Django

1. After creating the project : create a views.py where we render html web pages .

After We create a function that return an HTML string with response then we need to define it in URLS by calling the function in new path like Django says.

Views.py

from django.http import HttpResponse

"""

to render html web pages

"""

'''

def home(request):

    return response

the function take in  a request(django send request) and return HTML response

'''

HTML\_STRING=""" <h1>Hello World </h1> """

def home\_view(request):

    return HttpResponse(HTML\_STRING)

in urls:

"""Try\_django URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

    https://docs.djangoproject.com/en/4.0/topics/http/urls/

Examples:

Function views

    1. Add an import:  from my\_app import views

    2. Add a URL to urlpatterns:  path('', views.home, name='home')

Class-based views

    1. Add an import:  from other\_app.views import Home

    2. Add a URL to urlpatterns:  path('', Home.as\_view(), name='home')

Including another URLconf

    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

from .views import home\_view

urlpatterns = [

    path('',home\_view),

    path('admin/', admin.site.urls),

]

And so we can see the HTML : Hello World in our page

We can also make the HTML\_STRING more dynamic by writing it insite the function with variables : HTML\_STRING=f”<h1>Hello {name}</h1>”

**Database**

I create an app in the Django project with the code : python manage.py startproject blog

* We create the database models in the blog models :

# Create your models here.

class Article(models.Model):

    title=models.TextField()

    content=models.TextField()

* Then we register the model in Settings :

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'blog',

]

* Then we run : python manage.py makemigrations----- python manage.py migrate

**View the data from the database in the web page**

1-store the data manually from python manage.py shell:

from blog.models import Article

obj=Article.objects.create(title='the third title',content='the third content')

obj.save()

in views.py:

from blog.models import Article

def home\_view(request):

    article\_obj =Article.objects.get(id=1)

    article\_title=article\_obj.title

    article\_content=article\_obj.content

    H1\_string=f"""

    <h1>{article\_title}</h1>

    """

    p\_string=f"""

    <p>{article\_content}</p>

    """

    HTML\_STRING=H1\_string+p\_string

    return HttpResponse(HTML\_STRING)

# and so we can read in our home :

# this is the first title

Hello world

Another way to make the Django templates:

def home\_view(request):

    article\_obj =Article.objects.get(id=1)

    context= {

         "title":article\_obj.title,

         "id":article\_obj.id,

         "content":article\_obj.content

     }

     #django templates

    HTML\_STRING="""

    <h1>{title} (id : {id})</h1>

    <p>{content}</p>

    """.format(\*\*context)

    return HttpResponse(HTML\_STRING)

**using HTML templates:**

* Make new folder called templates where we store our HTML pages
* In the setting – TEMPLATES – DIRS paste the path of your templates folder(use ‘/’ not ‘\’) or the best way is ('DIRS': [ BASE\_DIR /'templates',],)#but take care of the location of your templates folder .
* Create HTML file like : home-view.html
* Write the html code in home-view.html
* In views : from django.template.loader import render\_to\_string
* def home\_view(request):
* article\_obj =Article.objects.get(id=1)
* context= {
* "title":article\_obj.title,
* "id":article\_obj.id,
* "content":article\_obj.content
* }
* #django templates
* HTML\_STRING=render\_to\_string("home-view.html",context=context)
* return HttpResponse(HTML\_STRING)

use double {{variable name }} in the html view to load variables :

<h1>{{title}} (id : {{id}})</h1>

<p>{{content}}</p>

**Inheritance**

To make our html page take info from other page we use the extend block :

Home-view.html:

{% extends 'base.html' %}

{% block content %}

<h1>{{title}} (id : {{id}})</h1>

<p>{{content}}</p>

{% endblock content %}

Base.html:

<!DOCTYPE html>

<html>

<body>

<h1>this is the base bage</h1>

{% block content %}

{% endblock content %}

<p>the end of base paragraph.</p>

</body>

</html>

The result :

# this is the base bage

# this is the first title (id : 1)

Hello world

the end of base paragraph.

**For loop in views & templates:**

Create a list in views:

def home\_view(request):

    article\_obj =Article.objects.get(id=1)

    my\_list=[300,250,200,150,100,50,0]

    my\_objects=Article.objects.all()

    context= {

         "my\_objects":my\_objects,

         "my\_list":my\_list,

         "title":article\_obj.title,

         "id":article\_obj.id,

         "content":article\_obj.content

     }

     #django templates

    HTML\_STRING=render\_to\_string("home-view.html",context=context)

    return HttpResponse(HTML\_STRING)

and in home-view :

{% extends "base.html" %}

{% block content %}

<h1>{{title}} (id : {{id}})</h1>

<p>{{content}}</p>

<ul>

{% for x in my\_list %}

<li>

{{x}}

</li>

{% endfor %}

</ul>

<ul>

    {% for x in my\_objects %}

    <li>

    {{x}}

    </li>

    {% endfor %}

    </ul>

{% endblock content %}

Result will be :

# this is the base bage

# this is the first title (id : 1)

Hello world

* 300
* 250
* 200
* 150
* 100
* 50
* 0
* Article object (1)
* Article object (2)
* Article object (3)

the end of base paragraph.

**If statement in html – Django :**

<ul>

    {% for x in my\_objects %}

        {% if x.title %}

        <li>

            {{x.title}}-{{x.content}}

            </li>

        {% endif %}

    {% endfor %}

    </ul>

{% endblock content %}

result ==

* this is the first title-Hello world
* this is the second title-Hello world
* the third title-the third content

# Dynamic URL Routing

If we want make new url routing for example : <http://127.0.0.1:8000/blog/1/>

1-we define new url in urls:

urlpatterns = [

    path('',home\_view),

    path('blog/<int:id>/',home\_view),

    path('admin/', admin.site.urls),

]

2-we will receive an ERROR :

|  |  |
| --- | --- |
| **Exception Value:** | home\_view() got an unexpected keyword argument 'id' |

To solve it we need :

3-change the argument for home\_view (request, \*args, \*\*Kwargs)

def home\_view(request,id=None, \*args, \*\*kwargs):

4-in blog view : from .models import Article

Create a function for the blog.view :

def blog\_detail\_view(request,id):

    article\_obj=None

    if id is not None:

        article\_obj=Article.objects.get(id=id)

    context={

        "object":article\_obj,

    }

    return render(request,"blog/detail.html",context=context)

then make the change in urls:

from blog import views

urlpatterns = [

    path('',home\_view),

    path('blog/<int:id>/',views.blog\_detail\_view),

    path('admin/', admin.site.urls),

]

Then we create the blog folder in templates and we create detail.html:

{% extends 'base.html' %}

{% block content %}

<h1>{{object.title}}</h1>

<h1>{{object.content}}</h1>

{% endblock content %}

Result:

# this is the base bage

# this is the first title (id : 1)

Hello world

* 300
* 250
* 200
* 150
* 100
* 50
* 0
* [this is the first title-Hello world](http://127.0.0.1:8000/blog/1/)
* [this is the second title-Hello world](http://127.0.0.1:8000/blog/2/)
* [the third title-the third content](http://127.0.0.1:8000/blog/3/)

the end of base paragraph.

After click on the link :🡺

# this is the base bage

# this is the first title

# Hello world

the end of base paragraph.

**Admin**

python manage.py createsuperuser

* + http://127.0.0.1:8000/admin/

Ahmad/15975300

# Register Model / data in the Admin site

In App admin (blog):

from .models import Article

admin.site.register(Article)

and to:

- display the title and id of the records instead of objects :

- creating search field to search for records:

from .models import Article

class ArticleAdmin(admin.ModelAdmin):

    list\_display=['id','title']

    search\_fields=['title','content']

admin.site.register(Article,ArticleAdmin)

# Search Form & Request Data

* make input form in base.html:
* <form action='/blog/'>
* <label for="q" class="form-label">Enter the id </label>
* <input type="text" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" name="q">

* <label for="exampleInputPassword1" class="form-label">Password</label>
* <input type="password" class="form-control" id="exampleInputPassword1">

* <button type="submit" class="btn btn-primary">Submit</button>
* </form>
* define a new function in blog view: def article\_search\_view(request):

def article\_search\_view(request):

    query\_dict=request.GET  #this is dictionary

    #query=query\_dict.get("q")

    try:

        query=int(query\_dict.get("q"))

    except:

        query=None

    article\_obj=None

    if query is not None:

        article\_obj=Article.objects.get(id=query)

    context={

        "object":article\_obj

    }

  return render(request, "blog/search.html", context=context)

* define the function in urls: path('blog/',views.article\_search\_view),
* create search.html:

{% extends 'base.html' %}

{% block content %}

<div class="card" style="width: 18rem;">

    <div class="card-body">

      <h5 class="card-title">id : {{object.id}}</h5>

      <p class="card-text">{{object.title}}</p>

      <p class="card-text">{{object.content}}</p>

      <a href="#" class="card-link">Card link</a>

      <a href="#" class="card-link">Another link</a>

    </div>

  </div>

{% endblock content %}

# Basic HTML Input Form

1- create function in view to create data (you can create the create.html page first 😊):

def blog\_create\_view(request):

    context={}

    if request.method=="POST":

        title=request.POST.get("title")

        content=request.POST.get("content")

        article\_object=Article.objects.create(title=title, content=content)

        context['object']=article\_object

        context['created']=True

    return render(request,"blog/create.html",context=context)

1. Define it in urls :

path('blog/create/',views.blog\_create\_view),

1. Create the create.html page :
2. <!DOCTYPE html>
3. <html>
4. <head>
5. <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
6. </head>
7. <body>
9. {% block content %}
10. {% if not created %}
11. <div style='margin-top:30px;'>
12. <form action='.' method="POST" >
13. {% csrf\_token %}
15. <label for="q" class="form-label">Enter the Title </label>
16. <input type="text" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" name="title">

19. <label for= class="form-label">content</label>
20. <textarea  class="form-control" id="exampleInputPassword1" name='content'></textarea>

23. <button type="submit" class="btn btn-primary" >Create Article</button>
24. </form>
25. </div>
26. {% else %}
27. <p> your article was created </p>
28. <a href='/blog/{{object.id}}/'> {{object.title}}-{{object.content}}</a>
29. {% endif %}
30. {% endblock content %}
31. <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
32. </body>
33. </html>

4- we need to use the {% csrf\_token %} in the form action because we are POSTing data to the database .

# Create a Login View to Authenticate Users

* 1. Create a folder if you like and call it accounts then create login.html inside it :
* <!DOCTYPE html>
* <html>
* <head>
* <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
* </head>
* <body>
* {% block content %}
* <div style='margin-top:30px;'>
* <form  method="POST" >
* {% csrf\_token %}
* {% if Error  %}
* <p style='color:red'>{{ Error }}</p>
* {% endif %}
* <label for="q" class="form-label">user name</label>
* <input type="text" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" name="username">

* <label for= class="form-label">password</label>
* <input type="password"  class="form-control" id="exampleInputPassword1" name='password'>

* <button type="submit" class="btn btn-primary" >login</button>
* </form>
* </div>
* {% endblock content %}
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
* </body>
* </html>

2- create the login\_view function in blog/views :

Import :

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login

def login\_view(request):

    if request.method=="POST":

        username=request.POST.get("username")

        password=request.POST.get("password")

        user=authenticate(request, username=username, password=password)

        if user is None:

            context={"Error":"Invalid username or password"}

            return render(request, "accounts/login.html", context)

        login(request, user)

        return redirect('/')

    return render(request, "accounts/login.html", {})

3- define the view in your project urls:

path('login/',views.login\_view),

# Logout View

* 1. Create logout.html:
* <!DOCTYPE html>
* <html>
* <head>
* <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
* </head>
* <body>
* {% block content %}
* {% if  request.user.is\_authenticated %}
* <div style='margin-top:30px;'>
* <form  method="POST" >
* {% csrf\_token %}
* <div class="alert alert-primary" role="alert">
* Are you sure you want to logout?
* </div>
* <button type="submit" class="btn btn-primary" >Logout</button>
* </form>
* </div>
* {% else %}
* <div>
* <div class="alert alert-success" role="alert">
* you are NOT logged in. would you like to  :  <a href="/login" class="alert-link">lOGIN</a>?</div>
* </div>
* {% endif %}
* {% endblock content %}
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
* </body>
* </html>

2-creat logout\_view function in blog/views:

def logout\_view(request):

    if request.method=="POST":

        logout(request)

        return redirect("/login/")

    return render(request, "accounts/logout.html", {})

3-define the urls :

path('logout/',views.logout\_view),

# Creating a login User Required

To make a login required to use/visit a page we will need to :

* In views we call the decorator login\_required:

from django.contrib.auth.decorators import login\_required

-then in setting we need to define :

LOGIN\_URL='/login/'

To redirect the user to the login page

* We call thi decorator before the function that we want to lock :
* @login\_required
* def blog\_create\_view(request):
* context={}
* if request.method=="POST":
* title=request.POST.get("title")
* content=request.POST.get("content")
* article\_object=Article.objects.create(title=title, content=content)
* context['object']=article\_object
* context['created']=True
* return render(request,"blog/create.html",context=context)

**another way to Create Form**

we have already a class for the Article in models:

from django.db import models

# Create your models here.

class Article(models.Model):

    title=models.TextField()

    content=models.TextField()

make class ArticleForm in forms:

import the Article from models:

from blog.models import Article

then:

class ArticleForm(forms.ModelForm):

    class Meta:

        model=Article

        fields=['title','content']

    def clean(self):

            data=self.cleaned\_data

            title=data.get("title")

            qs=Article.objects.filter(title\_\_icontains=title)

            if qs.exists():

                self.add\_error("title", f"{title} is already in use")

            return data

the function clean will check if the title exists and will not save the data if its exists

then in views the new function for the form will be :

def blog\_create\_view(request):

    form=ArticleForm(request.POST or None)

    context={

        "form":form

    }

    if form.is\_valid():

        article\_object=form.save()

        context["form"]=ArticleForm()

    return render(request,"blog/create.html",context=context)

that’s it 😊

# Register a User via built in Model Form

1-in views import : from django.contrib.auth.forms import UserCreationForm

2-define a register\_view function in views :

def register\_view(request):

    form = UserCreationForm(request.POST or None)

    if form.is\_valid():

        user\_obj=form.save()

        return redirect('/login')

    context={"form": form }

    return render(request,"accounts/register.html", context)

3- creathe path in urls :

path('register/',views.register\_view),

4-register.html =>

<!DOCTYPE html>

<html>

  <head>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

  </head>

<body>

{% block content %}

{% if not request.user.is\_authenticated %}

    <div style='margin-top:30px;'>

        <form  method="POST" >{% csrf\_token %}

            {{form.as\_p}}

            <button type="submit" class="btn btn-primary" >Register</button>

        </form>

          <p>Already have an account ?<a href="/login">Login</a></p>

    </div>

{% else %}

  <div>

    <div class="alert alert-success" role="alert">

      you are already logged in and cannot register would you like to  :  <a href="/logout" class="alert-link">Logout</a>?

    </div>

  </div>

  {% endif %}

{% endblock content %}

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</body>

</html>

Login view with the AuthenticationForm

The login\_view:

def login\_view(request):

    if request.method=="POST":

        form=AuthenticationForm(request, data=request.POST)

        if form.is\_valid():

            user=form.get\_user()

            login(request,user)

            return redirect('/')

    else:

        form=AuthenticationForm(request)

    context={"form":form}

    return render(request, "accounts/login.html", context)

-the login.html :

<!DOCTYPE html>

<html>

  <head>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

  </head>

<body>

{% block content %}

{% if not request.user.is\_authenticated %}

    <div style='margin-top:30px;'>

        <form  method="POST" >

            {% csrf\_token %}

            {{form.as\_p}}

            <button type="submit" class="btn btn-primary" >login</button>

          </form>

          <p>need  an account ?<a href="/register">Register</a></p>

    </div>

{% else %}

  <div>

    <div class="alert alert-success" role="alert">

      you are already logged in. would you like to  :  <a href="/logout" class="alert-link">Logout</a>?

    </div>

  </div>

  {% endif %}

{% endblock content %}

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

</body>

</html>

Slug fields in models:

1. In models we change the Article class (the table for Article) and import :

from django.utils.text import slugify

then add a slug field:

class Article(models.Model):

    title=models.CharField(max\_length=120)

    content=models.CharField(max\_length=120)

    timestamp=models.DateTimeField(auto\_now\_add=True)

    updated=models.DateTimeField(auto\_now=True)

    slug=models.SlugField(blank=True, null=True)

    def save(self,\*args, \*\*kwargs ):

        #this function mean :

        #obj=Article.objects.get(id=1)

        #set something

        if self.slug is None:

            self.slug=slugify(self.title)

        super().save(\*args, \*\*kwargs)

        #obj.save()

        #do another something

The function save: its make the slug field take the title and slugify it automatically .

**Slug with signals :**

from django.db.models.signals import pre\_save, post\_save

class Article(models.Model):

    title=models.CharField(max\_length=120)

    content=models.CharField(max\_length=120)

    timestamp=models.DateTimeField(auto\_now\_add=True)

    updated=models.DateTimeField(auto\_now=True)

    slug=models.SlugField(blank=True, null=True)

    def save(self,\*args, \*\*kwargs ):

        #this function mean :

        #obj=Article.objects.get(id=1)

        #set something

        #if self.slug is None:

            #self.slug=slugify(self.title)

        super().save(\*args, \*\*kwargs)

        #obj.save()

        #do another something

def article\_pre\_save(sender, instance, \*args, \*\*kwargs):

    if instance.slug is None:

        instance.slug=slugify(instance.title)

pre\_save.connect(article\_pre\_save, sender=Article)

def article\_post\_save(sender, instance,created, \*args, \*\*kwargs):

    if created:

        instance.slug=slugify(instance.title)

        instance.save()

post\_save.connect(article\_post\_save, sender=Article)

**unique slugs:**

from django.db import models

from django.utils import timezone

from django.utils.text import slugify

from django.db.models.signals import pre\_save, post\_save

# Create your models here.

class Article(models.Model):

    title=models.CharField(max\_length=120)

    content=models.CharField(max\_length=120)

    timestamp=models.DateTimeField(auto\_now\_add=True)

    updated=models.DateTimeField(auto\_now=True)

    slug=models.SlugField(blank=True, null=True)

    def save(self,\*args, \*\*kwargs ):

        #this function mean :

        #obj=Article.objects.get(id=1)

        #set something

        #if self.slug is None:

            #self.slug=slugify(self.title)

        super().save(\*args, \*\*kwargs)

        #obj.save()

        #do another something

def slugify\_instance\_title(instance, save=False):

    slug=slugify(instance.title)

    qs=Article.objects.filter(slug=slug).exclude(id=instance.id)

    if qs.exists():

        slug=f"{slug}-{qs.count()+1}"

    instance.slug=slug

    if save:

        instance.save()

    return instance

def article\_pre\_save(sender, instance, \*args, \*\*kwargs):

    if instance.slug is None:

        slugify\_instance\_title(instance, save=False)

pre\_save.connect(article\_pre\_save, sender=Article)

def article\_post\_save(sender, instance,created, \*args, \*\*kwargs):

    if created:

        slugify\_instance\_title(instance, save=True)

post\_save.connect(article\_post\_save, sender=Article)

# Slugs in Dynamic Urls

1-we change the blog\_detail\_view in views to :

def blog\_detail\_view(request,slug=None):

    article\_obj=None

    if slug is not None:

        article\_obj=Article.objects.get(slug=slug)

        try:

            article\_obj=Article.objects.get(slug=slug)

        except Article.DoesNotExist:

            raise Http404

        except Article.MultipleObjectsReturned:

            article\_obj=Article.objects.filter(slug=slug).first()

        except:

            raise Http404

    context={

        "object":article\_obj,

    }

    return render(request,"blog/detail.html",context=context)

2- we change the url in urls to :

path('blog/<slug:slug>/',views.blog\_detail\_view),

3- in home\_view.html we need to change the href for navigating to the records in article and make it depending on slug not id :

<ul>

    {% for x in my\_list %}

    <li>{{x}}</li>

    {% endfor %}

    </ul>

        <ul>

          {% for x in my\_objects %}

            {% if x.title %}

            <li>

               <a href='/blog/{{x.slug}}/'> {{x.title}}-{{x.content}}</a>

                </li>

            {% endif %}

          {% endfor %}

        </ul>

The result will be : <http://127.0.0.1:8000/blog/this-is-the-first-title/>

**Get absolute url**

* make function in models :
* def get\_absolute\_url(self):
* return f'/blog/{self.slug}/'

**in home view it will be :**

<ul>

    {% for x in my\_list %}

    <li>{{x}}</li>

    {% endfor %}

    </ul>

        <ul>

          {% for x in my\_objects %}

            {% if x.title %}

            <li>

               <a href='{{x.get\_absolute\_url}}'> {{x.title}}-{{x.content}}</a>

                </li>

            {% endif %}

          {% endfor %}

        </ul>

**In** search :

<div class="card" style="width: 18rem;">

      {%for object in object\_list%}

      {%if object.title%}

        <div class="card-body">

          <h5 class="card-title">id : {{object.id}}</h5>

          <p class="card-text">{{object.title}}</p>

          <p class="card-text">{{object.content}}</p>

          <a href="{{object.get\_absolute\_url}}" class="card-link">details</a>

        </div>

# Complex Search using Django Q Lookups

* 1. We change the search\_view function in views :

def article\_search\_view(request):

    query=request.GET.get('q')

    qs=Article.objects.all()

    if query is not None:

        lookups=Q(title\_\_icontains=query) #we can use multible Q lookuos =Q(title\_\_icontains=query)|Q(content\_\_icontains=query)

        qs=Article.objects.filter(lookups)

    context={

        "object\_list":qs

    }

    return render(request, "blog/search.html", context=context)

* 1. In search.html :
* <!DOCTYPE html>
* <html>
* <head>
* <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
* </head>
* <body>
* <div class="card" style="width: 18rem;">
* {%for object in object\_list%}
* {%if object.title%}
* <div class="card-body">
* <h5 class="card-title">id : {{object.id}}</h5>
* <p class="card-text">{{object.title}}</p>
* <p class="card-text">{{object.content}}</p>
* <a href="{{object.get\_absolute\_url}}" class="card-link">details</a>
* </div>
* {% endif %}
* {% endfor %}
* </div>
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
* </body>
* </html>

# Model Managers & Custom QuerySets for Search:

# In Video 47: (https://www.youtube.com/watch?v=3qViDQ6SUC8&list=PLEsfXFp6DpzRMby\_cSoWTFw8zaMdTEXgL&index=47)

**Recipe App**

We will make a recipe app so we will :

* Create the app : python manage.py startapp recipe
* In the app models we create out database/tables :
* from django.conf import settings
* from django.db import models
* # Create your models here.
* class Recipe(models.Model):
* user=models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)
* name=models.CharField(max\_length=220)
* description=models.TextField(blank=True, null=True)
* direction=models.TextField(blank=True, null=True)
* timestamp=models.DateTimeField(auto\_now\_add=True)
* updated=models.DateTimeField(auto\_now=True)
* active=models.BooleanField(default=True)
* class RecipeIngredient(models.Model):
* recipe=models.ForeignKey(Recipe, on\_delete=models.CASCADE)
* name=models.CharField(max\_length=220)
* description=models.TextField(blank=True, null=True)
* quantity=models.CharField(max\_length=50)
* unit=models.CharField(max\_length=50)
* direction=models.TextField(blank=True, null=True)
* timestamp=models.DateTimeField(auto\_now\_add=True)
* updated=models.DateTimeField(auto\_now=True)
* active=models.BooleanField(default=True)
* In general settings define the new app in installed apps :
* INSTALLED\_APPS = [
* 'django.contrib.admin',
* 'django.contrib.auth',
* 'django.contrib.contenttypes',
* 'django.contrib.sessions',
* 'django.contrib.messages',
* 'django.contrib.staticfiles',
* 'blog',
* 'recipes',
* ]
* then makemigration – migrate.

**Showing/registering the models in Admin page**

* 1. in Admin.py :
* from django.contrib import admin
* from django.contrib.admin.options import TabularInline
* from django.db.models.base import Model
* # Register your models here.
* from .models import Recipe, RecipeIngredient
* #to register the model in admin
* admin.site.register(RecipeIngredient)
* #to show the RecipeIngredient in a better way
* class RecipeIngredientInline(admin.StackedInline):
* model=RecipeIngredient
* extra=0 #to see few fields in the page
* #fields=['name','quantity','unit','directions'] #if i want to limit the fields
* class RecipeAdmin(admin.ModelAdmin):
* inlines=[RecipeIngredientInline]
* list\_display=['name','user']
* readonly\_fields=['timestamp','updated'] #so we cant change this fields
* raw\_id\_fields=['user'] #to show the user as id in admin page
* admin.site.register(Recipe,RecipeAdmin)

# Tests Relationships between Models

# Link: <https://www.youtube.com/watch?v=c48rGz1EfFk&list=PLEsfXFp6DpzRMby_cSoWTFw8zaMdTEXgL&index=52>

# Custom Validation for Unit Measurements

# create validators.py

# we can install pint(pip install pint)

# in validators :

* import pint
* from django.core.exceptions import ValidationError
* from pint.errors import UndefinedUnitError
* valid\_unit\_measurements=['pounds','lbs','gram']
* def validate\_unit\_of\_measure(value):
* ureg=pint.UnitRegistry()
* try:
* single\_unit=ureg[value]
* except UndefinedUnitError as e:
* raise ValidationError(f"{e}")
* except:
* raise ValidationError(f"{value} is not a valid unit of measure")

# 3- in models : from .validators import validate\_unit\_of\_measure

# 4-in the unit field we add :

# unit=models.CharField(max\_length=50, validators=[validate\_unit\_of\_measure])

**CRUD**

1-create forms.py to make our forms to create data :

from django import forms

from .models import Recipe

class RecipeForm(forms.ModelForm):

    class   Meta:

        model=Recipe

        fields=['name','description','direction']

2-in model/class Recipe we create the get\_absolute\_url:

    def get\_absolute\_url(self):

        return reverse("recipes:detail", kwargs={"id":self.id})

3- in views we create the functions that we need (create, details,update,list, forms) :

from django.contrib.auth.decorators import login\_required

from django.shortcuts import redirect, render, get\_object\_or\_404

from .models import Recipe

from .forms import RecipeForm

# Create your views here.

#CRUD--> create retrieve update delete

@login\_required

def recipe\_list\_view(request):

    qs=Recipe.objects.filter(user=request.user)

    context={

        "object\_list":qs

    }

    return render(request, "recipes/list.html",context)

@login\_required

def recipe\_detail\_view(request, id=None):

    obj=get\_object\_or\_404(Recipe, id=id, user=request.user)

    context={

        "object":obj

    }

    return render(request, "recipes/detail.html",context)

@login\_required

def recipe\_create\_view(request):

    form=RecipeForm(request.POST or None)

    context={

        "form":form

    }

    if form.is\_valid():

        obj=form.save(commit=False)

        obj.user=request.user

        obj.save()

        return redirect(obj.get\_absolute\_url())

    return render(request, "recipes/create-update.html",context)

@login\_required

def recipe\_update\_view(request, id=None):

    obj=get\_object\_or\_404(Recipe, id=id, user=request.user)

    form=RecipeForm(request.POST or None, instance=obj)

    context={

        "form":form,

        "object":obj

    }

    if form.is\_valid():

        form.save()

        context['message']='Data Saved.'

    return render(request, "recipes/create-update.html",context)

urls -- create new urls.py with :

from django.urls import path

from .views import \*

app\_name='recipes'

urlpatterns = [

    path("", recipe\_list\_view, name='list'),

    path("create/", recipe\_create\_view, name='create'),

    path("<int:id>/edit/", recipe\_update\_view, name='update'),

    path("<int:id>/", recipe\_detail\_view, name='detail'),

]

The models.py will be like :

from django.conf import settings

from django.db import models

from .validators import validate\_unit\_of\_measure

from django.urls import reverse

# Create your models here.

class Recipe(models.Model):

    user=models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

    name=models.CharField(max\_length=220)

    description=models.TextField(blank=True, null=True)

    direction=models.TextField(blank=True, null=True)

    timestamp=models.DateTimeField(auto\_now\_add=True)

    updated=models.DateTimeField(auto\_now=True)

    active=models.BooleanField(default=True)

    def get\_absolute\_url(self):

        return reverse("recipes:detail", kwargs={"id":self.id})

    def get\_edit\_url(self):

        return reverse("recipes:update", kwargs={"id":self.id})

    def get\_ingredients\_children(self):

        return self.recipeingredient\_set.all()

class RecipeIngredient(models.Model):

    recipe=models.ForeignKey(Recipe, on\_delete=models.CASCADE)

    name=models.CharField(max\_length=220)

    description=models.TextField(blank=True, null=True)

    quantity=models.CharField(max\_length=50)

    unit=models.CharField(max\_length=50, validators=[validate\_unit\_of\_measure])

    direction=models.TextField(blank=True, null=True)

    timestamp=models.DateTimeField(auto\_now\_add=True)

    updated=models.DateTimeField(auto\_now=True)

    active=models.BooleanField(default=True)

    def get\_absolute\_url(self):

        return self.recipe.get\_absolute\_url()

create-update.html:

<h1>create-update</h1>

{% if message %}

<p>{{ message }}</p>

{% endif %}

<div>

<form action='.' method='POST'>

{% csrf\_token %}

{{ form.as\_p }}

<button type='submit'>Save</button>

</form>

</div>

Detail.html :

<h1>Detail view</h1>

<h1>{{object.name}}<small><a href='{% url 'recipes:update' id=object.id %}'>Edit</a></small></h1>

<p>{{object.description}}</p>

<p>{{object.direction}}</p>

{% for ingredient in object.get\_ingredients\_children  %}

<h1>{{ingredient.name}}</h1>

<p>{{ingredient.quantity}}</p>

<p>{{ingredient.unit}}</p>

<p>{{ingredient.updated}}</p>

{% endfor %}

List.html:

<h1>My Recipes</h1>

<h1></h1>

<div style='margin-top:30px;'>

    <a href='{% url 'recipes:create' %}'>Add Recipe</a>

    <ul>

        {% for x in object\_list   %}

        <li>

            <a href='{{ x.get\_absolute\_url }}'>{{ x.name }} </a>

        </li>

        {% endfor %}

    </ul>

</div>