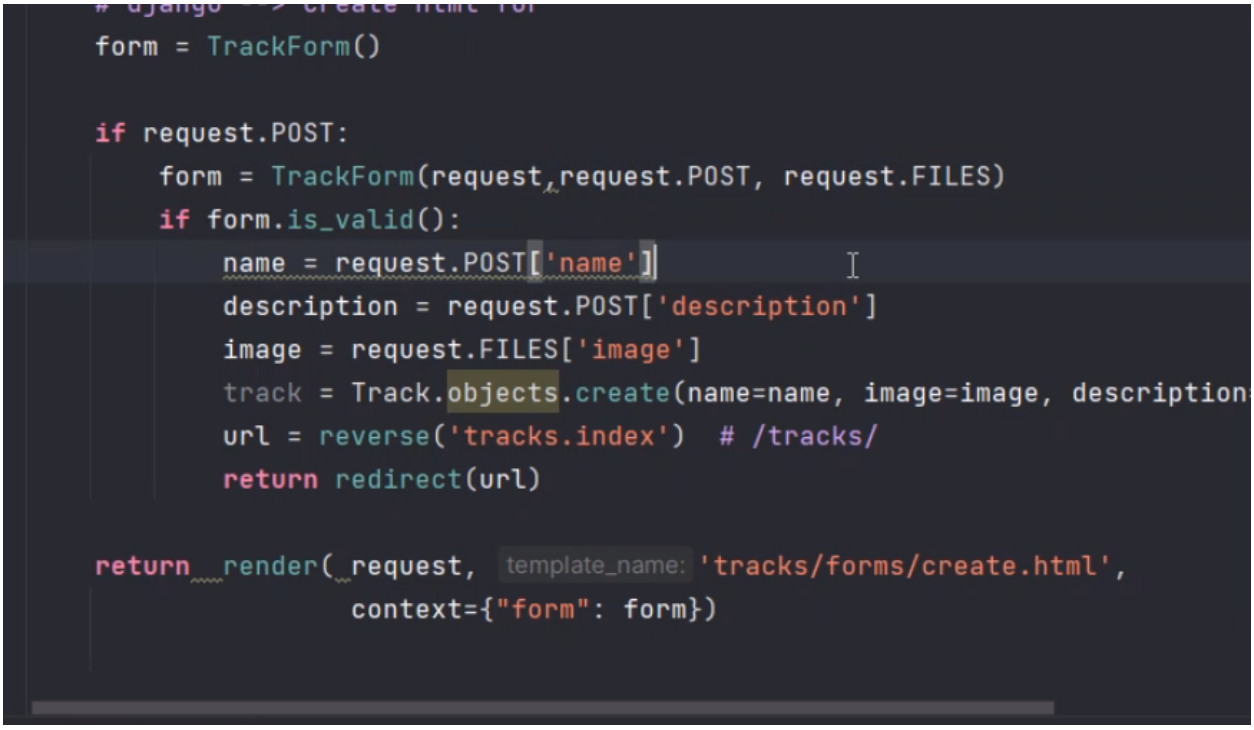
-inside your tapp, make : **forms.py**

```
from django import forms

class TrackForm(forms.Form):
  name=forms.charField()
  rest like model ....
```

```python

from django import forms


2 usages  new *
class TrackForm(forms.Form):
    name = forms.CharField()
    description = forms.CharField()
    image = forms.ImageField()


```

in views :

```python
2 usages  new *
def createViaForm(request):
    # django --> create html form

    form = TrackForm()
    return render( request, template_name: 'tracks/forms/create.html',
                   context={"form": form})
```



Lab 3 :

-Using forms to : create product - edit product - modify image in product model to be image field (dont put url , instead : upload image)

-dont work in model forms

-modify products model to include category

-category has a model - index - show (1 to many)

-when u click show category, its page will display products in this category

-category has the following features : name - description - image

Registeration Form

Created	@October 14, 2023 11:34 AM

-The quick steps to create a registeration form:

1) Views

2) Template

3) Forms.py

4) Install Crispy Forms

but .. we'll go through them in details to make applying them easier, so we'll divide them to 6 smaller steps like this :

6 Detailed Steps to create a registeration form :

-After creating your new app : User ,

1) In Views, import the UserModelForm from django

2) Make your HTML Template

3) (Optional) : Add url of User app in project's urls.py

4) Update your Views to get and save data

4) Create your desired form fields in forms.py

5) Apply the changes to your views.py

6) Install Crispy forms for styling

and in the end we'll summarize them again as the quick 4 steps ..

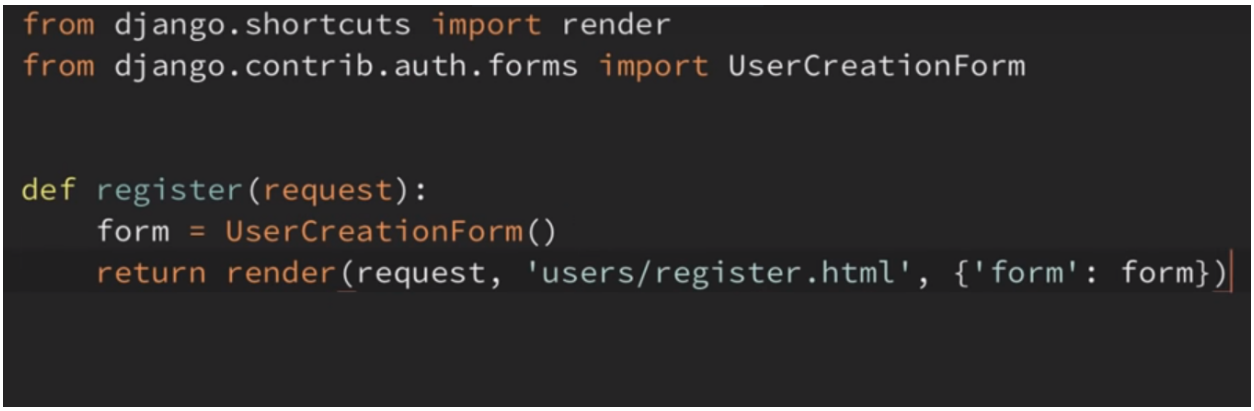
---

1) views.py :

- **import UserCreationForm for the built in django forms in the package : django.contrib.auth.form**

  this package makes you a built in form input so you don't have to manually create it .. all you have to do is just render it in your html templates, and just create the forms method tags & csrf token & the buttons you like (submit, login,…) as well as give them the styles

- **create an instance from that form**
- **render it in your html template using a context name "form" from that form**



```
from django.shortcuts import render
from django.contrib.auth.forms import UserCreationForm


def register(request):
    form = UserCreationForm()
    return render(request, 'users/register.html', {'form': form})
```

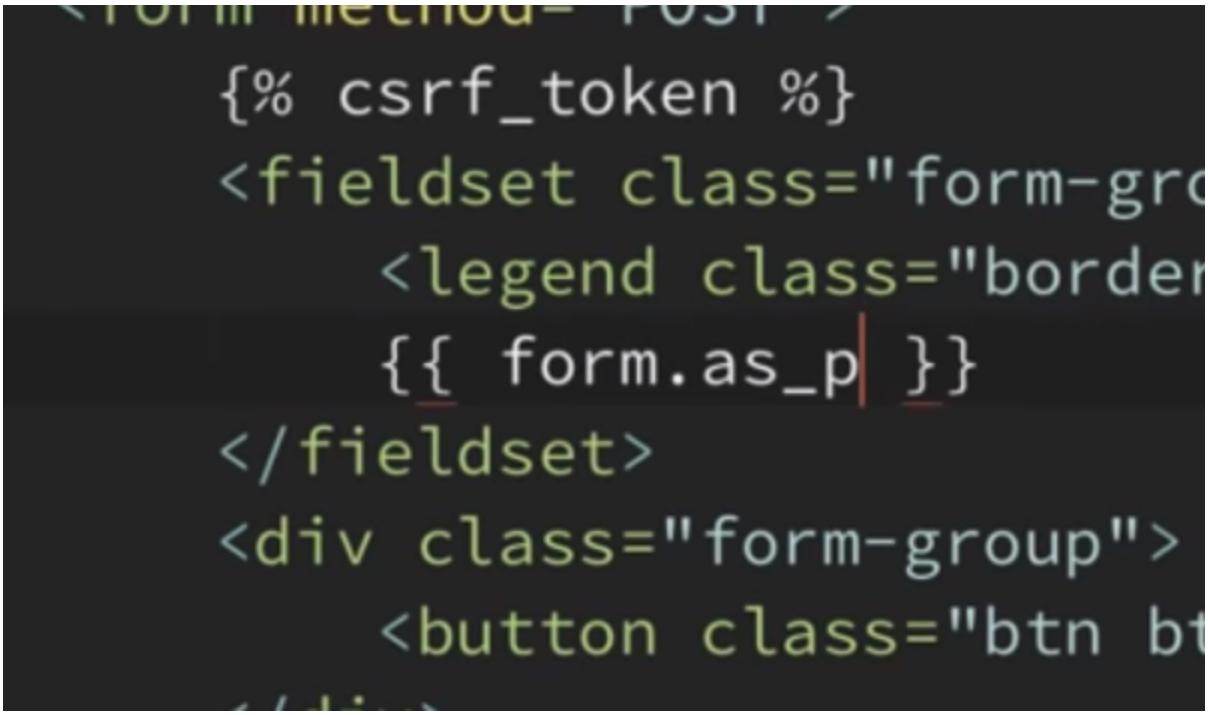
2) Your register.html template :

-here, you write the form method tags along with the csrf token, and you just render the built in django forms like this : {{form}} , and maybe give them bootstrap styles

-In case you forgot, the csrf token is a certificate given for the security of your POST method data so no mulware manipulates your post body data and your data goes safely to the server

```
{% extends "blog/base.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Join Today</legend>
                {{ form }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info" type="submit">Sign Up</
            </div>
        </form>
        <div class="border-top pt-3">
            <small class="text-muted">
                Already Have An Account? <a class="ml-2" href="#">Sign In</a
            </small>
        </div>
    </div>
{% endblock content %}
```

for better styling, you can do this : {{ form.as_p }} —> this “ as_p “ will display your form inputs as paragraph elements with spaces between each input



3) (Optional) : Add url of User app in project's urls.py

An important note :

Instead of adding your app url in the app urls.py then connecting it to the whole project in the project's urls.py,, you can do this only once by adding the URL of the app only once in the urls.py of the whole project by importing views there like this :

```python
from django.contrib import admin
from django.urls import path, include
from users import views as user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name='register'),
    path('', include('blog.urls')),
]
```

Update your views.py to get and save data

Now, if you try to write your info in the form on the server and press submit, nothing will happen and you'll be redirected to the same page because you haven't wrote what to do with that information in the form,

so now in the views again :

- **you'll first tell the server to check whether the method is post or get**
- **if it's post , then …. if it's get then leave it as it is redirecting you to the same page**
- **if it's post, then**

  … make an instance of the form who's request method is post (form = …… (request.post))

  …. validate that your correct data are in the form

  ….if validation is true, then : - save the form using formsave() …. -get the username … -display a flash msg (msg that appears only once .. you import it from django.contrib and there 4 types : messages.success / messages.info / messages.warning / warning.error) …. don't forget to apply it in your base.html page so it is displyed in any other page

…. then redirect the user to a new page when logging in (home page for ex)

```python
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm
from django.contrib import messages


def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username}!')
            return redirect('blog-home')
    else:
        form = UserCreationForm()
    return render(request, 'users/register.html', {'form': form})
```

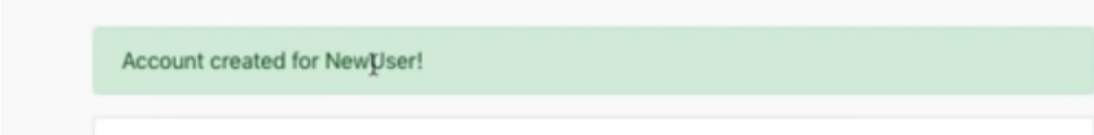
AND DON'T FORGET THE FORM.SAVE()

```python
from django.contrib import messages


def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username}!')
            return redirect('blog-home')
    else:
        form = UserCreationForm()
    return render(request, 'users/register.html', {'form': form})
```

and display the message.success here in the base.html

```
    </nav>
  </header>
  <main role="main" class="container">
    <div class="row">
      <div class="col-md-8">
        {% if messages %}
          {% for message in messages %}
            <div class="alert alert-{{ message.tags }}">
              {{ message }}
            </div>
          {% endfor %}
        {% endif %}
        {% block content %}{% endblock %}
```

now, if you try signing up in the form, you'll get this message.success :



4) Create your desired form fields in forms.py

-Now what if you wanna add more fields to your forms ??

-Here in the template you render the whole django form {{ form }} instead of specific fields

-You'll have to create a new file in your User app that's called : forms.py

-There, you'll inherit from your User Model that django created from you , and you'll be able to add all the fields you want .. for EX : email field here ..

-class Meta : is a class of data about the data (aka forms) ..this is where you write the name of the model you inherit from , and the name of the fields you wanna display

In forms.py :

```python
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm


class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
```

5) Apply the changes to your views.py

-NOW since you inherited from the User model, apply these changes (replace UserCreationForm with UserRegisterForm) to your Views.py :

In views.py :

```python
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import UserRegisterForm


def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username}!')
            return redirect('blog-home')
    else:
        form = UserRegisterForm()
    return render(request, 'users/register.html', {'form': form})
```

6) Intsall Crispy Forms

This gives your django forms all the great styles without u manually writing them on your own.

1) pip install django-crispy-forms

2) Add it in the settings installed apps :

in your settings .py :

```
INSTALLED_APPS = [
    'blog.apps.BlogConfig',
    'users.apps.UsersConfig',
    'crispy_forms',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

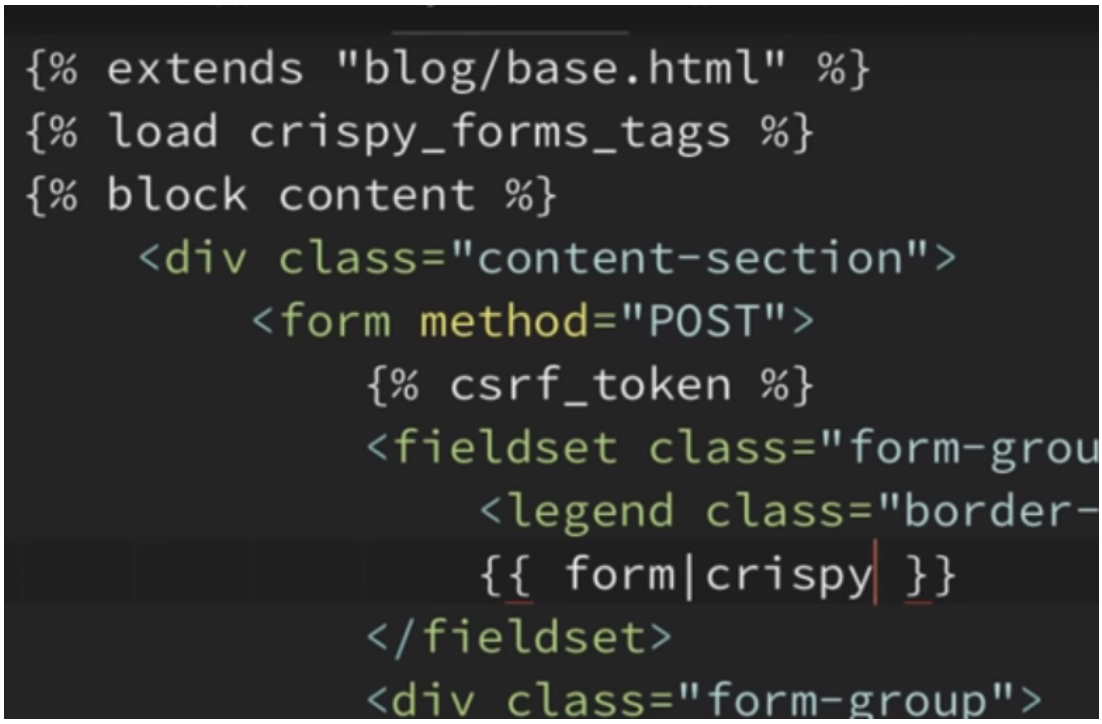
3) **set the styling template that crispy forms will use : for ex : bootstrap 5 or 4**

```
STATIC_URL = '/static/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

4) load the crispy forms template in your register.html **{% load crispy_forms_tags %}**

& replace it with the old **.as_p** **method so it becoms** **{{ form|crispy }}** **instead**

in your register.html :



```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class="content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-grou
                <legend class="border-
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
```

NOTE : If this didn’t work with you, you can try the following because I had a problem with it and this fixed it :

33

I also ran to this problem but crispy-form is already supporting boostrap 5. In their git was instructed as so

```
$ pip install django-crispy-forms

$ pip install crispy-bootstrap5
```

And in settings.py

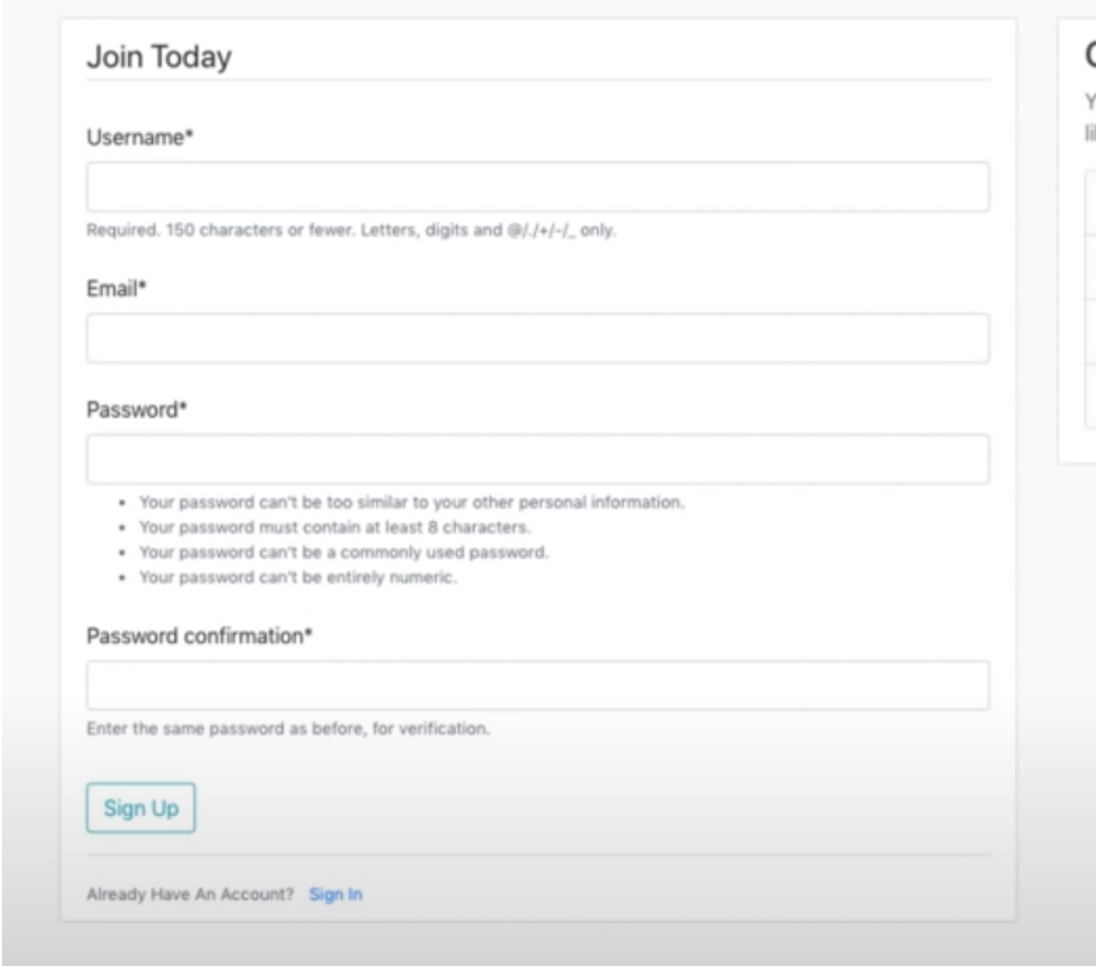
```
INSTALLED_APPS = [
 ...,
'crispy_forms',
'crispy_bootstrap5',  # Forgetting this was probably your error
 ]
```

And then at the bottom of the page of settings.py

```
CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap5"
CRISPY_TEMPLATE_PACK = "bootstrap5"
```

This worked for me solving the TemplateDoesNotExist error. No need to downgrade bootstrap4

and now …. this is what your form will look like :



SO … Summarizing them into 4 quick steps :

1) Forms.py

```python
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm


class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
```

2) Views.py

```python
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import UserRegisterForm


def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username}!')
            return redirect('blog-home')
    else:
        form = UserRegisterForm()
    return render(request, 'users/register.html', {'form': form})
```

3) Templates :

first .. register.html :

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class="content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Join Today</legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info" type="submit">Sign Up</
            </div>
        </form>
        <div class="border-top pt-3">
            <small class="text-muted">
                Already Have An Account? <a class="ml-2" href="#">Sign In</a
            </small>
        </div>
    </div>
{% endblock content %}
```

second .. base.html :

```
    </nav>
  </header>
  <main role="main" class="container">
    <div class="row">
      <div class="col-md-8">
        {% if messages %}
          {% for message in messages %}
            <div class="alert alert-{{ message.tags }}">
              {{ message }}
            </div>
          {% endfor %}
        {% endif %}
        {% block content %}{% endblock %}
```

Don’t forget to add the Project’s urls.py

```python
from django.contrib import admin
from django.urls import path, include
from users import views as user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name='register'),
    path('', include('blog.urls')),
]
```

4) Intsall Crispy Forms

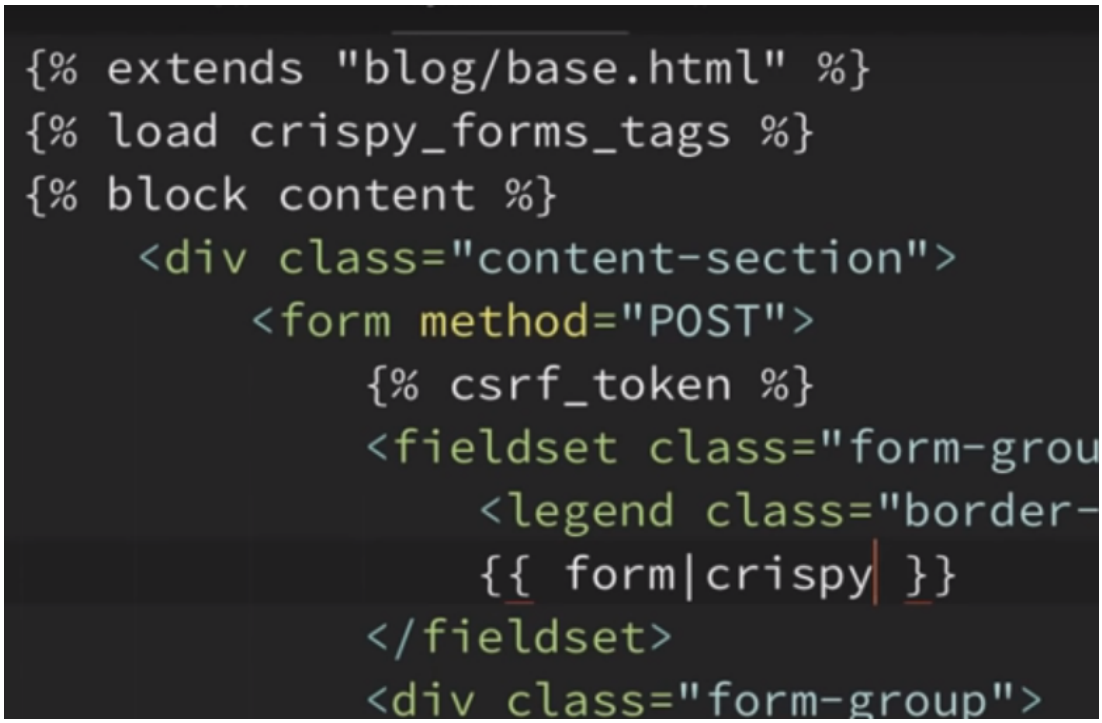
pip django-crispy-forms

add User app & crispy app & crispy template

```
INSTALLED_APPS = [
    'blog.apps.BlogConfig',
    'users.apps.UsersConfig',
    'crispy_forms',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

```
STATIC_URL = '/static/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

Apply crispy forms tags to your templates


{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}

<form method="POST">
{% csrf_token %}
<fieldset class="form-grou
<legend class="border-
{{ form|crispy }}
</fieldset>

Login & Logout System

Created	@October 14, 2023 7:06 PM

Steps to create a Django Login & Logout system :

1) Whole Project/s urls.py

2) login.html & logout.html

3) settings.py

4) Add the profile in your views

5) create the profile.html template to render the user name there

6) Add the profile link in the URL

7) Update Navbar Depending on user authentication status

---

1) Whole Project/s urls.py

-use auth to import views as auth_views

-Notice the Capitalized first letters in LoginView & LogoutView in the paths because they are classes that we're gonnna create

-Don't forget to add .as_view()

```
"""
from django.contrib import admin
from dajngo.contrib.auth import views as auth_views
from django.urls import path, include
from users import views as user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name='register'),
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('', include('blog.urls')),
]
```

and add your template name in the as_view()

2) In your login.html page :

copy paste the register.html code that we did before and just change the button names to Login and Sign up Instead of submit and login

```
class="content-section">
<form method="POST">
    {% csrf_token %}
    <fieldset class="form-group">
        <legend class="border-bottom mb-4">Log In</legend>
        {{ form|crispy }}
    </fieldset>
    <div class="form-group">
        <button class="btn btn-outline-info" type="submit">Login</button>
    </div>
/form>
div class="border-top pt-3">
    <small class="text-muted">
        Need An Account? <a class="ml-2" href="{% url 'register' %}">Sign Up
    </small>
/div>
>
ck content %}
```

Logout template logout.html

```
{% block content %}
    <h2>You have been logged out</h2>
    <div class="border-top pt-3">
        <small class="text-muted">
            <a href="{% url 'login' %}">Log In Again</a>
        </small>
    </div>
{% endblock content %}
```

3) settings.py

write your home page url name here so when you login, your browser directs you into your home page

```
LOGIN_REDIRECT_URL = 'blog-home'
LOGIN_URL = 'login'
```

-you can no change your view so when a user signs up for the first time, they're directed to the login page

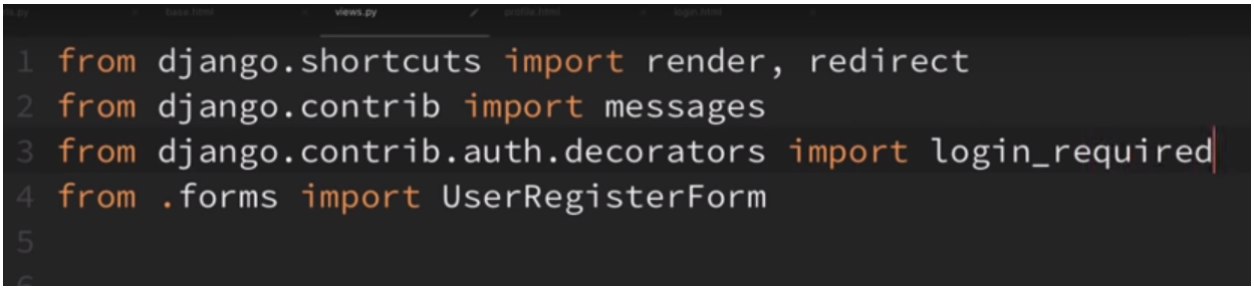
return redirect('login')

```
def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Your account has been created! You a
            return redirect('login')
    else:
        form = UserRegisterForm()
    return render(request, 'users/register.html', {'form': form})
```

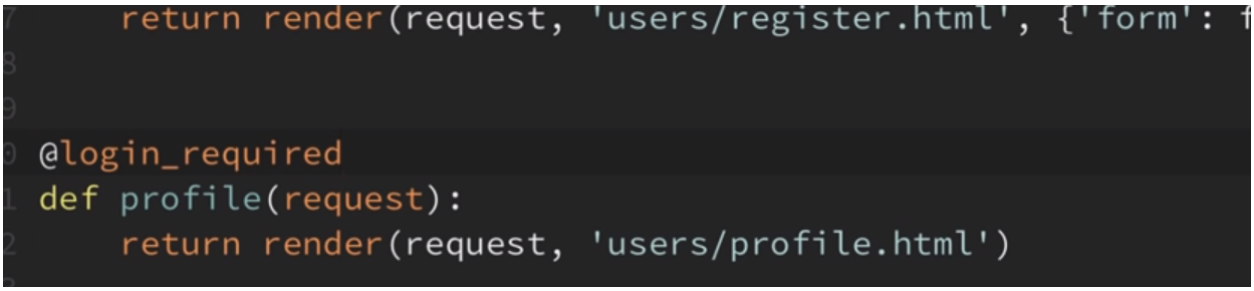
4) Add the profile in your views

- **this profile template is gonna display the user name of the logged on user**
- **import the login_required decorator**

  decorator : adds functionaly to an existing function, and in this case, it adds functionality in the profile view where the user must be logged in to view this page



```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from .forms import UserRegisterForm
```



```
    return render(request, 'users/register.html', {'form': f

@login_required
def profile(request):
    return render(request, 'users/profile.html')
```

5) create the profile.html template to render the user name there

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <h1>{{ user.username }}</h1>
{% endblock content %}
```

6) Add the profile link in the URL

```
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from users import views as user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name='register'),
    path('profile/', user_views.profile, name='profile'),
    path('login/', auth_views.LoginView.as_view(template_name
    path('logout/', auth_views.LogoutView.as_view(template_na
    path('', include('blog.urls')),
]
```

7) Update Navbar Depending on user authentication status

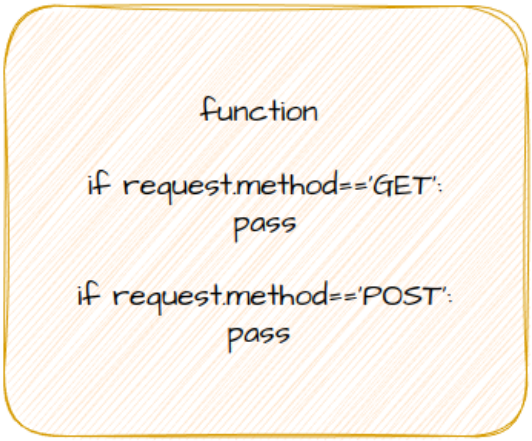
-if user is authenticated (aka loggin in) , show him the log out nav link.

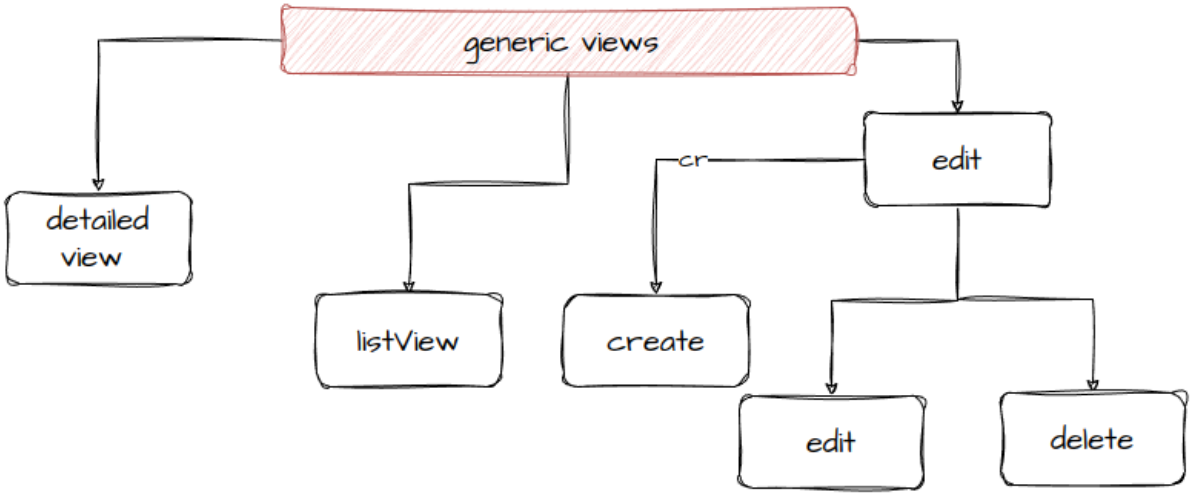
-If not, then show him the Login / Register nav links

```
  <!-- Navbar Right Side -->
  <div class="navbar-nav">
    {% if user.is_authenticated %}
      <a class="nav-item nav-link" href="{% url 'profile' %}">Profile</a>
      <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
    {% else %}
      <a class="nav-item nav-link" href="{% url 'login' %}">Login</a>
      <a class="nav-item nav-link" href="{% url 'register' %}">Register</a>
    {% endif %}
  </div>
</div>
```

Class Based Views (Create, Update, and Delete)

Created	@October 15, 2023 12:01 AM





List View

-youtube listinng your subscription videos

-Web page listing all blogs / posts / products

Detailed View

when clicking on a product from the list, a detailed page about the product is displayed

Update View & Delete View

ability to update blogs // videos

note : **update view can also be called edit view**

Steps to create a List View :

1) Add a class based list view in Views.py

---

1) Add a class based List View :

First : import it from the views.generic

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import Post


def home(request):
    context = {
```

Second : Add a class that inherits from ListView

- **give it the model name**
- **the template name you're gonna use as your new home page bc django server expects a template by default and its naming convention is app/model_viewtype.html**

  for ex : pages/Product_list.html
- **the context object : the variable name that we're gonna loop over**
- **the ordering (optional) : if you wanna load your items in a certain order .. the minus (-) means from newest to oldest aka newest at the top & oldest at the bottom**

```
def home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html', context)


class PostListView(ListView):
    model = Post
    template_name = 'blog/home.html' # <app>/<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']
```

Steps to create a detailed view

1) import it from the django.controb.views

in views.py

```
from django.shortcuts import render
from django.views.generic import ListView, DetailView
from .models import Post
```

```
class PostDetailView(DetailView):
    model = Post
```

2) add the link in the app urls.py

but don't forget to first import it

```
from django.urls import path
from .views import PostListView, PostDetailView
from . import views

urlpatterns = [
```

```
urlpatterns = [
    path('', PostListView.as_view(), name='blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('about/', views.about, name='blog-about'),
]
```

3) add template post_detail.html

with the naming convention : app/model_viewtype.html

and every object like post,title or whateeveris changed to object.title and so on

```
{% extends "blog/base.html" %}
{% block content %}
  <article class="media content-section">
    <img class="rounded-circle article-img" src="{{ object.author.profile
    <div class="media-body">
      <div class="article-metadata">
        <a class="mr-2" href="#">{{ object.author }}</a>
        <small class="text-muted">{{ object.date_posted|date:"F d, Y" }}
      </div>
      <h2 class="article-title">{{ object.title }}</h2>
      <p class="article-content">{{ object.content }}</p>
    </div>
  </article>
```

and now you can change the link of your old details.html page

```
href="{% url 'post-detail' post.id %}">{{ post.title }}</a></h
{ post.content }}</p>
```

Steps to use create view

1) Views

2) App Urls

3) Template holding both create & update

4) Models

1) Views

- import them
- create a class that inherits from CreateView
- give it the fields that you wanna have in your form (title, content)

```
from django.views.generic import (
    ListView,
    DetailView,
    CreateView
)
```

```
class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']
```

2) App Urls

import postdetailview and add its link

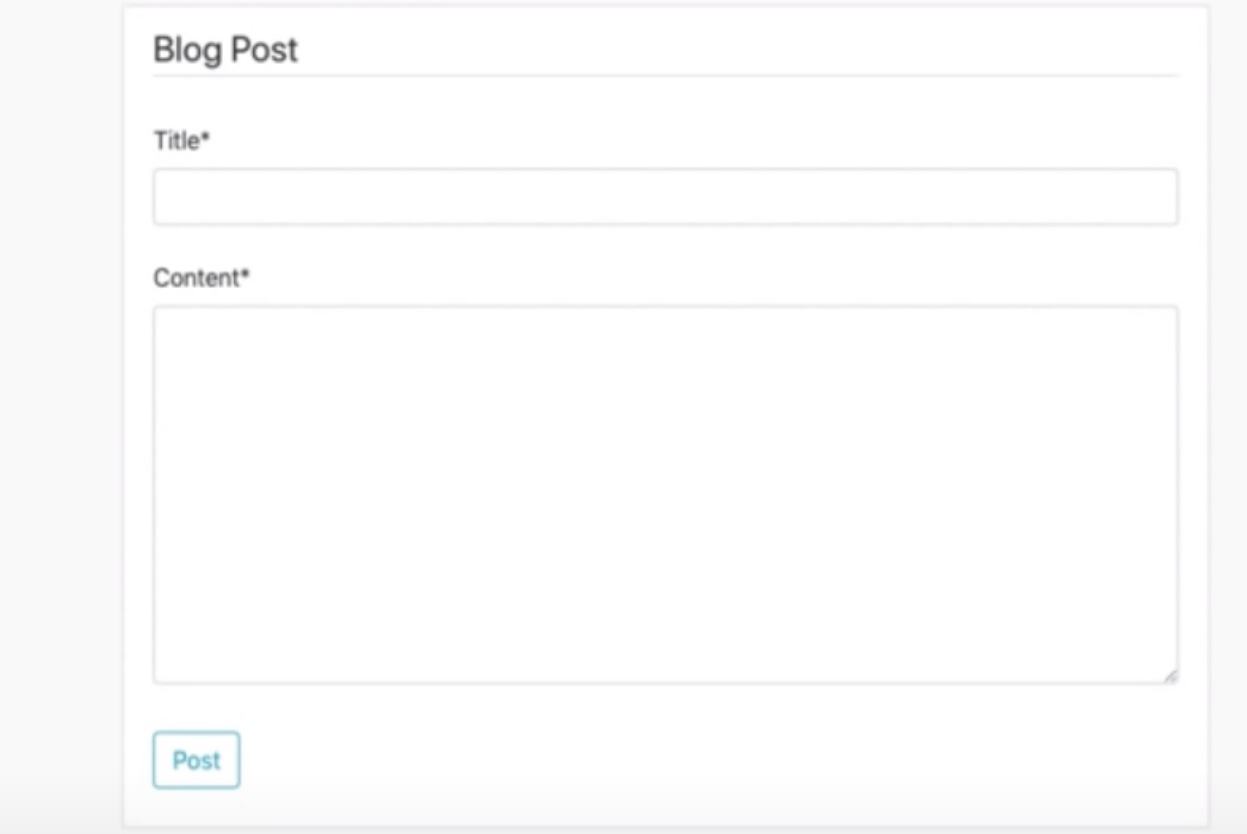
```
from django.urls import path
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView
)
from . import views

urlpatterns = [
    path('', PostListView.as_view(), name='blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-create'),
    path('about/', views.about, name='blog-about'),
]
```

3) Template holding both create & update

the same form we used a lot before but change the names to blog post or whatever the name you wanna be displayed

```
{% load crispy_forms_tags %}
{% block content %}
    <div class="content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Blog Post</legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info" type="submit">Post
            </div>
        </form>
    </div>
{% endblock content %}
```

the result :



4) Updating Views by setting the author as the current user

```
class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)
```

still you'll get an error because the server can't find the url to the model object you're referring to, so you'll have to edit it in the model by creating a get absolute url method that returns the path to any specific instance

5) Models

-First : get url of a particular route , so you'll need to use the reverse function

---

Note : Difference between redirect and reverse :

redirect —> Actually redirects you to a specific route

reverse —> Simply returns the full URL to that route as a string

---

-And in our situation, we just want the URL as a string, and the views is gonna handle the redirect for us.

so ..

- **first import reverse**

- **create the get_absolute_url method to tell django to return the full path as a string**

  (remember it needs a specific post with a primary key, so set

  kwargs={'pk': self.pk}

  so the instance of a post is a primary key

and it will direct you to the details page you just created

```
class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    date_posted = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('post-detail', kwargs={'pk': self.pk})
```

A login mixin

In your views.py :

1. **import it**

from django.contrib.auth.mixins import LoginRequiredMixin

so user is directed to login page whenever he opens the site

```python
from django.shortcuts import render
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import (
    ListView,
    DetailView,
    CreateView
)
```

2. **inherit it in the begging on the left in your create view**

```python
class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)
```

Steps to use update view :

1) Views

2) App Urls

3) Same Create View Form Template

4) Check if user updating is the actual author of the post

1) Views

- **import it**

```
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView
)
```

- **create the class (similar to the create view)**

```
class PostUpdateView(LoginRequiredMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)
```

2) App urls

- import it

```
from django.urls import path
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView,

)
from . import views
```

- **include its path**

(here, it's similar to the detail view with the int:pk ..)

include('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),

```
erns = [
h('', PostListView.as_view(), name='blog-home'),
h('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
h('post/new/', PostCreateView.as_view(), name='post-create'),
h('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
h('about/', views.about, name='blog-about'),
```

3) Same Create View Form Template

4) Check if user updating is the actual author

- **using another mixin called UserPassesTestMixin**

```
django.shortcuts import render
django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
django.views.generic import (
istView,
etailView,
reateView,
```

- **make your update view class inherit form it (and it MUST BE ON THE LEFT of the UpdateView)**

```
ass PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
  model = Post
  fields = ['title', 'content']

  def form_valid(self, form):
      form.instance.author = self.request.user
```

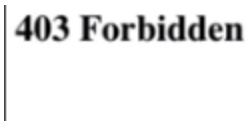
- **create a test function to test the mixin**

  -to get the post you're currently trying to update

  -check if the current user is the author of the post

```python
class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False
```

and now, if you try to update another user's post, you'll get a 403 Forbidden message



Steps to use delete view

(similar to detail view)

1) Views

2) App Urls

3) Template : post_confirm_delete.html

4) Update Navbar

1) Views

- import

```
from django.contrib.auth.mixins impo
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView,
    DeleteView
)
from .models import Post
```

- **class delete view**

(like detail view in the model only , but takes the same mixin as the update view and the test_func)

```
class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False
```

- Provide success url for after deleting a post

success_url = '/' .. the slash is the url of home page

```python
class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post
    success_url = '/'

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False
```

2) App Urls

import it above and write it path with the same int:pk ..

```python
PostDeleteView

. import views

atterns = [
path('', PostListView.as_view(), name='blog-home'),
path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
path('post/new/', PostCreateView.as_view(), name='post-create'),
path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-delete'),
path('about/', views.about, name='blog-about'),
```

3) Template : post_confirm_delete.html

-copy paste our normal forms and change the desired values

-Here we don't need the form, so we can delete the crispy forms tags .. it's just a page to confirm our delete post and put the title of the post : {{ object.title }}

- **create an href that takes you back to the detailed view of this exact post,**
- **we can put href = "{% url ' post-detail' object.id %}**

and pass in the primary key so we can create a URL of the post detail page for this post ID

---

4) Update Navbar & post_detail.html

In Navbar, add a **link to create a new post**

```
-- Navbar Right Side -->
iv class="navbar-nav">
{% if user.is_authenticated %}
  <a class="nav-item nav-link" href="{% url 'post-create' %}">New Post</a>
  <a class="nav-item nav-link" href="{% url 'profile' %}">Profile</a>
  <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
{% else %}
  <a class="nav-item nav-link" href="{% url 'login' %}">Login</a>
  <a class="nav-item nav-link" href="{% url 'register' %}">Register</a>
{% endif %}
```

In the post_Details.html template, add **links to update & delete the posts in the post_detail.html**

for update :

```
ounded-circle article-img" src="{{ object.author.profile.image.url }}">
edia-body">
'article-metadata">
'mr-2" href="#">{{ object.author }}</a>
ss="text-muted">{{ object.date_posted|date:"F d, Y" }}</small>
ct.author == user %}
="btn btn-secondary btn-" href="{% url 'post-update' object.id %}">Update</a
}
                    btn-secondary

rticle-title">{{ object.title }}</h2>
ticle-content">{{ object.content }}</p>
```

for delete :

```
class="mr-2" href="#">{{ object.author }}</a>
all class="text-muted">{{ object.date_posted|date:"F d, Y" }}</small>
if object.author == user %}
a class="btn btn-secondary btn-sm mt-1 mb-1" href="{% url 'post-update' obje
a class="btn btn-danger btn-sm mt-1 mb-1" href="{% url 'post-delete' object.
endif %}
>
lass="article-title">{{ object.title }}</h2>
```