

django

**AMIR MOHAMMAD
HOUSHMAND**

DON'T REPEAT YOURSELF



Hello
what is your name :

TOPICS

01 Getting Started

Why Django? What I learn in this course 3

02 COURSE SETUP

How crate a Django project and setup VIRTUALENV 4

03 URLS & VIEWS

How crate a simple page in Django with views and URL 5

04 TEMPLATES & STATIC FILES

How render a html file and load static file and like this 9

06 Data & Models

Database and models 19

07 Admin

Administration and use feature of that 30

08 Relationships

Add relationships to models 34

10 Forms

Form and unput the data of users like login and signup 40

11 Class Views

About Class Views 54

12 File Uploads

About how upload the data in django 60

13 Sessions

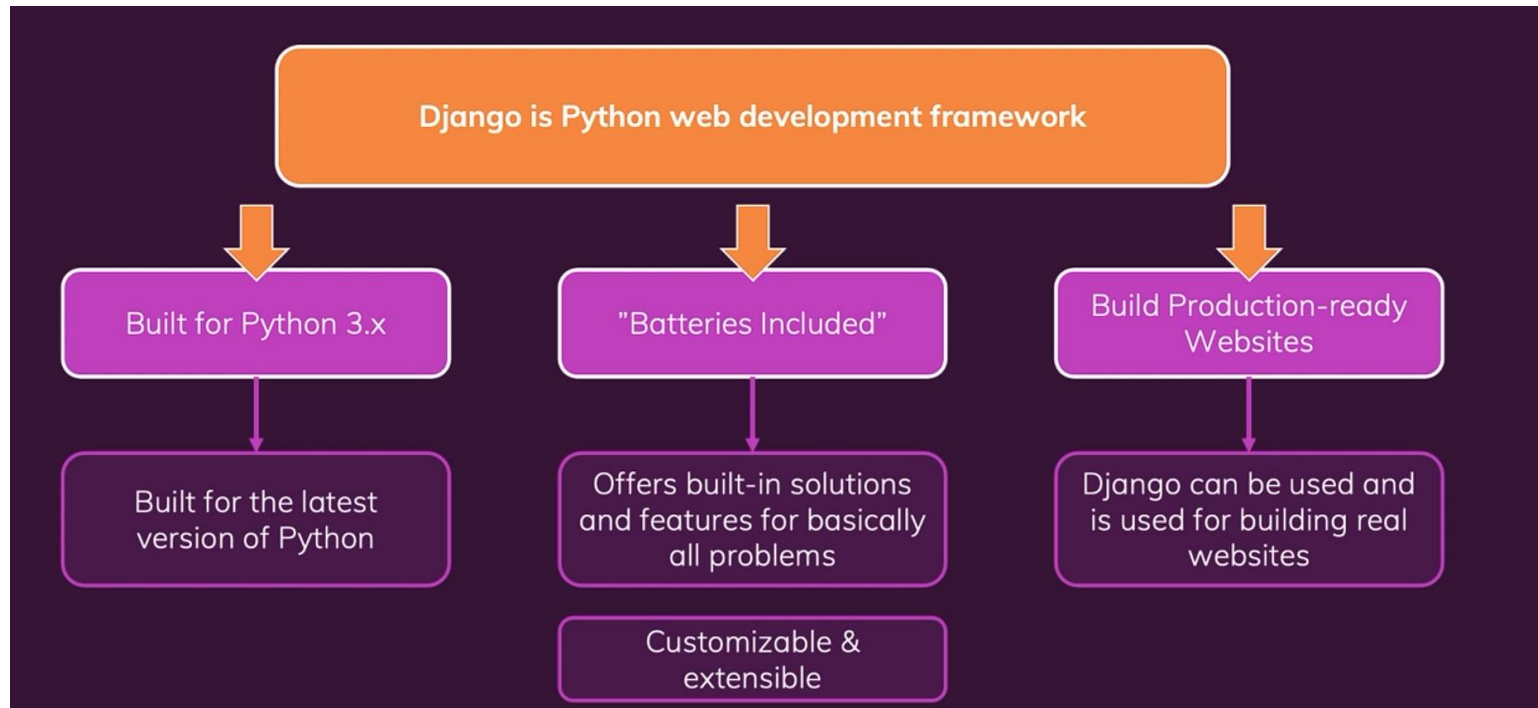
what are sessions and how we can use it 65

14 Important notes

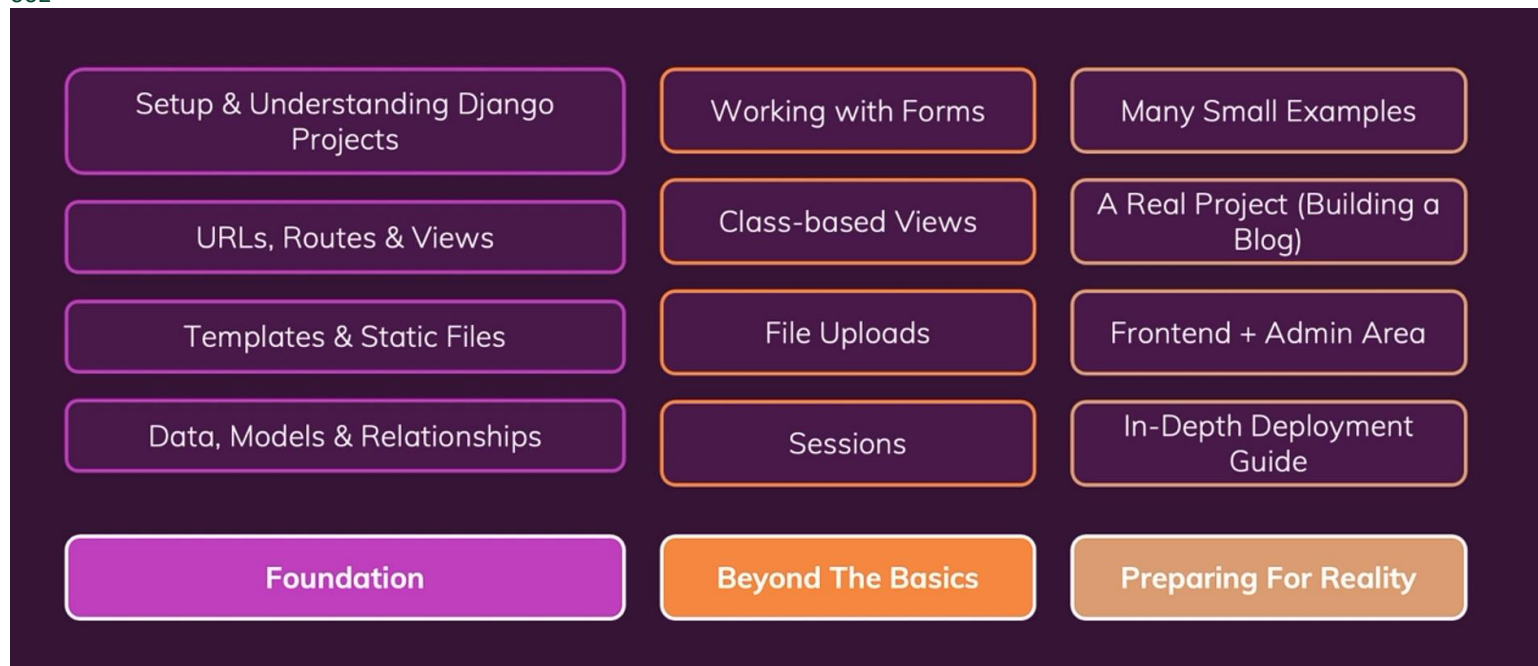
A short section on what we can do after finishing the book 67

01

GETTING STARTED



002



004

All this part is sync by this course:
<https://www.udemy.com/course/python-django-the-practical-guide/>

02

COURSE SETUP

For create a venv:

```
pip install virtualenv  
virtualenv -p python3 venv
```

Now active the venv:

```
.\venv\Scripts\activate
```

And then install Django:

```
pip install django
```

Create Django project:

```
django-admin startproject hash_neco
```

Run the Django project:

```
python manage.py runserver
```

Now go to browser and type this:

```
http://127.0.0.1:8000/
```

Or:

```
localhost:8000
```

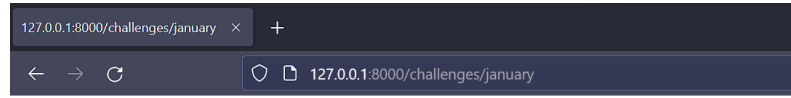
For create app in Django:

```
python manage.py startapp NameOfApp
```

03

URLS & VIEWS

004



Hello, world. You're at the hash_neco index.

For show this page we must config URLs and views

In our app(challenges) views we must add some **function** so I create a function named index you can named this function everything you want!

hash_neco\challenges\views.py

```
def index(request):  
    return HttpResponse("Hello, world. You're at the hash_neco index.")
```

but for Django understand what is HttpResponse we need import this

```
from django.http import HttpResponse
```

this **function** needs a URL to Know where this message shows up! So in our app we create an `urls.py` in this file we need import our views file and an `urlpatterns` to add our URLs for add an URL we need import `path` from Django URLs.

hash_neco\challenges\urls.py

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

```
urlpatterns = [  
    path("january", views.index, name="january"),  
]
```

Path("here we put the URL", here the view we want for the URL we add, and here we give this URL a name for some usage),

But Django doesn't know we add some URL so in main folder(hash_neco) `urls.py` we add some path for add our paths in our app but for doing this we need import `include` from Django URLs

hash_neco\hash_neco\urls.py

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

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

006

Now if we want to add the other month, we most use Dynamic Path to don't hard work so for this we add a new URL

hash_neco\challenges\urls.py

```
urlpatterns = [
    path("<month>", views.month_index, name="month_index"),
]
```

Path("here we need a parameter for month so in django we most put parameter in <> (Dynamic Segments)", views, name)

So, in views we add a function

hash_neco\challenges\views.py

```
def month_index(request, month):
    if month == "january":
        challenge_text = "this is january challenge:)"
    elif month == "february":
        challenge_text = "this is february challenge:)"
    elif month == "march":
        challenge_text = "this is march challenge:)"
    return HttpResponse(challenge_text)
```

in the second parameter we most add the parameter in URL so when we type something the text goes in month and then if the text It was equal to one of month, then the challenge_text is the text and then Django render this page.

But if user input number or something else we most handle this with HttpResponseNotFound like this (this text is in function):

```
else:
    return HttpResponseNotFound("<h1>404</h1>")
```

So, if I input:

http://127.0.0.1:8000/challenges/january

http://127.0.0.1:8000/challenges/february

http://127.0.0.1:8000/challenges/march

http://127.0.0.1:8000/challenges/somethingelse

This is output:

➔ this is january challenge

➔ this is february challenge

➔ this is march challenge

➔ 404

oo7

But if we want user give us a number and we see what the month is and then render the month we most give another parameter (Dynamic Segments) like <int:month> and <str:month> so if we input num it can be go on another view like :

hash_neco\challenges\urls.py

```
urlpatterns = [
    path("<int:month>", views.month_index_bynumber, name="month_index"),
    path("<str:month>", views.month_index, name="month_index"),
]
```

oo8

Let's back to our app for better code in views we can made a dictionary for month like this :

hash_neco\challenges\views.py

```
month_challenge = {
    "january": "this is january challenge:",
    "february": "this is february challenge:",
    "march": "this is march challenge:",
    ...
}
```

And then in function:

```
def month_index(request, month):
    try:
        challenge_text = month_challenge[month]
        return HttpResponse(challenge_text)
    except:
        return HttpResponseNotFound("<h1>404</h1>")
```

Then we user try and except for if user input something else show 404 error

o09

Now we want to add if user type number of month Django redirect to month for redirect we need import HttpResponseRedirect from django http

```
from django.http import HttpResponse, HttpResponseNotFound, HttpResponseRedirect
```

And then in function:

```
def month_index_bynumber(request, month):
    months = list(month_challenge.keys())
    if month > len(months):
        return HttpResponseNotFound("<h1>404</h1>")
    forward_month = months[month - 1]
    return HttpResponseRedirect(f"/challenges/{forward_month}")
```

First, we get month number then we list the dictionary keys this is output:

```
['january', 'february', 'march']
```

And then if number is bigger than len(months) return 404 error but if is smaller we do this

```
forward_month = months[month - 1]
```

we do this because lists start at 0. then we redirect user to moth.

o10

One of other thing we hard coded is redirect path Why?

Because if we change the main URL (path("challenges/", include("challenges.urls"))), our redirect method doesn't work

Because we hard code so for right way we must name "<str:month>" URL like name="month_index"

than we import reverse from django URLs in views.py

```
from django.urls import reverse
```

then in month_index_bynumber function we can do this :

```
redirect_path = reverse("month_index", args=[forward_month])
```

```
redirect_path = reverse("here we put URL name", args=[here we input month name])
```

→

```
/Month_index/forward_month
```

Like:

```
/challenges/January
```

So, if I input this:

Main URL(challenges URL) → challenges

127.0.0.1:8000/challenges/1 → 127.0.0.1:8000/challenges/january

Main URL(challenges URL) → challenge

127.0.0.1:8000/challenge/1 → 127.0.0.1:8000/challenge/january

Main URL(challenges URL) → challenge

127.0.0.1:8000/challenges/1 → 404

Main URL(challenges URL) → challenges

127.0.0.1:8000/challenge/1 → 404

With no change in code just use reverse()

o11

Now if we want add dome html code in our response we can do this :

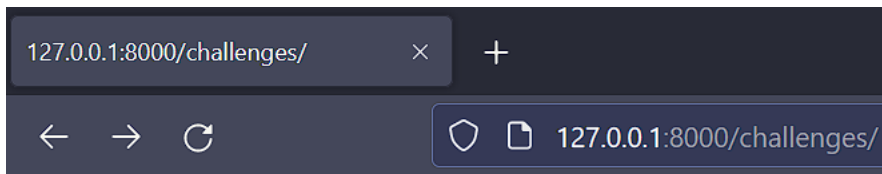
```
...
challenge_text = month_challenge[month]
response_data = "<h1>{}</h1>".format(challenge_text)
return HttpResponse(response_data)
...
```

Or :

```
response_data = f"<h1>{challenge_text}</h1>"
```

o12

Now if we want to make a thing like hub for month, we most create a new view so let's go



- [january](#)
- [february](#)
- [march](#)

(Like this)

First, we add our path in URL

hash_neco\challenges\urls.py

```
path("", views.chall_index, name="index"),
```

Then in views we add a new function called `chall_index` now we get our month name then we create an empty list and then with if we input the month and the link of month with reverse method as you see the I input all of this in `` tags and then django can render this.

hash_neco\challenges\views.py

```
def chall_index(request):
    months = list(month_challenge.keys())
    listofmonth = ""
    for month in months:
        listofmonth += f"<li><a href='{reverse('month_index', args=[month])}'>{month}</a></li>"
    return HttpResponse(f"<ul>{listofmonth}</ul>")
```

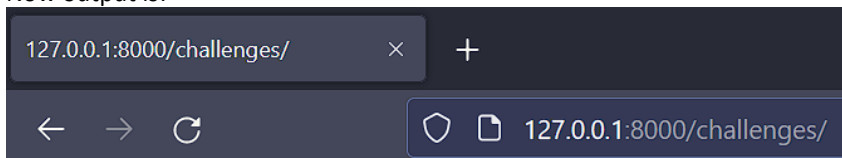
this is output in html:

```
<ul>
  <li><a href='/challenges/january'>january</a></li>
  <li><a href='/challenges/february'>february</a></li>
  <li><a href='/challenges/march'>march</a></li>
</ul>
```

Now if we want show capitalize format of month, we can do this:

```
...
for month in months:
    capitalized_month = month.capitalize()
    listofmonth += f"<li><a href='{reverse('month_index', args=[month])}'>{capitalized_month}</a></li>"
...
```

Now output is:



- [January](#)
- [February](#)
- [March](#)

https://github.com/houshmand-2005/hash_neco/tree/urls_views_003

04

TEMPLATES & STATIC FILES

So, as you know our website it's not looks good so we must add some CSS code and better html for this we need template and static file. Let's do this!

oo2

Now we most add a folder for our template for doing this in our app folder we create a sub folder named **template** this name most be template. Now in this folder we create a sub folder for our app, so I named this to challenges.

hash_neco\challenges\templates\challenges

base folder\App folder\templates\our App name

now in this folder we create HTML file like this: hash_neco\challenges\templates\challenges\challenge.html

hash_neco\templates\challenges\challenge.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Monthly Challenge</title>
</head>
<body>
  <h1>this month Challenge</h1>
  <h2>just a test</h2>
</body>
</html>
```

Now for render this file in views.py we import render_to_string form django.template.loader

hash_neco\challenges\views.py

```
from django.template.loader import render_to_string
```

Now our html file convert to string for render. For response data we most add render_to_string function.

```
def month_index(request, month):
    try:
        challenge_text = month_challenge[month]
        response_data = render_to_string("challenges/challenge.html")
        return HttpResponse(response_data)
    except:
        return HttpResponseNotFound("<h1>404</h1>")
```

in render_to_string() we add our template. Now if I run this, we get 404 error. Because django doesn't know where the template is. For that in setting file of project we most add our template.

There is two ways. **First** in template list(TEMPLATES = [])in DIRS list we add our path('DIRS':[]).we can use Base DIR to find base directory of project and then we type our path

hash_neco\ settings.py

```
TEMPLATES = [
    ...
    'DIRS': [
        BASE_DIR / "challenges" / "templates"
    ]
    ...
]
```

But the **second** way (this is better for subpages and the first way is better for main page) in INSTALLED_APPS list (INSTALLED_APPS[])

We add template.

```
INSTALLED_APPS = [
    ...
    'challenges',
    ...
]
```

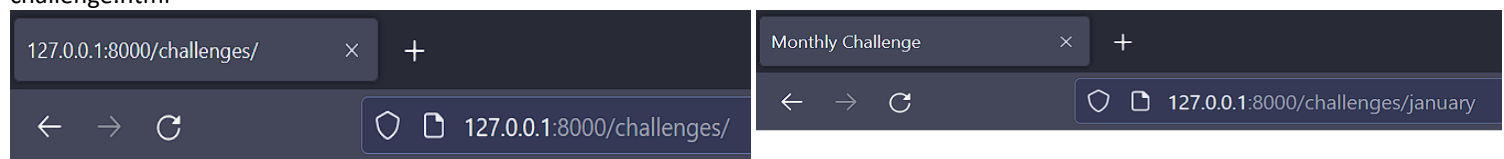
But at first APP_DIRS in TEMPLATES must be true to read template in INSTALLED_APPS.

```
TEMPLATES = [
    ...
    'APP_DIRS': True,
    ...
]
```

Now django knows about our template but we just add main folder of template no challenges folder. do you think we have problem? No
hash_neco\challenges\views.py

```
response_data = render_to_string("challenges/challenge.html")
```

here we say where our template is and now django go in template main folder and then as we say go in challenges folder and read challenge.html



- [January](#)
- [February](#)
- [March](#)

this month Challenge

just a test

Now if I go in one of month I see my html file.

oo3

now we can do better in views, before we use render to string but now its better to use render from django shortcuts.

```
from django.shortcuts import render
```

but render needs author argeumnt named request, now we don't user it but it can exteact data from page.

```
def month_index(request, month):
    ...
    return render(request, "challenges/challenge.html")
    ...
```

oo4

Now if we want show month challenge in html file we can do this, in render we have therd argeumnt in this argeumnt we can push a vriable

```
def month_index(request, month):
    ...
    challenge_text = month_challenge[month]
    return render(request, "challenges/challenge.html", {
        "challenge_text": challenge_text
    })
    ...
```

Now we have accses challenges_text vriable in html file.

```
<h1>this month Challenge</h1>
<h2>{{challenge_text}}</h2>
```

In html files django has Django Template Language (DTL) for use our commands we must put in { { here } }

oo5

If we wanna add name of month in html file we can do this

```
def month_index(request, month):
    ...
    challenge_text = month_challenge[month]
    return render(request, "challenges/challenge.html", {
        "challenge_text": challenge_text,
        "nameofmonth": month.capitalize(),
    })
    ...
```

Now in html file we add name of month var

hash_neco\templates\challenges\challenge.html

```
...
<title>{{nameofmonth}} Challenge</title>
...
<body>
    <h1>{{nameofmonth}} Challenge</h1>
    <h2>{{challenge_text}}</h2>
</body>
```

Now out put in html file is

http://127.0.0.1:8000/challenges/january

```
<html lang="en">
<head>
    <title>January Challenge</title>
</head>
<body>
    <h1>January Challenge</h1>
    <h2>this is january challenge:</h2>
</body>
</html>
```

0o6

For capitalaze we can do another way to do this. We can delete capitallaze in view

hash_neco\challenges\views.py

```
...
"nameofmonth": month,
...
```

and in html file we can use template filter.like this:

hash_neco\templates\challenges\challenge.html

```
...
<body>
    <h1>{{nameofmonth|title}} Challenge</h1>
    <h2>{{challenge_text}}</h2>
</body>
...
```

{{ nameofmonth|title }} this means you can capital the text in nameofmonth.in django we have a lot of them like:

cut

Removes all values of arg from the given string.

For example:

{{ value|cut:" " }}

If value is "String with spaces", the output will be "Stringwithspaces".

Full list : <https://docs.djangoproject.com/en/4.0/ref/templates/builtins/#built-in-filter-reference>

oo8

As you know our index page(<http://127.0.0.1:8000/challenges/>) is work with for loop and render In view but the better way is create a loop in htm fie for each month.so first we must create another html file for this page named index.html

hash_neco\challenges\views.py

```
def chall_index(request):
    months = list(month_challenge.keys())
    return render(request, "challenges/index.html", {
        "monthnames": months,
    })
```

Here we send all of month in a list in html file.

hash_neco\templates\challenges\index.html

```
...
<body>
  <ul>
    {% for month in monthnames %}
    <li><a href="">{{ month|title }}</a></li>
    {% endfor %}
  </ul>
</body>
...
```

In html file we can use for loop with django template language

```
{% for name_of_var in our_list %}
```

Code

```
{% endfor %}
```

We **must** close for loop in DTL(django template language)

Now the output is:

URL: http://127.0.0.1:8000/challenges/

```
<body>
  <ul>
    <li><a href="">January</a></li>
    <li><a href="">February</a></li>
    <li><a href="">March</a></li>
  </ul>
</body>
```

o09

Now for link location we can do this:

In <a> tag we do this: href="/challenges/{{ month }}"

But we have problem if we changes sub url("/challenges/...") this link dosen't work. As you see in before we use reverse function for this but in DTL we can't do this. we can use this {% url %} in this tag we have this {% url 'name of url in url.py' dynamic segment %} (like reverse)

hash_neco\challenges\urls.py

```
...
path("<str:month>", views.month_index, name="month_index"),
...
```

The name of this URL is month_index so we do this:

hash_neco\templates\challenges\index.html

```
...
{% for month in monthnames %}
<li><a href="{% url 'month_index' month %}">{{ month|title }}</a></li>
{% endfor %}
...
```

httpc127.0.0.1:8000/challenges/

```
<body>
  <ul>
    <li><a href="/challenges/january">January</a></li>
    <li><a href="/challenges/february">February</a></li>
    <li><a href="/challenges/march">March</a></li>
  </ul>
</body>
```

o10

Now if one of month doesn't have challenge text we wanna tell there is no challenge for this month
So in dictionary of month for one of month we set challenge text None

```
month_challenge = {
    "january": "this is january challenge:",
    "february": None,
    "march": "this is march challenge:",
}
```

So now if we go in `http://127.0.0.1:8000/challenges/february` it shows

```
...
    <h1>February Challenge</h1>
    <h2>None</h2>
...
```

Now in challenge html file we can add some logic with if and else

```
{% if something == 1 %}
code
{% else %}
Some code
{% endif %}
```

Now for this job we can do this:

```
...
<body>
    <h1>{{nameofmonth|title}} Challenge</h1>
    {% if challenge_text is not None %}
    <h2>{{challenge_text}}</h2>
    {% else %}
    <h2>No challenge for {{nameofmonth}}</h2>
    {% endif %}
</body>
...
```

`http://127.0.0.1:8000/challenges/february`

```
...
<body>
    <h1>February Challenge</h1>
    <h2>No challenge for february</h2>
</body>
...
```

011

as you see when we create a new page we need write all of html file but we can do better with copy all of skeleton of html file, and then every thing that page needs more we added for this page. for this we can create a base template folder for our project

`\template`

Now in this folder we create a base html file

`\template\base.html`

In this file we copy all we need for a base html file like this :

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title></title>
    </head>
    <body>
    </body>
</html>
```

Now we use block for add data to this page blocks need a name like this `{% block content %}` and we need close this block `{% endblock %}`

So for our `/challenges` url we need customize title and add our month names so we create two tag for this.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>{% block title %}place holder*{% endblock %}</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
</html>

```

*if we don't have anything from the page for this tag this tx shows but we send something this text is delete

In index.html file we need do something to read the skeleton file from base.html form the root folder so for doing this we need a new tag named extends.**this tag must be upper than all of codes (first line of codes).**

hash_neco\templates\challenges\index.html

```

{% extends "base.html" %}
...

```

Now index file read base.html file but at first we need tell django where is base.html

So in settings file in DIRS(Because this template is root) we add root template folder

hash_neco\settings.py

```

TEMPLATES = [
...
    'DIRS': [
        BASE_DIR / "template",
    ],
...
]

```

Now we can delete all of other code(base code) from index.html.but for month and title of page we can use block tags

hash_neco\templates\challenges\index.html

```

{% extends "base.html" %}
{% block title %}All challenges{% endblock %}
{% block content %}
<ul>
    {% for month in monthnames %}
    <li><a href="{% url 'month_index' month %}">{{ month|title }}</a></li>
    {% endfor %}
</ul>
{% endblock %}

```

Evreything in block can replace in base.

Output → http://127.0.0.1:8000/challenges/

```

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>All challenges</title>
    </head>
    <body>
<ul>
    <li><a href="/challenges/january">January</a></li>
    <li><a href="/challenges/february">February</a></li>
    <li><a href="/challenges/march">March</a></li>
</ul>
    </body>
</html>

```

o12

now for challnge pages we add this blocks

at first we delete unuseles code and then we add extends tag

hash_neco\templates\challenges\challenge.html

```

{% extends "base.html" %}

```

And then we add other part in title block and content block.

```
{% block title %}{{nameofmonth|title}} Challenge{% endblock %}
{% block content %}
    <h1>{{nameofmonth|title}} Challenge</h1>
    {% if challenge_text is not None %}
    <h2>{{challenge_text}}</h2>
    {% else %}
    <h2>No challenge for {{nameofmonth}}</h2>
    {% endif %}
{% endblock %}
```

Output → http://127.0.0.1:8000/challenges/march*

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>March Challenge</title>
    </head>
    <body>
        <h1>March Challenge</h1>
        <h2>this is march challenge:</h2>
    </body>
</html>
```

*it can be other month

o13

now if we wanna add navigation bar to our html we can added in ech file but as you know in more pages we have navigattion so do you think we add to our base temaplate? Omm for main pages nav yes but if our site has different nav we must use **include** tag.

So in our template app we create a folder named include now in this folder we add header.html file(nav)

.\challenges\templates\challenges\includes\header.html

Now in this file we can create our nav like this:

.\challenges\templates\challenges\includes\header.html

```
<header>
    <nav><a href="{% url 'index' %}">All challenges</a></nav>
</header>
```

In challenges html file and index we can added with include command

hash_neco\templates\challenges\challenge.html and hash_neco\templates\challenges\index.html

```
...
{% include "challenges/includes/header.html" %}
...
```

In challenges html file and index file we see this now.

```
<header>
    <nav><a href="/challenges/">All challenges</a></nav>
</header>
```

Include tag has another feature. When we use a variable like nameofmonth in challenges file we have access this var in file we included(header.html)

```
...
{% block content %}
    {% include "challenges/includes/header.html" %}
    <h1>{{nameofmonth|title}} Challenge</h1>
    ...
{% endblock %}
...
```

And we can create a new vareable fot this header file like this.now active page is sent in header

```
...
{% block content %}
    {% include "challenges/includes/header.html" with active_page="challenge" %}
    ...
{% endblock %}
...
```

And another var for index page

hash_neco\templates\challenges\index.html

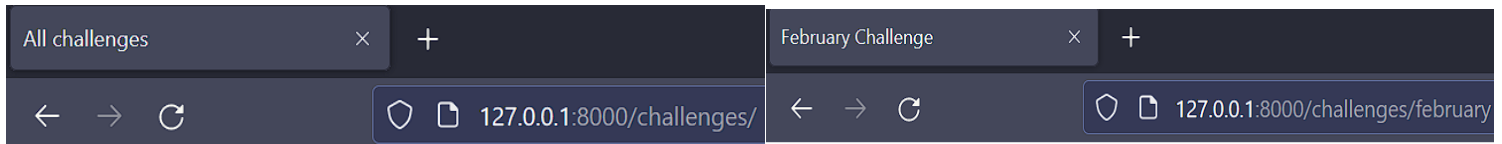
```
...
{% block content %}
    {% include "challenges/includes/header.html" with active_page="index" %}
{% endblock %}
...
```

Now in header file we can add this

\challenges\templates\challenges\includes\header.html

```
<header>
    <nav>your active page is : {{active_page}}</nav>
</header>
```

To see the var



[All challenges](#)

your active page is : index

- [January](#)
- [February](#)
- [March](#)

[All challenges](#)

your active page is : challenge

February Challenge

No challenge for february

So if I go in /challenges → active_page is index and in /challenges/february → active_page is challenge

o14

Accessing Dictionary Fields in Templates

When accessing dictionary data in a template, you DON'T use this syntax:

```
{{ myDictionary['some_key'] }}
```

Instead, you use the dot notation - as if it were a regular Python object:

```
{{ myDictionary.some_key }}
```

This might look strange, but keep in mind, that the DTL is a custom-made language. It looks like Python, but ultimately it is NOT Python - it's a language parsed and executed by Django. Hence, its syntax can deviate - just as it does here.

CALLING FUNCTIONS IN TEMPLATES

Calling functions in templates also works differently than it does in Python.

Instead of calling it, you use functions like regular variables or properties.

I.e., instead of:

```
{{ result_from_a_function() }}
```

you would use

```
{{ result_from_a_function }}
```

o15

as you know our 404 error page is still load from views but we can do it from templates like others. So because the 404 page is important we create this file in root template folder.

\template\404.html

```
{% extends "base.html" %} {%block title%} something went wrong. Please try again.
{%endblock%}
{%block content%}
<h1>404</h1>
```



```
<p>The page you are looking for does not exist.</p>
{%endblock%}
```

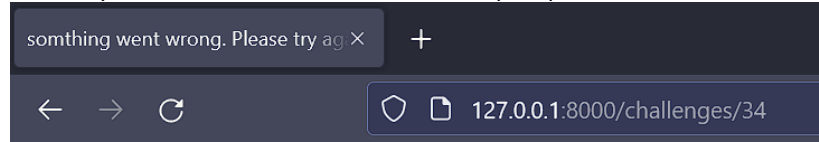
Now for load this we can use render to string again and put the var in HttpResponseRedirect we need import render to string

```
from django.template.loader import render_to_string
```

and then when we need 404 error we put this:

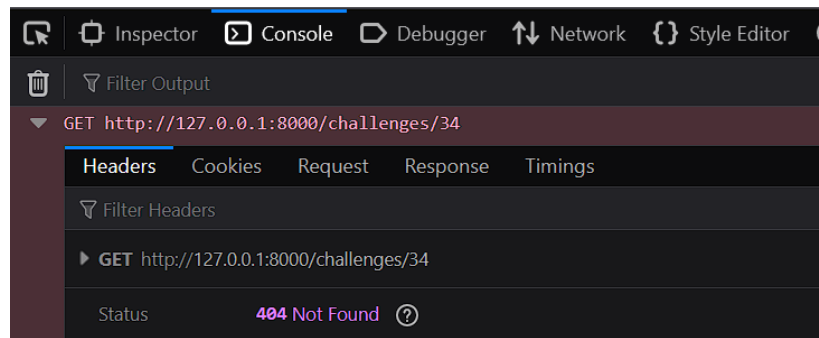
```
...
    response_data = render_to_string("404.html")
    return HttpResponseRedirect(response_data)
...
```

But why we don't render this and we use HttpResponseRedirect because in code we have different this is the standard form of 404 error



404

The page you are looking for does not exist.



As you see [here](#)

But also we can do something better .we import

```
hash_neco\challenges\views.py
```

```
from django.http import Http404 ...
```

this and now we can use this:

```
...
except:
    raise Http404()
...
```

Now if in our template we have a 404.html file automatically django return this.but this method only work when **debug** mode on setting is set to **false**

We can false this parameter but when we do this the **local server dosent work**.only in real server this method works.

```
\hash_neco\settings.py
```

```
DEBUG = True # we must set False this for using this method
```

But for now, let's use the previous way.(render_to_string)

Output: url → http://127.0.0.1:8000/challenges/34

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title> something went wrong. Please try again.
  </title>
</head>
  <body>
<h1>404</h1>
<p>The page you are looking for does not exist.</p>
  </body>
</html>
```

o16

Now as you see we don't have any style so we can add css in two way

First we add css in html file(internal) but second way(better way) is we create a sub folder to our project named static

In static folder we create a sub folder (Because we can add other file type like js) css and in this folder we can create a css file like: mycss.css

for example, we can add this style to UL tags:

static\css\mycss.css

```
ul {  
  list-style: none;  
}
```

Now in base html file we add a new block in head for css

hash_neco\template\base.html

```
...  
<head>  
  {% block css_files %}{% endblock %}  
</head>  
...
```

Now we can add any different style for each page.like index page.but at first we must create static tag

hash_neco\challenges\templates\challenges\index.html

```
...  
{% load static %}  
...
```

Now we can use the static folder

```
{% block css_files %}  
  <link rel="stylesheet" href="{% static 'css/mycss.css' %}">  
{% endblock %}
```

Now this link goes in base file in head part.

In href part we enter the folder name and then name of css file.

Now if we run still we see the dots for UL tags,why?

At first we must have this

\hash_neco\settings.py

```
INSTALLED_APPS = [  
    'django.contrib.staticfiles',  
]
```

In installed apps. And then in

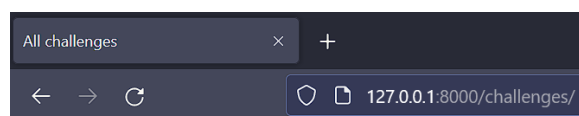
```
STATIC_URL = 'static/'
```

We must show django where is our static file.

Now we can run again (re run Because it won't work still, we don't restart the server) we cant see the dots.

But if still we have problem we must add this to setting.py

```
STATICFILES_DIRS = BASE_DIR, 'static'
```

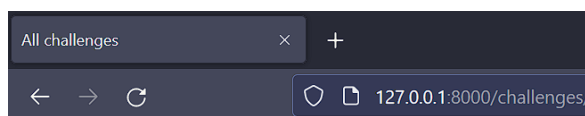


[All challenges](#)

your active page is : index

- [January](#)
- [February](#)
- [March](#)

Before



[All challenges](#)

your active page is : index

[January](#)
[February](#)
[March](#)

After

https://github.com/houshmand-2005/hash_neco/tree/Templates_and_Static_File_004

06

DATA & MODELS

oo6

ok now for this section I create a new django project named `book_store` and in this project, I create an app named `books`.

At first we must add app name in setting

`\book_store\settings.py`

```
INSTALLED_APPS = [  
    'books',  
]
```

Now in this app we have `models.py` app in this file we can add our models.

So, for doing this we need a class like `book` and in this class we have `books` parameter like `title`.

`\book_store\books\models.py`

```
class Book(models.Model):  
    title = models.CharField(max_length=50)  
    rating = models.IntegerField()
```

`class name_of_class(models.Model):`
parameter like `title`

Now every parameter has a argument (but not for all) like:

`title = models.CharField(max_length=50)`

`rating = models.IntegerField()`

`CharField` = for string value

`IntegerField` = for integer value

`max_length=50` = max length of string must be 50

oo7

But in table still we don't have these parameters. for add these we use this command:

Python `.\manage.py makemigrations`

Out put:

Migrations for 'books':

`books\migrations\0001_initial.py`
- Create model Book

Now in app folder in migration folder we have new migrate file this file made for our `models.py` file and django translate this file to SQL

But for doing this (make tables in database) we must run this:

python `.\manage.py migrate`

as you see we have lots of migrate alongside of our app migrate, these are from internal apps in django like admin panel.

Now if we see our data base we have these parameters

*

Table:

books_book

id	title	rating
Filter	Filter	Filter

*DB browser (SQLite)

oo8

you know if we wanna add any book we use `views.py` file but for now for learning we use shell

python `.\manage.py shell`

then we need import `Book` from `models`

`>>> from books.models import Book`

For insert data at first we create a var and make equals with `Book()` and in this class we have two parameters `title` and `rating` so we insert data like this: `varnamehere = Book(title="name of book", rating=an integer number like 1)`

`>>> test_book1 = Book(title="test 1 book", rating=5)`

`>>> test_book2 = Book(title="test 2 book", rating=2)`

`>>> test_book1.save()`

`>>> test_book2.save()`

At end we must save them.

Now if we use DB browser (SQLite) we can see these parameters

Table: books_book

	id	title	rating
	Filter	Filter	Filter
1	1	test 1 book	5
2	2	test 2 book	2

The id made automatically

For exit from shell use this:

```
>>> exit()
```

oo9

If we want to see our data is saved (if we don't use DB browser) we can use this:

```
>>> Book.objects.all()
```

```
<QuerySet [ <Book: Book object (1)>, <Book: Book object (2)> ]>
```

It means we have two parameters in this table.

o10

now if we want to see the name and rating we can add a function called `__str__` and then in this function we can set this like this:

```
class Book(models.Model):
    title = models.CharField(max_length=50)
    rating = models.IntegerField()
    def __str__(self):
        return f'{self.title} - ({self.rating})'
```

now if I run `Book.objects.all()` the out put is:

```
<QuerySet [ <Book: test 1 book - (5)>, <Book: test 2 book - (2)> ]>
```

As you see we don't migrate this file but it works, the methods don't need the migrate because nothing change in the database.

now we add some new parameters or change these we like for rating we want to number must between 1 and 5 so for doing this we can

Writing validators(<https://docs.djangoproject.com/en/4.0/ref/validators/>) but also we can use ready validators like

`MaxValueValidator` and `MinValueValidator`. for use this validators we do this:

```
class Book(models.Model):
    ...
    rating = models.IntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)])
    ...
```

Now if we add a new parameter like is best seller (get true or false) and add another_field we can code these

```
class Book(models.Model):
    ...
    another_field = models.CharField(max_length=100)
    is_best_seller = models.BooleanField()
    ...
```

Now if we want to add these parameters we use this command:

```
Python .\manage.py makemigrations
```

we have an error because the data before these changes they don't have any data for the parameters so for fix this we can use default or null set true.

```
class Book(models.Model):
    ...
    another = models.CharField(max_length=100, null=True)
    is_best_seller = models.BooleanField(default=False)
    ...
```

Now if we don't input the another is set to null and if we don't input `is_best_seller` is set to False

```
Python .\manage.py makemigrations
```

Out put:

Migrations for 'books':

```
books\migrations\0002_book_another_book_is_best_seller_alter_book_rating.py
- Add field another to book
- Add field is_best_seller to book
- Alter field rating on book
```

Then we make migrate

```
python .\manage.py migrate
```

Table: books_book

	id	title	rating	another	is_best_seller
	Filter	Filter	Filter	Filter	Filter
1	1	test 1 book	5	NULL	0
2	2	test 2 book	2	NULL	0

now we have access in to the shell to see items

like this : `name_of_class.objects.all()[index_of_data_we_want].name_of_parameters_we_want_to_see`

```
>>> Book.objects.all()[0].another
>>> Book.objects.all()[0].title
'test 1 book'
>>> Book.objects.all()[1].title
'test 2 book'
>>> Book.objects.all()[1].is_best_seller
False
```

o12

For update the data we can do like this:

```
>>> book1 = Book.objects.all()[0]
>>> book1.another = "test1ofbook1"
>>> book1.is_best_seller = True
```

But changes are not in database they are in just python memory for save changes in data base we must use:

```
>>> book1.save()
```

Now you can see the changes

```
>>> Book.objects.all()[0].is_best_seller
True
>>> Book.objects.all()[0].another
'test1ofbook1'
```

Table: books_book

	id	title	rating	another	is_best_seller
	Filter	Filter	Filter	Filter	Filter
1	1	test 1 book	5	test1ofbook1	1
2	2	test 2 book	2	NULL	0

o13

now for delete a data we can use this method

```
>>> book2 = Book.objects.all()[1]
>>> book2
<Book: test 2 book - (2)>
>>> book2.delete()
(1, {'books.Book': 1})
```

Now the second book is gone

```
>>> Book.objects.all()
<QuerySet [<Book: test 1 book - (5)>]>
```

Table: books_book

	id	title	rating	another	is_best_seller
	Filter	Filter	Filter	Filter	Filter
1	1	test 1 book	5	test1ofbook1	1

o14

Ok there is a new way to create a new book in data base like this:

```
>>> Book.objects.create(title="myStory", rating=1, another="test2", is_best_seller=False)
```

o15

I add some books for filter the data. If we want to query some data the before method isn't to good(`Book.objects.all()[1]`)

Now we use `Book.objects.get()`

Like:

```
>>> Book.objects.get(id=5)
<Book: Bioshock - (5)>
```

Now if I enter an id I deleted before we get an error (the id deleted forever)

```
>>> Book.objects.get(id=2)
```

error

```
>>> Book.objects.get(title="myStory")
<Book: myStory - (1)>
```

Now if we query some data and it returned more than one we get an error

```
>>> Book.objects.get(is_best_seller=False)
```

error

... `get()` returned more than one Book -- it returned 3!

So now if we want to query some data it returned more than one we can use `Book.objects.filter()`

```
>>> Book.objects.filter(is_best_seller=True)
```

```
<QuerySet [<Book: test 1 book - (5)>, <Book: Bioshock - (5)>, <Book: testtherat - (0)>]>
```

Return if we want to use `<, =, >` we must use this syntax → `Book.objects.filter(rating__lt=3)`

__lt means lower than

```
>>> Book.objects.filter(rating__lt=3)
<QuerySet [<Book: myStory - (1)>, <Book: Hello - (2)>, <Book: testtherat - (0)>]>
```

More like this syntax: (<https://docs.djangoproject.com/en/4.0/topics/db/queries/>)

Also, we can add another if for filter like this:

```
>>> Book.objects.filter(rating__lt=3, is_best_seller=True)
<QuerySet [<Book: testtherat - (0)>]>
```

if we want to find a text like something we use `TheNameOfVar__contains="the text"` (case sensitive but in SQLite this command is like `icontains`) and `TheNameOfVar__icontains` (not case sensitive).

o16

if we want to add 'and' & 'or' in query's we must import `Q` from `django` like this:

```
>>> from django.db.models import Q
```

and, or:

```
>>> Book.objects.filter(Q(rating__lt=3) | Q(is_best_seller=True), Q(title="myStory"))
<QuerySet [<Book: myStory - (1)>]>
```

the pipe (`|`) is for OR and '`,`' is for AND. the comma must be after OR

o19

now as you know we don't use models like this so for that lets Bild a simple project and then use models.

We must create a template folder

```
\book_store\books\templates\book_outlet
```

And in this folder, we can make these files

```
base.html, book_page.html, index.html
```

now we create a URL file for this app and fill with index view

```
\book_store\books\urls.py
```

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

so, for let's crate a view for this URL

```
\book_store\books\views.py
```

```
from django.shortcuts import render
def index(request):
    return render(request, 'book_outlet/index.html')
```

now we must include URL file in base URL file

```
\book_store\book_store\urls.py
```

```
from django.urls import path, include
urlpatterns = [
    ...
    path('', include('books.urls'))
]
```

Now in base.html file we create base html thing for other pages.
\\book_store\\books\\templates\\book_outlet\\base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{% block title %}{% endblock %}</title>
</head>
<body>
  {% block content %}
  {% endblock %}
</body>
</html>
```

And now for test we fill index.html

\\book_store\\books\\templates\\book_outlet\\index.html

```
{% extends "book_outlet/base.html" %}
{% block title %}
All Books
{% endblock %}
{% block content %}
<ul>
  <li>
    book
  </li>
</ul>
{% endblock %}
```

o20

now we use models how? In our views.py file we query data from database and pass into the template

at first we need import our Books class then we query all the data and then we send in to the template by 'books' name. Like this:

\\book_store\\books\\views.py

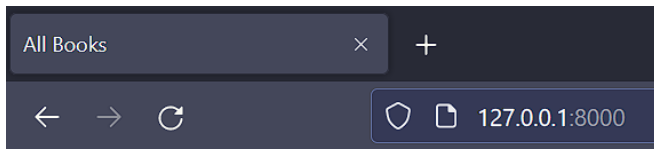
```
from .models import Book
def index(request):
    books = Book.objects.all()
    return render(request, 'book_outlet/index.html', {
        "books": books
    })
```

Now in index file we use this data with for loop

\\book_store\\books\\templates\\book_outlet\\index.html

```
{% block content %}
<ul>
  {% for book in books %}
  <li>
    {{ book.title }} rating: {{ book.rating }}
  </li>
  {% endfor %}
</ul>
{% endblock %}
```

Output:



- test 1 book rating: 5
- myStory rating: 1
- Hello rating: 2
- Bioshock rating: 5
- testtherat rating: 0
- 8rating rating: 8

Now we have the books from database.

o21

Now let's create book page for this we need a view.as you now in book page we want to user see one book and details of that.

We must query data by id of books. Then we render it.

`\book_store\books\views.py`

```
def book_page(request, id):
    book = Book.objects.get(pk=id)
    return render(request, 'book_outlet/book_page.html', {
        "title": book.title,
        "another": book.another,
        "reting": book.rating,
        "is_best_seller": book.is_best_seller
    })
```

This command is for query by id and pk it means primary key like id.so when the id we give it to this it looks for the same id and get details of that.

`book = Book.objects.get(pk=id)`

means

now we need id from URLs so let's create URL for that

`\book_store\book_store\urls.py`

```
urlpatterns = [
    ...
    path("<int:id>", views.book_page, name="book_page")
]
```

Now in book page we add these parameters.

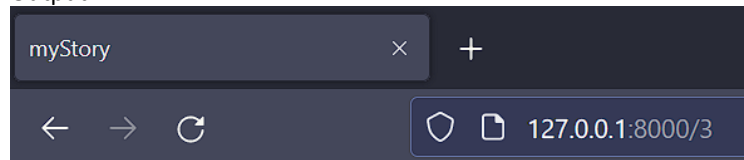
`\book_store\books\templates\book_outlet\book_page.html`

```
{% extends "book_outlet/base.html" %}
{% block title %}
{{title}}
{% endblock %}
{% block content %}
<h1>{{title}}</h1>
<h2>{{another}}</h2>
<p>this book has a rating of {{reting}} and {{another}}
{% if is_best_seller %}
and is the bestseller.
{% else %}
but isn't the bestseller
```



```
{% endif %}
</p>
{% endblock %}
```

Output:



myStory

test2

this book has a rating of 1 and test2 but isn't the bestseller

But if we enter some id, we don't have in database what we can do? We can use try and except

\book_store\books\views.py

```
from django.http import Http404
def book_page(request, id):
    try:
        book = Book.objects.get(pk=id)
    except:
        raise Http404()
    ...
```

But we can use another way (its smaller) but at first, we need import something

```
from django.shortcuts import render, get_object_or_404
```

and now if there is no object by id it's give us 404

get_object_or_404(NameOfModels, Condition)

```
def book_page(request, id):
    book = get_object_or_404(Book, pk=id)
```

o22

In main page now we want add link for each month a to the book_page of those books

In index.html file we can solve this with two ways. First way:

\book_store\books\templates\book_outlet\index.html

```
...
{% for book in books %}
    <a href="{% url 'book_page' book.id %}">
        <li>{{ book.title }} rating: {{ book.rating }}</li>
    </a>
{% endfor %}
...
```

In this way we use the book id.

Second way:

In this way also we get the id of book and with reverse method we create the link.

\book_store\books\models.py

```
from django.urls import reverse
class Book(models.Model):
    ...
    def get_absolute_url(self):
        return reverse("book_page", args=[self.id])
```

and in index.html we use this method we create.

\book_store\books\templates\book_outlet\index.html

```
...
{% for book in books %}
<a href="{ book.get_absolute_url }">
  <li>{{ book.title }} rating: {{ book.rating }}</li>
</a>
{% endfor %}
...
```

Output:

```
<body>
  <ul>
    <a href="/1">
      <li>test 1 book rating: 5</li>
    </a>
    <a href="/3">
      <li>myStory rating: 1</li>
    </a>
    <a href="/4">
      <li>Hello rating: 2</li>
    </a>
    <a href="/5">
      <li>Bioshock rating: 5</li>
    </a>
    <a href="/6">
      <li>testttherat rating: 0</li>
    </a>
    <a href="/7">
      <li>8rating rating: 8</li>
    </a>
  </ul>
</body>
```

o23

Output but return by id is not very good for SEO so we use slug.so for this in models we add a new var called slug

\book_store\books\models.py

```
class Book(models.Model):
    slug = models.SlugField(default="", null=False)
```

this slug field just hold slugs

now if we add a new book also, we add slug but what about old books?

We can create a new method for save (over writing save method) now for overwrite this we need 'args' and 'kwarg'

```
from django.utils.text import slugify
class Book(models.Model):
    ...
    def save(self, *args, **kwargs):
        self.slug = slugify(self.title)
        super().save(*args, **kwargs)
```

now if we save a book this method load and create slug by title

title = "This is title 2" → slug = "this-is-title-2"

now for apply these changes we need make migrations

python .\manage.py makemigrations

python .\manage.py migrate

now we need save all books to add slug, so we use shell

python .\manage.py shell

```
>>> from books.models import Book
>>> Book.objects.get(title="Hello").save()
>>> Book.objects.get(title="Hello").slug
'hello'
```

Books now we need do this for all books so we can use python power :)

```
>>> for i in range(1,8):
...     if i != 2:
...         Book.objects.get(id=i).save()
(if i != 2 because I don't have id 2 and python gets error)
```

Output:

Table: books_book						
	id	title	rating	another	is_best_seller	slug
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	test 1 book	5	test1ofbook1	1	test-1-book
2	3	myStory	1	test2	0	mystory
3	4	Hello	2	this_is_hello_story	0	hello
4	5	Bioshock	5	the_best_game	1	bioshock
5	6	testttherat	0	test	1	testttherat
6	7	8rating	8	ratingtest2	0	8rating

o24

Now if we want to link books to their page with slug, we need do this:

The URL segment we use must be slug so for that we do this:

`\book_store\books\urls.py`

```
urlpatterns = [
    path('', views.index, name='index'),
    path("<slug:slug>", views.book_page, name="book_page")
]
```

And so, we need change id to slug in views (slug=slug means when the slug we enter is a slug in database then give it)

`\book_store\books\views.py`

```
def book_page(request, slug):
    book = get_object_or_404(Book, slug=slug)
    ...
```

And if you remember in `get_absolute_url` we use id, so we change this to

`\book_store\books\models.py`

```
class Book(models.Model):
    ...
    def get_absolute_url(self):
        return reverse("book_page", args=[self.slug])
```

we can also do something to have better performance how by add indexing to slug model(it doesn't need migration)

```
class Book(models.Model):
    ...
    slug = models.SlugField(default="", null=False, db_index=True)
```

Output:

```
<body>
  <ul>
    <a href="/test-1-book">
      <li>test 1 book rating: 5</li>
    </a>
    <a href="/mystory">
      <li>myStory rating: 1</li>
    </a>
    <a href="/hello">
      <li>Hello rating: 2</li>
    </a>
    <a href="/bioshock">
      <li>Bioshock rating: 5</li>
    </a>
    <a href="/testttherat">
      <li>testttherat rating: 0</li>
    </a>
    <a href="/8rating">
      <li>8rating rating: 8</li>
    </a>
  </ul>
</body>
```

o25

now also we can add total number of books and avg rating so let's do this
so, in index function we can do this.

`\book_store\books\views.py`

```
def index(request):  
    books = Book.objects.all()  
    num_books = books.count()
```

we use the book var not create a query because it's too bad for performance
and for avg or like this (min and max) we need import this:

```
from django.db.models import Avg
```

then we say which models needs avg

```
def index(request):  
    books = Book.objects.all()  
    num_books = books.count()  
    avg_rating = books.aggregate(Avg('rating'))
```

now we passed these to context dictionary

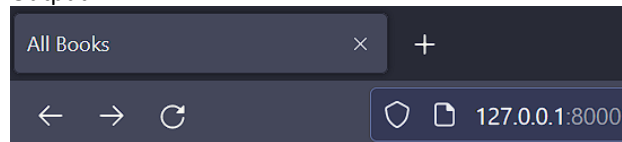
```
def index(request):  
    ...  
    return render(request, 'book_outlet/index.html', {  
        "books": books,  
        "total_count": num_books,  
        "avg_rating": avg_rating  
    })
```

And now we can add these var to index.html file

`\book_store\books\templates\book_outlet\index.html`

```
{% block content %}  
...  
<p>total number of books: {{ total_count }}</p>  
<p>avg rating : {{avg_rating }}</p>  
{% endblock %}
```

Output:



- [test 1 book rating: 5](#)
- [myStory rating: 1](#)
- [Hello rating: 2](#)
- [Bioshock rating: 5](#)
- [testtherat rating: 0](#)
- [8rating rating: 8](#)

total number of books: 6

avg rating : {'rating__avg': 3.5}

But as you see the avg give us a dictionary for get just value we can do this:

```
{% block content %}  
...  
<p>avg rating : {{avg_rating.rating__avg }}</p>  
{% endblock %}
```

Now its just return the value



- [test 1 book rating: 5](#)
- [myStory rating: 1](#)
- [Hello rating: 2](#)
- [Bioshock rating: 5](#)
- [testtherat rating: 0](#)
- [8rating rating: 8](#)

total number of books: 6

avg rating : 3.5

Also, we can order these books like by title

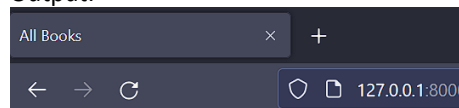
`\book_store\books\views.py`

```
def index(request):  
    books = Book.objects.all().order_by("title")
```

its order the books with title by A to Z and if we want to do this by Z to A, we can do this:

```
books = Book.objects.all().order_by("-title")
```

Output:



- [testtherat rating: 0](#)
- [test 1 book rating: 5](#)
- [myStory rating: 1](#)
- [Hello rating: 2](#)
- [Bioshock rating: 5](#)
- [8rating rating: 8](#)

total number of books: 6

avg rating : 3.5

ADMIN

oo2

Now as you know if we want to add a book we must go in shell and insert the data but if we have interface for that it's too better. So, we can create some but django for this has a ready template.

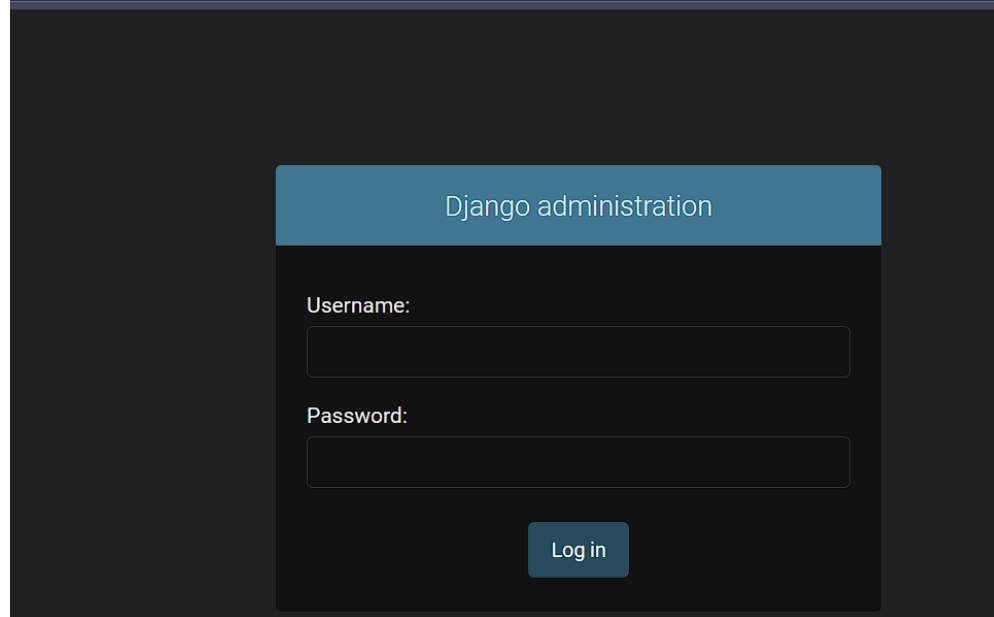
Now if you remember in url.py file we have a built in URL called /admin

`\book_store\books\urls.py`

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

Now if we enter this address what happen?

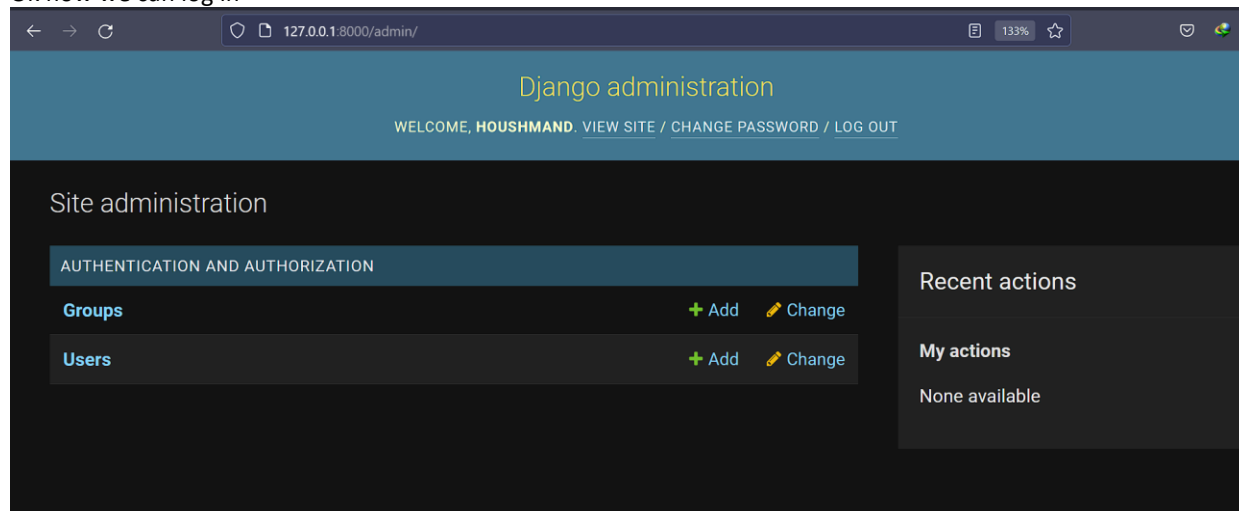
`127.0.0.1:8000/admin/login/?next=/admin/`



Ok but how we log in we don't have username or password, so we create one

```
python .\manage.py createsuperuser
Username (leave blank to use 'amir'): houshmand
Email address: test@gmail.com
Password: *
Password (again): *
Superuser created successfully.
*You can't see when you type your password.
```

Ok now we can log in

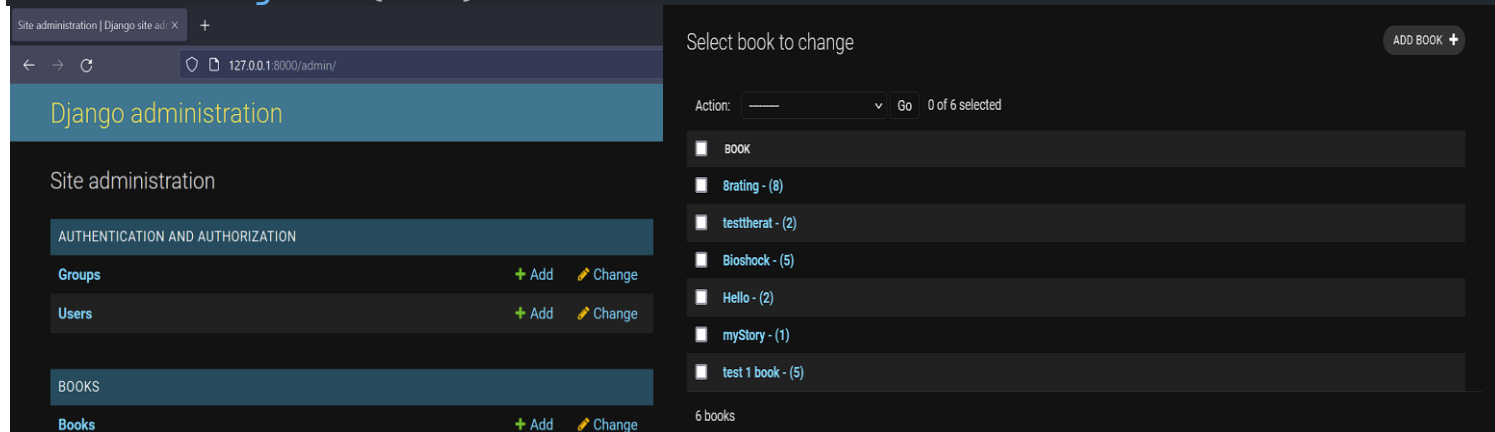


oo3

now if we want to add book with admin panel, we can do it very simple. at first we need import our models next we need register the books models like this:

```
\book_store\books\admin.py
```

```
from django.contrib import admin
from .models import Book
admin.site.register(Book)
```



And now we can delete or add or edit the data and see books. Now as you know the slug field fill automatically but if in the admin panel, we want to add a book we need fill the slug but after we enter something it overwritten anyways.

The screenshot shows the 'Add book' form. It has fields for 'Title' (filled with 'admin book'), 'Rating' (set to 3), 'Another' (filled with 'hhh'), and 'Is best seller' (checked). The 'Slug' field is empty and has a red error message 'This field is required.' below it. At the bottom, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

So, we need customize this panel.

oo4

so now for doing this we can set blank = true in slug field and then its work with empty data (as you are now it's never going to be empty) but if we want its never show up, we can set editable = false like this:

```
\book_store\books\models.py
```

```
class Book(models.Model):
    ...
    slug = models.SlugField(default="", null=False,
                           db_index=True, editable=False, blank=True)
```

now if we go in admin panel the slug field is not showing up

The screenshot shows the 'Add book' form. It has fields for 'Title' (empty), 'Rating' (set to 3), and 'Another' (empty). The 'Is best seller' checkbox is checked. At the bottom, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'. The 'Slug' field is not visible.

oo5

also, we can do a better way to do this. We can add a class in admin.py file and add var for doing this like this:

\book_store\books\admin.py

```
class BookAdmin(admin.ModelAdmin):  
    readonly_fields = ("slug",)
```

now this field get our field to set it to read only like slug (it's tuple so we add comma at the end)

when we register our models, we need to add this class like this:

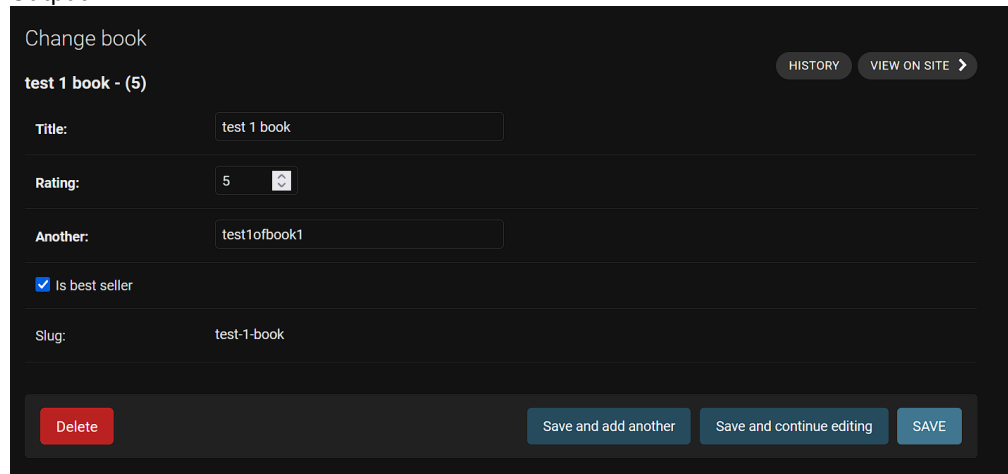
```
admin.site.register(Book, BookAdmin)
```

now we can delete the read only field in slug models

\book_store\books\models.py

```
class Book(models.Model):  
    ...  
    slug = models.SlugField(default="", null=False,  
                           db_index=True, blank=True)
```

Output:



now if we write the title the slug creates live, and we see it for do this we need do this

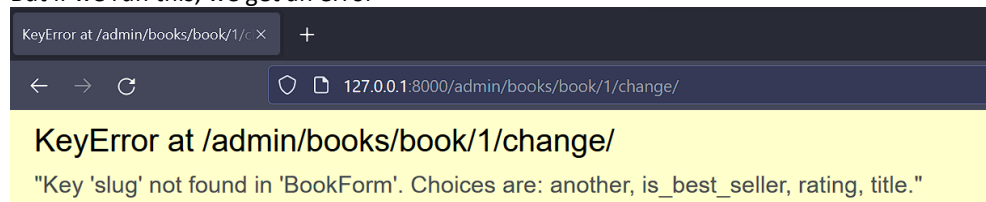
add a new var for this like this:

\book_store\books\admin.py

```
class BookAdmin(admin.ModelAdmin):  
    readonly_fields = ("slug",)  
    prepopulated_fields = {"slug": ("title",)}
```

it's a dict and we say we need ore populate which field and fill it with what field.

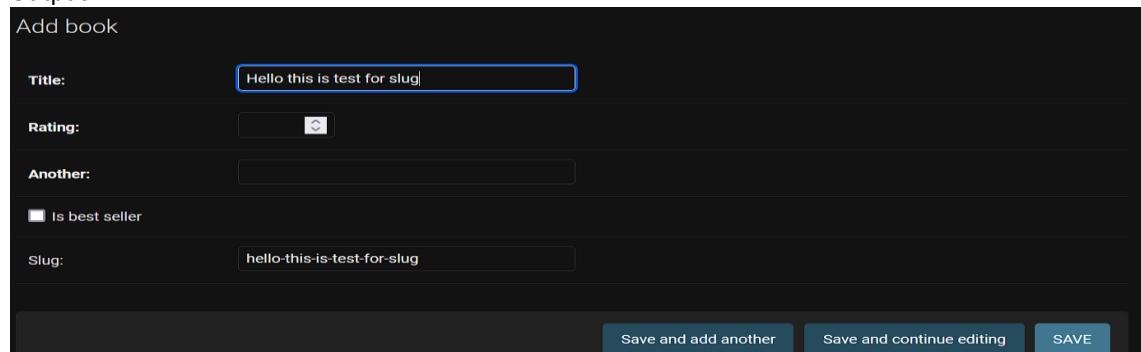
But if we run this, we get an error



As you see slug is not show because it read only so we can delete read only part

```
class BookAdmin(admin.ModelAdmin):  
    prepopulated_fields = {"slug": ("title",)}
```

Output:



As you now user now can edit this field but after save its overwritten so we can delete the save def

\book_store\books\models.py

```
class Book(models.Model):
    # def save(self, *args, **kwargs):
    #     self.slug = slugify(self.title)
    #     super().save(*args, **kwargs)
```

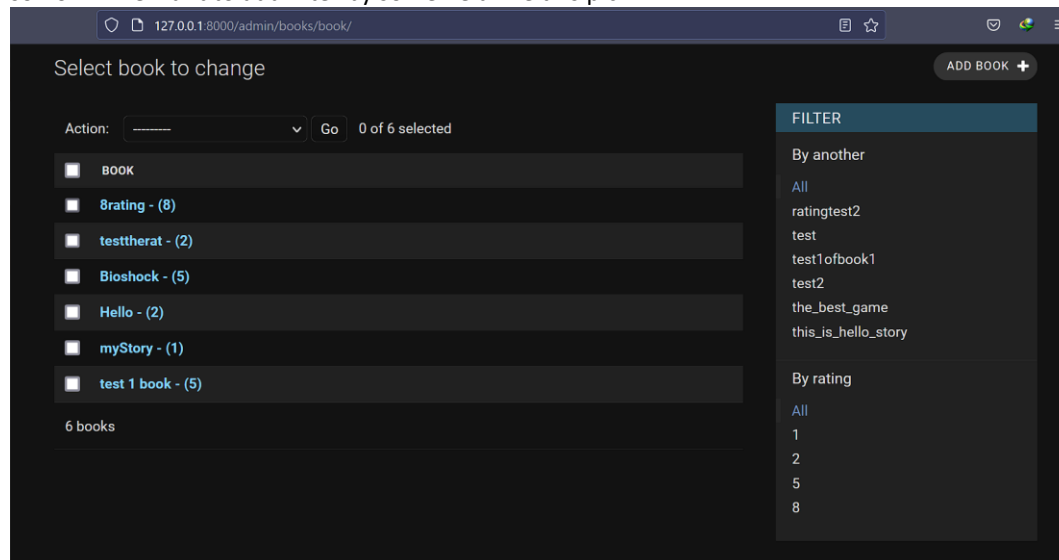
But now if user edit the slug and its turn to unvalidated field django can control this.

Enter a valid "slug" consisting of letters, numbers, underscores or hyphens.

Slug:

006

so now if we want to add filter by some field like this pic:



We need add a list of filter field like this:

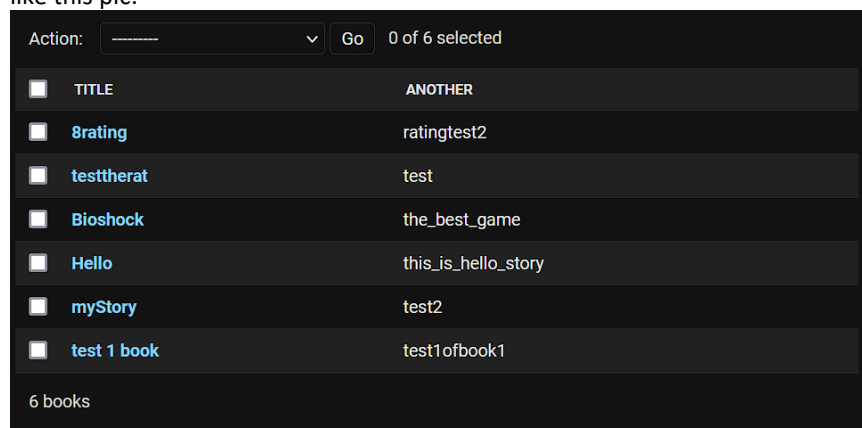
\book_store\books\admin.py

```
class BookAdmin(admin.ModelAdmin):
    ...
    list_filter = ("another", "rating")
```

now its work and we get this filter part

but we can add some columns for another field like best seller or ...

like this pic:



\book_store\books\models.py

For this we need create a list for this named list display

```
class BookAdmin(admin.ModelAdmin):
    ...
    list_display = ("title", "another")
```

now we had done for admin panel.

https://github.com/houshmand-2005/hash_neco/tree/Admin_07

08

RELATIONSHIPS

oo3 (one-to-many)

So, as you now field may have a connection with another field, like 'another' (author but as you are now, I write it uncorrected so let's keep with this)

Like we want to add first name and last name for this, and we also want use in different table or something

So, we create a new class for this field

```
class Another(models.Model): # Author
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
```

now if we want to create a connection with Book class, we need add a foreign key field. On foreign key we need say name of the class we want to connection with, so we type Another and, we need add something else, if we delete another data like first name what happen in the book models for that we have to many options like CASCADE this field delete data from another and from Book class, so the data is completely delete. Also, we add null to true to we can migrate (because the database doesn't have any data now for Another class)

```
class Book(models.Model):
    ...
    another = models.ForeignKey(Another, on_delete=models.CASCADE, null=True)
```

ok now we can make migrations:

```
python .\manage.py makemigrations
```

but we get error because we already have some another data as you remember

so django can't apply these changes so we have some few ways because the database must design at first. Now for this we can easily delete the data.

```
python .\manage.py shell
```

```
>>> from books.models import Book
```

```
>>> Book.objects.all().delete()
```

```
(6, {'books.Book': 6})
```

```
>>> exit()
```

```
python .\manage.py makemigrations
```

```
...
```

```
python .\manage.py migrate
```

Operations to perform:

Apply all migrations: admin, auth, books, contenttypes, sessions

Running migrations:

Applying books.0004_another_alter_book_slug_alter_book_another... OK

now we add the relationships between Book and Another class

oo4

now if we want to add a new book, we can use shell and admin panel but for now we work with shell

so, we open shell

```
>>>python .\manage.py shell
```

```
>>>from books.models import Book, Another
```

and we create a new another field

```
>>> houshmand = Another(first_name="amir mohammad", last_name="houshmand")
```

```
>>>houshmand.save()
```

```
>>>Another.objects.all()[0].first_name
```

```
'amir mohammad'
```

And now we create a book and connect another field to Another table field like this

```
>>>testbook1 = Book(title="new test book 1", rating=4, another=houshmand, is_best_seller=1, slug="new-test-book-1")
```

Connect as you see for connect we just say the name of variable of another(houshmand) and because we delete the save method for slug new we need handmade this(if we use admin panel its ok).

```
>>> testbook1.save()
```

```
>>> Book.objects.all()
```

```
<QuerySet [ <Book: new test book 1 - (4)> ]>
```

now if we want to see another data of this book, we can do this:

```
>>>get_another_tbook1 = Book.objects.get(title="new test book 1")
```

we store the result of book in a new variable and then we get another field

As you see django goes in 'Another' table

```
>>> get_another_tbook1.another.first_name
'amir mohammad'
>>> get_another_tbook1.another.last_name
'houshmand'
```

Table: books_book						
id	title	rating	is_best_seller	slug	another_id	
Filter	Filter	Filter	Filter	Filter	Filter	
1	1 new test book 1	4	1	new-test-book-1	1	

Table: books_another		
id	first_name	last_name
Filter	Filter	Filter
1	1 amir mohammad	houshmand

As you see django connect them by id

oo5

ok now if we need find all book from another field or find another by a book, we can do this:

```
python .\manage.py shell
```

```
>>> from books.models import Another, Book
```

And now we create a variable for all books by another field

```
>>> books_by_houshmand = Book.objects.filter(another__last_name="houshmand")
```

```
>>> books_by_houshmand
```

```
<QuerySet [<Book: new test book 1 - (4)>]>
```

Now for found books for another field we do this:

```
>>> houshmand = Another.objects.get(first_name="amir mohammad")
```

```
>>> houshmand
```

```
<Another: Another object (1)>
```

```
>>> houshmand.book_set
```

```
<django.db.models.fields.related_descriptors.create_reverse_many_to_one_manager.<locals>.RelatedManager object at 0x0000023B664E0F70>
```

```
>>> houshmand.book_set.all()
```

```
<QuerySet [<Book: new test book 1 - (4)>]>
```

For this as you see we use _set for find it by we have another way

We can add a related name dor findit byt this name like

```
class Book(models.Model):
    ...
    another = models.ForeignKey(
        Another, on_delete=models.CASCADE, null=True, related_name="books")
```

Because we change this field, we need do migration

```
python .\manage.py makemigrations
```

```
python .\manage.py migrate
```

now we have access to find it by books

```
>>> houshmand.books.all()
```

```
<QuerySet [<Book: new test book 1 - (4)>]>
```

Now also we can add filter

```
>>> houshmand.books.get(rating__gt=3)
```

```
<Book: new test book 1 - (4)>
```

oo6

now if want to add this another table to admin panel we can added by this command

```
\book_store\books\admin.py
```

```
from .models import Book, Another
admin.site.register(Another)
```

output:

<div>Select book to change</div> <div>Action: <input type="text"/> Go 0 of 1 selected</div> <table><thead><tr><th><input type="checkbox"/></th><th>TITLE</th><th>ANOTHER</th></tr></thead><tbody><tr><td><input type="checkbox"/></td><td>new test book 1</td><td>Another object (1)</td></tr></tbody></table> <div>1 book</div>	<input type="checkbox"/>	TITLE	ANOTHER	<input type="checkbox"/>	new test book 1	Another object (1)	<div>Select another to change</div> <div>Action: <input type="text"/> Go 0 of 1 selected</div> <table><tbody><tr><td><input type="checkbox"/></td><td>ANOTHER</td></tr><tr><td><input type="checkbox"/></td><td>Another object (1)</td></tr></tbody></table> <div>1 another</div>	<input type="checkbox"/>	ANOTHER	<input type="checkbox"/>	Another object (1)
<input type="checkbox"/>	TITLE	ANOTHER									
<input type="checkbox"/>	new test book 1	Another object (1)									
<input type="checkbox"/>	ANOTHER										
<input type="checkbox"/>	Another object (1)										

```
\book_store\books\models.py
```

```
class Another(models.Model): # Author
    def full_name(self):
        return f'{self.first_name} {self.last_name}'
    def __str__(self):
        return self.full_name()
```

why we just didn't add this code to `__str__` because when we code like this we can access in template and more ...
output:

Select another to change		Action: <input type="text"/> Go 0 of 1 selected
Action: <input type="text"/> Go 0 of 1 selected		<input type="checkbox"/> TITLE ANOTHER
<input type="checkbox"/> ANOTHER	<input type="checkbox"/> new test book 1	amir mohammad houshmand
<input type="checkbox"/> amir mohammad houshmand	1 book	

oo7 (one-to-one)

now if we want to add a relation with one-to-one like for another field, we can do this

```
class Address(models.Model):
    street = models.CharField(max_length=100)
    city = models.CharField(max_length=100)
    postal_code = models.CharField(max_length=10)
```

as you see we create a class for another field address
so now how we connect this class to another class

```
class Another(models.Model): # Author
    address = models.OneToOneField(
        "Address", on_delete=models.CASCADE, null=True)
```

Now for one-to-one relation we didn't use foreign key we use one to one field. And we set null to true for older another field.

And now we run the migrations

```
python .\manage.py makemigrations
```

```
python .\manage.py migrate
```

oo8

ok now let's add some address to another field by shell

```
python .\manage.py shell
```

```
>>> from books.models import Another, Address, Book
```

And now add new address

```
>>> addr1 = Address(street="test street num 1", postal_code="2534", city="london")
```

```
>>> addr2 = Address(street="test street num 2", postal_code="95433", city="mashhad")
```

Ok now let's save them

```
>>> addr1.save()
```

```
>>> addr2.save()
```

Ok but now if we want to add an address for another field, we need do this:

```
>>> houshmand = Another.objects.get(last_name="houshmand")
```

```
>>> houshmand.address
```

```
>>> houshmand.address = addr1
```

```
>>> houshmand.save()
```

As you see we at first query another field and then we add the addr1 to this field.

```
>>> houshmand.address
```

```
<Address: Address object (1)>
```

```
>>> houshmand.address.street
```

```
'test street num 1'
```

And now also we can get another field by address like this:

```
>>> Address.objects.all()
```

```
<QuerySet [ <Address: Address object (1)>, <Address: Address object (2)> ]>
```

```
>>> Address.objects.all()[0].another
```

```
<Another: amir mohammad houshmand>
```

Table: books_address			
	id	street	city
	Filter	Filter	Filter
1	1	test street num 1	london
2	2	test street num 2	mashhad

Table: books_another			
	id	first_name	last_name
	Filter	Filter	Filter
1	1	amir mohammad	houshmand

oo9

now for show this Address table in admin panel we add it like this:

\book_store\books\admin.py

```
from .models import Book, Another, Address
admin.site.register(Address)
```

output:

Action: Go 0 of 2 selected

☐ ADDRESS

☐ Address object (2)

☐ Address object (1)

2 addresss

But we need fix the how names are show

\book_store\books\models.py

```
class Address(models.Model):
    ...
    def __str__(self):
        return f"{self.street}, {self.city}, {self.postal_code}"
```

output:

Action: Go 0 of 2 selected

☐ ADDRESS

☐ test street num 2, mashhad, 95433

☐ test street num 1, london, 2534

2 addresss

Now as you see in this picture:

BOOKS

Addresses + Add

Others + Add

Books + Add

It is not very good name for Address field so we can change it!

```
class Address(models.Model):
    ...
    class Meta:
        verbose_name_plural = ("Addresses Entries")
```

now in Address class we can create another class named meta and now we set a variable for that.

BOOKS

Addresses Entries + Add

Others + Add

Books + Add

o10 (many-to-many)

And now many to many relationships. we can add a country table to store which book publish in which country so for this we create a new model

```
class Country(models.Model):
    name = models.CharField(max_length=80)
    code = models.CharField(max_length=2)
```

now in our book model we need add it to

```
class Book(models.Model):
    publish_country = models.ManyToManyField(Country)
```

and as you see we don't enter on delete parameter you now why? Because in many to many django create a new table to store it not on Book not in Country and so if it deletes, its delete from all models.

Now we need do migrations

```
python .\manage.py makemigrations
python .\manage.py migrate
```

o11

Now let's create a country and add the relationships

```
python .\manage.py shell
>>> from books.models import Book, Country
>>> Book.objects.all()
<QuerySet [(<Book: new test book 1 - (4)>)]>
>>> ntb = Book.objects.all()[0]
>>> ntb.publish_country
<django.db.models.fields.related_descriptors.create_forward_many_to_many_manager.<locals>.ManyRelatedManager object at 0x0000026A48072380>
>>> ntb.publish_country.all()
<QuerySet []>
```

Now as you see we don't have any country data so let's add

```
>>> usa = Country(name="usa", code="us")
>>> usa.save()
```

And now for add the relationships we can't use equal (=) because this isn't for many to many for this we use add () like this

```
>>> ntb.publish_country.add(usa)
>>> ntb.publish_country.all()
<QuerySet [(<Country: Country object (1)>)]>
>>> ntb.publish_country.filter(code="us")
<QuerySet [(<Country: Country object (1)>)]>
```

Also, we can do this as reverse

```
>>> Country.objects.all()
<QuerySet [(<Country: Country object (1)>)]>
>>> usa_co = Country.objects.all()[0]
>>> usa_co.book_set.all()
<QuerySet [(<Book: new test book 1 - (4)>)]>
```

o12

so, for add the country table to admin panel like before we add it

`\book_store\books\admin.py`

```
from .models import Book, Another, Address, Country
admin.site.register(Country)
```

output:

Action:	<input type="text" value="-----"/>	<input type="button" value="Go"/>	0 of 1 selected
<input type="checkbox"/>	COUNTRY		
<input type="checkbox"/>	Country object (1)		
1 country			

Also, its better to change how its shows the name of country

```
class Country(models.Model):
    ...
    def __str__(self):
        return self.name
```

output:

Action: 0 of 1 selected

<input type="checkbox"/>	COUNTRY
<input type="checkbox"/>	usa

1 country

And when we want to add book and select the country, we can select multiples countries

Title:

Rating:

Another:

☐ Is best seller

Slug:

Publish country:

usa

+

Hold down "Control", or "Command" on a Mac, to select more than one.

(Because its many to many)

10

FORMS

oo2

So now if user wants to login or something like this, we must have Forms so let's started this by create a base project. Like this:

```
.gitignore
db.sqlite3
manage.py

feedback
├── asgi.py
├── settings.py
├── urls.py
├── wsgi.py
├── __init__.py
├── __pycache__
│   ├── settings.cpython-310.pyc
│   ├── urls.cpython-310.pyc
│   ├── wsgi.cpython-310.pyc
│   └── __init__.cpython-310.pyc
└── reviews
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    ├── views.py
    ├── __init__.py
    ├── migrations
    │   └── __init__.py
    ├── __pycache__
    │   ├── urls.cpython-310.pyc
    │   ├── views.cpython-310.pyc
    │   └── __init__.cpython-310.pyc
    └── templates
        └── reviews
            └── reviews.html
```

ok now in URL.py file of app we write this for main URL

```
\feedback\reviews\urls.py

from django.urls import path
from . import views
urlpatterns = [
    path("", views.review)
]
```

(Side note :) yes, I change my theme of IDE before I write code in vs code now is in PyCharm)

And in views file write this (for now):

```
\feedback\reviews\views.py
```

```
def review():
    pass
```

and because we add a URL file in app, we need include it

```
\feedback\feedback\urls.py
```

```
from django.urls import path, include

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


003

So now let's add a simple form. at first we need add our app in setting.py

```
\feedback\feedback\settings.py
```

```
INSTALLED_APPS = [  
    ...  
    'reviews'  
]
```

And now let's create form in html file

```
\feedback\templates\reviews\reviews.html
```

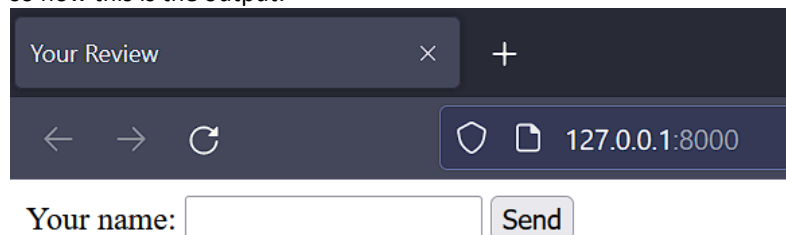
```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>Your Review</title>  
</head>  
<body>  
<form>  
    <label for="username">Your name:</label>  
    <input id="username" name="username" type="text">  
    <button>Send</button>  
</form>  
</body>  
</html>
```

The 'for' element in label is for what label is for what input. And then we add a button. But still, we can't see it because we didn't add this page to views.py so let's do it

```
\feedback\reviews\views.py
```

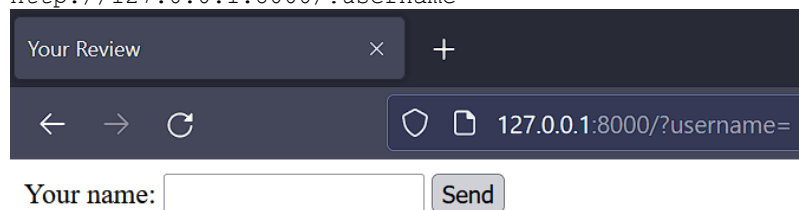
```
from django.shortcuts import render  
def review(request):  
    return render(request, "reviews/reviews.html")
```

so now this is the output:



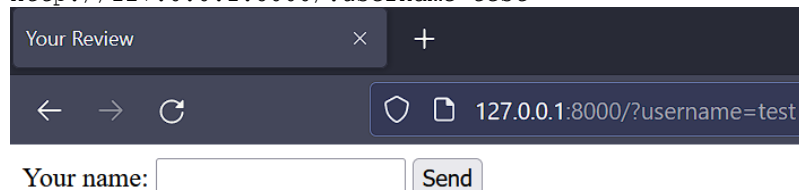
But we something we have here, and this is about the URL when we press button. When your name filed is empty and we press the button the URL is:

```
http://127.0.0.1:8000/?username=
```



But if we enter something like "test" the URL changed to

```
http://127.0.0.1:8000/?username=test
```



But what is this and why this is too important? In next part we can find out

004

type of Button tag in default is submit when we use this type its like this but if we use Button type then when we click nothing happening.

```
\feedback\templates\reviews\reviews.html
```

```
<button type="button">Send</button>
```

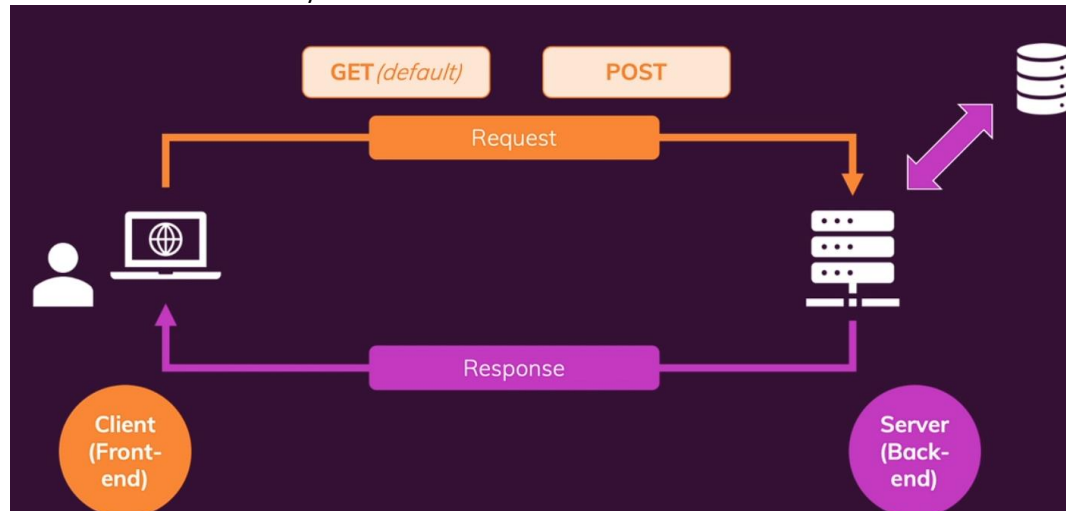
Like this:

`http://127.0.0.1:8000/` click on the button → `http://127.0.0.1:8000/`

but we want to say what is the username for login because we need the data. So, I change it to submit

```
<button type="submit">Send</button>
```

But let's see the different by GET and POST.



Ok but now when we click the button, still we use GET because this is the default but as you are now, we need Post data to submit it.

But for that we need set this form to POST how? Like this:

```
<form method="post">
```

```
...
```

```
</form>
```

And now if we press the button, we see we called it to POST

POST	http://127.0.0.1:8000/
Status	403 Forbidden ?
Version	HTTP/1.1
Transferred	2.72 KB (2.45 KB size)
Referrer Policy	same-origin

Headers	Cookies	Request	Response	Timings
Filter Request Parameters				
Form data				
username: "test"				

But we get an error when we submit it by button

403 Forbidden

Forbidden (403)

CSRF verification failed. Request aborted.

Help

Reason given for failure:
CSRF token missing.

005

this error is because we didn't load the CSRF token ok but what is CSRF token. This tag can protect server from fake request for more information please see this video (S10 E5). Ok now for add this tag we can do this:

```
<form method="post">
  {% csrf_token %}
  <label for="username">Your name:</label>
  <input id="username" name="username" type="text">
  <button type="submit">Send</button>
</form>
```

Now if we reload the page, we see a new input (this input made automatically by CSRF token and the value in this tag is random)

Output:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Your Review</title>
6 </head>
7 <body>
8 <form method="post">
9     <input type="hidden" name="csrfmiddlewaretoken" value="5At5qrbCUDfc9HEjKHnsi5nG1jiDrbNV53yIFie4vggscFq8mJwEcrJjupUXIZok">
10    <label for="username">Your name:</label>
11    <input id="username" name="username" type="text">
12    <button type="submit">Send</button>
13 </form>
14 </body>
15 </html>

```

Now we can submit the data.

006

ok now let's extract the data in input and redirect to another page. for the redirect part we can create a page for this so

\feedback\templates\reviews\thank_you.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Thank you!</h1>
</body>
</html>

```

And create a URL for it

\feedback\reviews\urls.py

```

urlpatterns = [
    ...
    path("thank_you", views.thank_you)
]

```

ok now let's extract the data

\feedback\reviews\views.py

```

from django.http import HttpResponseRedirect
...
def review(request):
    if request.method == "POST":
        entered_username = request.POST['username']
        print(entered_username)
        return HttpResponseRedirect("thank_you")

    return render(request, "reviews/reviews.html")

```

ok at first, we check the request is Post or no (because see user input the data or no)

next we save the data from username input*

\feedback\templates\reviews\reviews.html

```

* <form method="post">
    <input id="username" name="username" type="text">
</form>

```

Then we can insert that data in database but for now just print it.

And then we redirect the user to thank you URL (we did not create it for now) but if the request is not POST we just render the reviews page. now let create a thank you view

\feedback\reviews\views.py

```

def thank_you(request):
    return render(request, "reviews/thank_you.html")

```

Output:

http://127.0.0.1:8000/ → insert "houshmand" then click the button → http://127.0.0.1:8000/thank_you → and the log is:

[15/Jun/2022 19:05:17] "POST / HTTP/1.1" 302 0

Houshmand

007

ok as you now if we did not insert data and click on button, we go to thank you page, but we don't do this we must add an error for that to user understands own mistake.

Ok for this we need another if. So, if user input nothing then clicks the button has_error is set to True but if not is set to False

```
def review(request):
    if request.method == "POST":
        entered_username = request.POST['username']
        if entered_username == "":
            return render(request, "reviews/reviews.html", {
                "has_error": True
            })
        print(entered_username)
        return HttpResponseRedirect("thank_you")

    return render(request, "reviews/reviews.html", {
        "has_error": False
    })

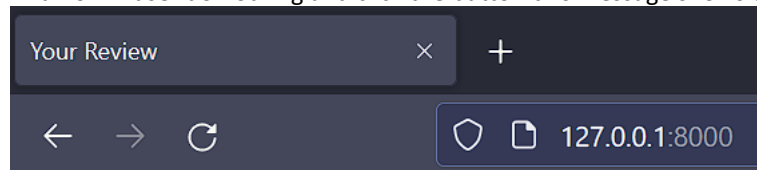
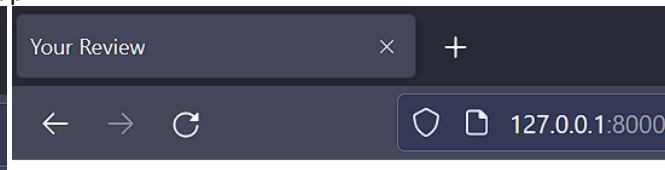
def thank_you(request):
    return render(request, "reviews/thank_you.html")
```

and in reviews page we need an If for this

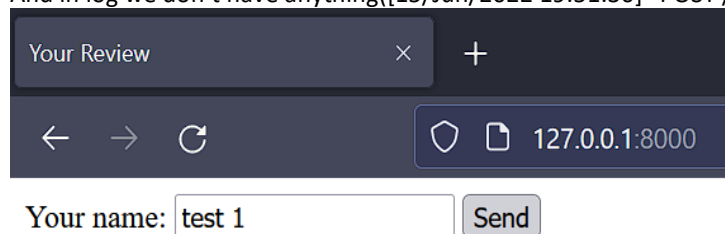
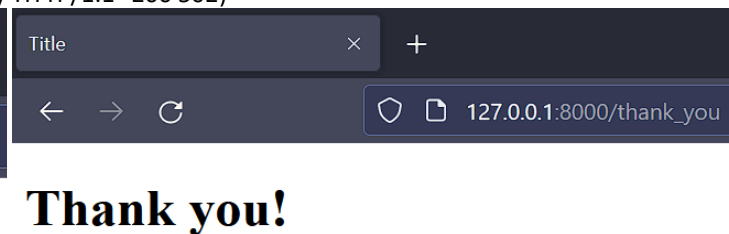
feedback\templates\reviews\reviews.html

```
<form method="post">
    {% csrf_token %}
    {% if has_error %}
        <p>This form is invalid - please enter a valid username </p>
    {% endif %}
    ...
</form>
```

And now if user do nothing and click the button this message shows up.

	
--	--

And in log we don't have anything([15/Jun/2022 19:51:30] "POST / HTTP/1.1" 200 502)

	
--	--

Log:

[15/Jun/2022 19:52:22] "GET / HTTP/1.1" 200 428

test 1

but if we have a lots of input or lots of conditions, we can use django form class, we use it in next lesson.

008

now with form class we can do this automatically. For that we need a new file named forms.py (we can change the name)

in this file we need import form django and create a class for the form we need in this from we add the parameters we need for form.

feedback\reviews\forms.py

```
from django import forms
class ReviewForm(forms.Form):
    user_name = forms.CharField()
```

just like models we enter type of inputs. Then in views.py file we need change code with this

`\feedback\reviews\views.py`

```
from .forms import ReviewForm
def review(request):
    if request.method == "POST":
        form = ReviewForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data)
            return HttpResponseRedirect("/thank_you")
    else:
        form = ReviewForm()
    return render(request, "reviews/reviews.html", {
        "form": form
    })
```

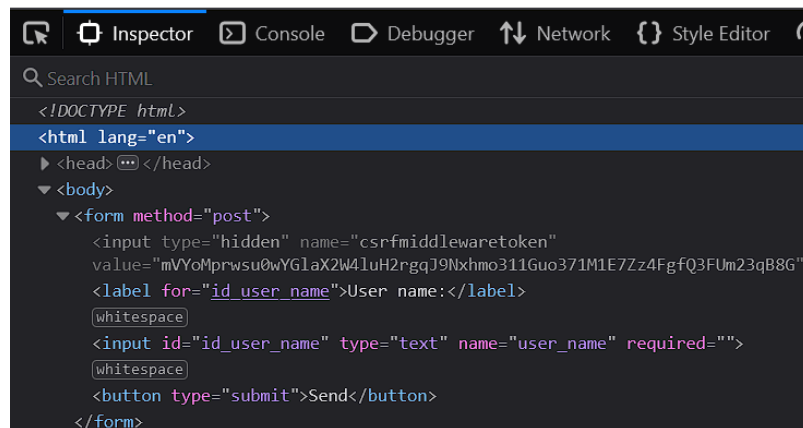
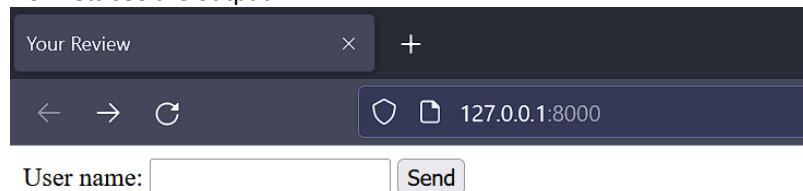
ok now if user wants to enter the data, we can find out with request. `ReviewForm(request.POST)` and we don't need to check user to for empty input because we use form and django automatically check the input with parameters in forms.py so if user just click the button and doesn't enter the data in input django gets error to the user. We check this with `form.valid()` (gets True or False value) and for get the data we use `form.cleaned_data` this gives us a dictionary.

And in reviews.html we can add the inputs automade

`\feedback\templates\reviews\reviews.html`

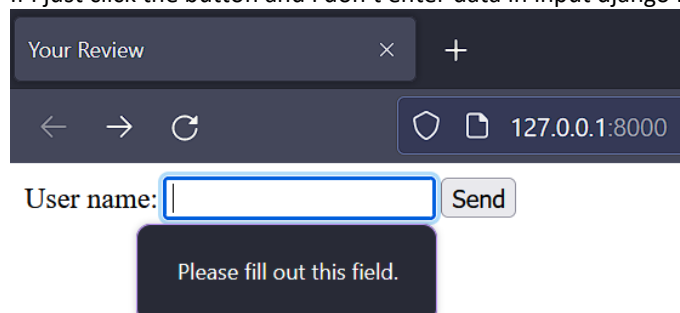
```
<body>
<form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Send</button>
</form>
</body>
```

Now lets see the output:



As you see this html code are made by django


If I just click the button and I don't enter data in input django handled this



Thank you!

A screenshot of a web browser's address bar. The address bar is dark grey. On the left, it says 'Your Review' in white text. To the right of the text are two icons: a white 'x' and a white '+' sign. Below the address bar, there is a navigation bar with three icons: a white left arrow, a white right arrow, and a white circular refresh icon. To the right of these icons is a shield icon, a document icon, and the text '127.0.0.1:8000' in white.

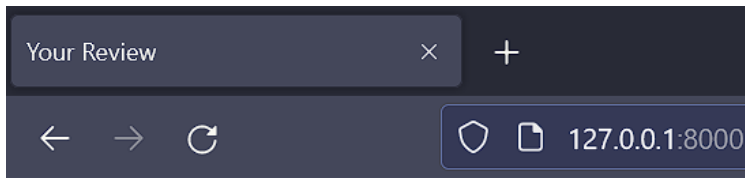
```
class ReviewForm(forms.Form):
    user name = forms.CharField(label="Your Name", max length=20)
```



Your Name:

fgsgfgdgdgdfhghghfg

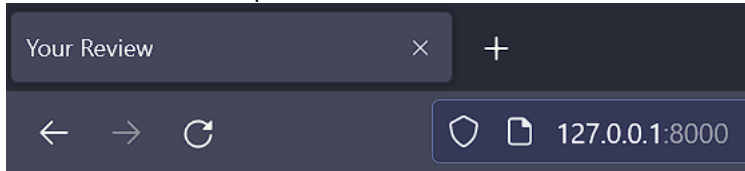
```
class ReviewForm(forms.Form):
    user_name = forms.CharField(label="Your Name", max_length=20, error_messages={
        "required": "Your name must not be empty",
        "max_length": "please enter a shorter name!"
    })
```



Your Name:

- please enter a shorter name!

As you see our message shows up.
And let's delete the required field



Your Name:

- Your name must not be empty

Also, we can set required to false if we want for this input

```
class ReviewForm(forms.Form):
    user_name = forms.CharField(label="Your Name", required=False, ...)
```

now we don't have this required validation

but I don't like required false for this input so I can set it to true but in default is set to true, so I delete this parameter.

For more knowledge for this you can visit this : <https://docs.djangoproject.com/en/4.0/ref/forms/fields/>

011

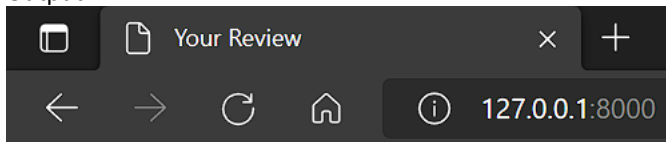
now if we want to change the how display this input we can do this(we want show the error messages a bow the input)

`\feedback\templates\reviews\reviews.html`

```
<form method="post">
    {% csrf_token %}
    {{ form.user_name.label_tag }}
    {{ form.user_name }}
    {{ form.user_name.errors }}
    <button type="submit">Send</button>
</form>
```

With this method we have acces to the different part of the form like this

Output:



Your Name:

- Your name must not be empty

Now as you see in the different site when they have error they change some styling like add a red color or something for that we can to this:

```
{% csrf_token %}
<div class="form-control{% if form.user_name.errors %} errors{% endif %}">
    {{ form.user_name.label_tag }}
    {{ form.user_name }}
```

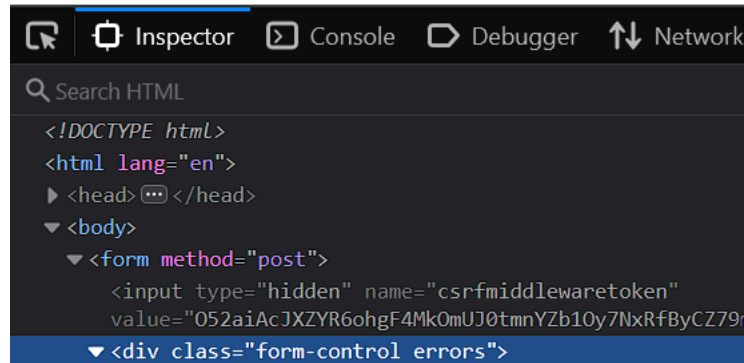
```
{{ form.user_name.errors }}
</div>
```

Now if user has error the class we'll be form-control errors like this:

Your Name:

- Your name must not be empty

Send



o12

now for add some style we can create a folder for that

\feedback\reviews\static\reviews\styles.css

```
body {
  font-family: sans-serif;
}
button {
  cursor: pointer;
  padding: 0.5rem 1.5rem;
}
button:hover, button:active {
  background-color: antiquewhite;
}
.errorlist {
  list-style: none;
  margin: 0.5rem 0;
  padding: 0;
  color: #a71212;
}
.errors label {
  color: #a71212;
}
```

some basic CSS and in reviews.html file

```
<head>
  <link rel="stylesheet" href="{% static 'reviews/styles.css' %}">
</head>
```

Now if we have error this is the output:

Your Name:

Your name must not be empty

Send

013

now as you know we want to user add his or her reviews to something, so we need more input like the review text and rating so for this we add those filed.

`\feedback\reviews\forms.py`

```
class ReviewForm(forms.Form):  
  
    ...  
    review_text = forms.CharField(label="Your Feedback",  
    widget=forms.Textarea, max_length=200)  
    rating = forms.IntegerField(label="Your Rating", min_value=1,  
    max_value=5)
```

ok for the review text we need some bigger space for long text like text area for this we add widget and set it to forms.textarea.

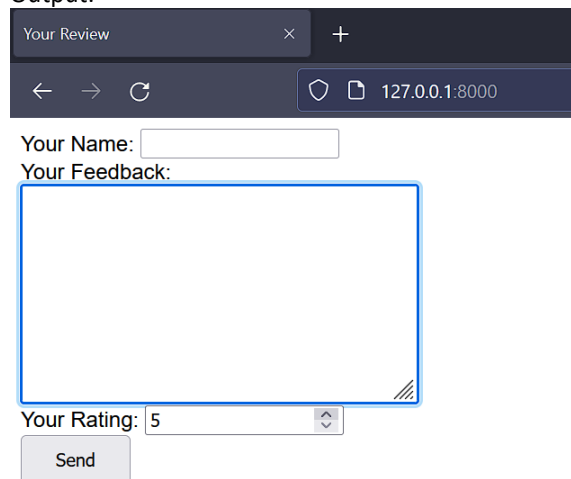
now if we save and look the output we did not see any changes because we dont add these input in html (beacse we render the form manually).

We can copy what we write for name input like this:

`\feedback\templates\reviews\reviews.html`

```
<form method="post">  
    {% csrf_token %}  
    <div class="form-control{% if form.user_name.errors %} errors{% endif %}">  
        {{ form.user_name.label_tag }}  
        {{ form.user_name }}  
        {{ form.user_name.errors }}  
    </div>  
    <div class="form-control{% if form.review_text.errors %} errors{% endif %}">  
        {{ form.review_text.label_tag }}<br>  
        {{ form.review_text }}  
        {{ form.review_text.errors }}  
    </div>  
    <div class="form-control{% if form.rating.errors %} errors{% endif %}">  
        {{ form.rating.label_tag }}  
        {{ form.rating }}  
        {{ form.rating.errors }}  
    </div>  
    <button type="submit">Send</button>  
</form>
```

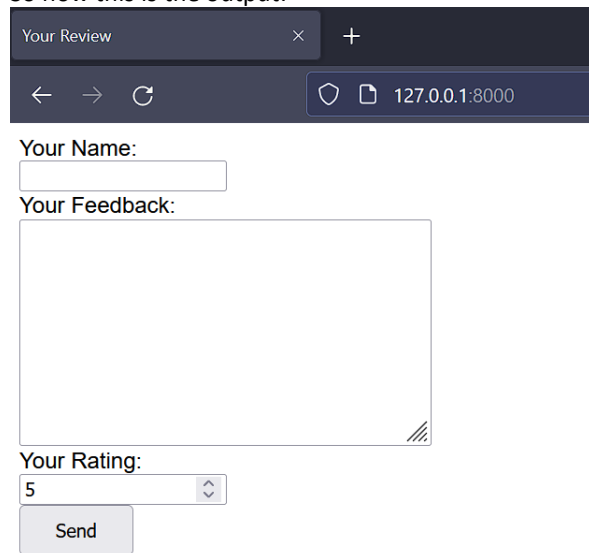
Output:



But as you see we copy over and over again and this is not good so we can do this with for loop like this:

```
<form method="post">  
    {% csrf_token %}  
    {% for field in form %}  
        <div class="form-control{% if field.errors %} errors{% endif %}">  
            {{ field.label_tag }}<br>  
            {{ field }}  
            {{ field.errors }}  
        </div>  
    {% endfor %}  
    <button type="submit">Send</button>  
</form>
```

So now this is the output:



And if we input some data this is the log:

[19/Jun/2022 19:50:59] "GET / HTTP/1.1" 200 1009

{'user_name': 'test', 'review_text': 'hello this is a test', 'rating': 3}

o14

now we want store the data in database not just print it, for this we need to create a tabel with models.py like this:

`\feedback\reviews\models.py`

```
class Review(models.Model):
    user_name = models.CharField(max_length=20)
    review_text = models.TextField()
    rating = models.IntegerField()
```

now we need to make migrations

`python .\manage.py makemigrations`

`python .\manage.py migrate`

now for saving data we nneed something in views.py file

`\feedback\reviews\views.py`

```
from .models import Review
def review(request):
    if request.method == "POST":
        form = ReviewForm(request.POST)
        if form.is_valid():
            review_get = Review(user_name=form.cleaned_data['user_name'],
                               review_text=form.cleaned_data['review_text'],
                               rating=form.cleaned_data['rating'])
            review_get.save()
            return HttpResponseRedirect("/thank_you")
```

At first we need import our models then we sve it like this.

`Modelsfield = from.cleaned_data['here_put_the_name_of_form_field']`

Then we need save the database now lets test it:

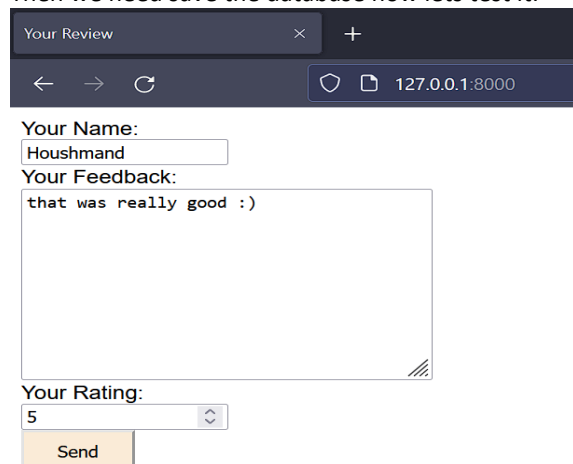


Table: reviews_review				
	id	user_name	review_text	rating
	Filter	Filter	Filter	Filter
1	1	Houshmand	that was really good :)	5

o15

we have another way to create a form in this way we create it by models. This is like in admin panel and django create it automatically
`\feedback\reviews\forms.py`

```
from .models import Review
class ReviewForm(forms.ModelForm):
    class Meta:
        model = Review
        fields = '__all__'
```

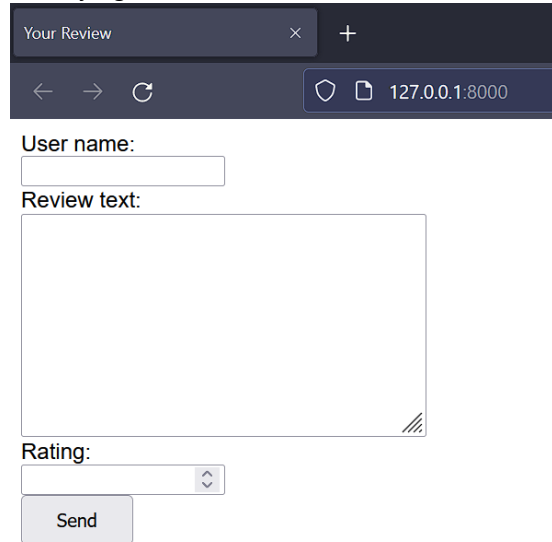
like this we create it auto by django in fields we select which field shows up. Also, we can create a list of fields for this if we want

```
fields = ['user_name', 'review_text', 'rating']
```

also we can exclude fields like this:

```
exclude = ['something']
```

then django create it like this:



The screenshot shows a web browser window with the title 'Your Review'. The browser's address bar shows '127.0.0.1:8000'. The form contains three input fields: 'User name' (a single-line text box), 'Review text' (a large multi-line text area), and 'Rating' (a dropdown menu). A 'Send' button is located at the bottom of the form.

But as you see the label are gone because it's made by django but like before we can edit it.

o16

for customization we have some key like `error_messages` and labels like this:

```
class ReviewForm(forms.ModelForm):
    class Meta:
        ...
        labels = {
            "user_name": "Your Name",
            "review_text": "Your Feedback",
            "rating": "Your Rating"
        }
```

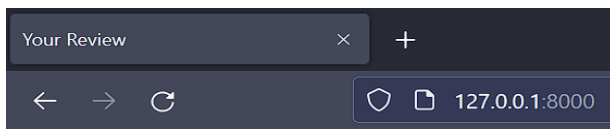
"Name of field": "label"

And for error messages:

```
class ReviewForm(forms.ModelForm):
    class Meta:
        ...
        error_messages = {
            "user_name": {
                "required": "Your name must not be empty",
                "max_length": "please enter a shorter name!"
            }
        }
```

```
error_messages= {
    "Which_field": {
        "Some_error": "text of error"
    }
}
```

Output:



Your Name:

Your name must not be empty

Your Feedback:

Your Rating:

Send

o17

when we use model forms, we can save data with other way

we can just called form.save()

\feedback\reviews\views.py

```
def review(request):
    if request.method == "POST":
        form = ReviewForm(request.POST)
        if form.is_valid():
            form.save()
```

with this way also we can update a review if we need. If you want, you can do like this:

```
def review(request):
    if request.method == "POST":
        existing_model = Review.objects.get(pk=1)
        form = ReviewForm(request.POST, instance=existing_model)
        if form.is_valid():
            form.save()
```

with this we can update a field. But we don't need it for now, so I deleted

now lets test it:



Your Name:

Your Feedback:

Your Rating:

Send

Database Structure Browse Data Edit Pragmas				
Table: reviews_review				
	id	user_name	review_text	rating
	Filter	Filter	Filter	Filter
1	1	batman	I'm batman	1

o18

as you see in views.py file for 'get' and 'post' we have different commands and we use if to separate them from each other. but we have another way, the class based views we have a lots of class baesd views but for now we use one of them.(in next section we learn more about it).

At first, we need import view from Django.view (for now, because we have lots of class based view).

```
from django.views import View
```

then we create class for this view and create get function and post function like this:

```
class ReviewView(View):
    def get(self, request):
        ...
    def post(self, request):
        ...
```

Now for each requests we can add some code like simple view

```
class ReviewView(View):
    def get(self, request):
        form = ReviewForm()
        return render(request, "reviews/reviews.html", {
            "form": form
        })
    def post(self, request):
        form = ReviewForm(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect("/thank_you")
        return render(request, "reviews/reviews.html", {
            "form": form
        })
```

but with class based view we need change something in URLs

`\feedback\reviews\urls.py`

```
urlpatterns = [
    path("", views.ReviewView.as_view()),
]
```

for classed based view : `path("url", views.NameOfViews.as_view())`

now if we run it its like before. But why we do this?

Because now it's a little clearer but class based views has another ability for now we just test it and in next part we learned more about it.

11

CLASS VIEWS

oo2

Ok for start lets create a base.html file like before, why? Because later we add more html file.

\feedback\templates\reviews\base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="{% static "reviews/styles.css" %}">
</head>
<body>
{% block content %}
{% endblock %}
</body>
</html>
```

Then in review html file and thank you page let's use this.

\feedback\templates\reviews\reviews.html

```
{% extends "reviews/base.html" %}
{% block title %}Your review{% endblock %}
{% block content %}
<form method="post">
    {% csrf_token %}
    {% for field in form %}
        <div class="form-control{% if field.errors %} errors{% endif %}">
            {{ field.label_tag }}<br>
            {{ field }}
            {{ field.errors }}
        </div>
    {% endfor %}
    <button type="submit">Send</button>
</form>
{% endblock %}
```

\feedback\templates\reviews\thank_you.html

```
{% extends "reviews/base.html" %}
{% block title %}thank you{% endblock %}
{% block content %}
<h1>Thank you!</h1>
{% endblock %}
```

now we can add a page for shows all of reviews in database and more with class views.

oo3

now before the reviews project lets change thank _you view to class base view (there is no problem with before just for learn class views).

\feedback\reviews\views.py

```
class ThankYou(View):
    def get(self, request):
        return render(request, "reviews/thank_you.html")
```

and also, for URL:

\feedback\reviews\urls.py

```
urlpatterns = [
    path("", views.ReviewView.as_view()),
    path("thank_you", views.ThankYou.as_view())
]
```

as you know we don't need post request function for this, so for sometimes we don't need post we can use a built-in method called Template View, this class view is specific for when we just have 'get' and no post like the thank you page as you see.

At first, we need to import it

```
\feedback\reviews\views.py
```

```
from django.views.generic import TemplateView
```

```
then
class ThankYou(TemplateView):
    template_name = "reviews/thank_you.html"
```

as you see in template_name we say where the template is.

Now if we want to add some context like before we need do this:

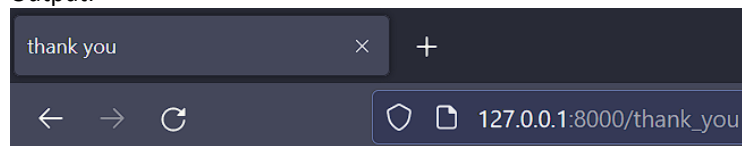
```
class ThankYou(TemplateView):
    template_name = "reviews/thank_you.html"
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["message"] = "This works!"
        return context
```

and in thank you page

```
\feedback\templates\reviews\thank_you.html
```

```
{% block content %}
<h1>Thank you!</h1>
<h3>{{ message }}</h3>
{% endblock %}
```

Output:



Thank you!

This works!

004

now let's create a list of all reviews

at first create a html file for this.

```
\feedback\templates\reviews\review_list.html
```

Then let's create a view

```
\feedback\reviews\views.py
```

```
from .models import Review
class ReviewList(TemplateView):
    template_name = "reviews/review_list.html"
    def get_context_data(self, **kwargs):
        reviews = Review.objects.all()
        context = super().get_context_data(**kwargs)
        context["reviews"] = reviews
        return context
```

like this we get all of reviews from database in context. For show this data we can do this:

```
\feedback\templates\reviews\review_list.html
```

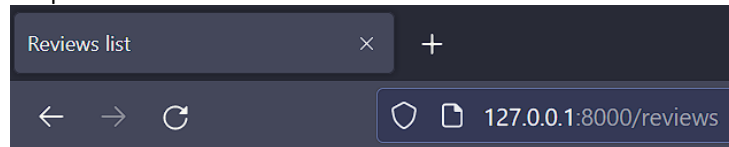
```
{% extends "reviews/base.html" %}
{% block title %}Reviews list{% endblock %}
{% block content %}
<ul>
{% for review in reviews %}
    <li>
        <p>{{ review.user_name }} (Rating : {{ review.rating }})</p>
    </li>
{% endfor %}
</ul>
{% endblock %}
```

Now we need add a URL for this view

```
\feedback\reviews\urls.py
```

```
urlpatterns = [
    ...
    path("reviews", views.ReviewList.as_view())
]
```

Output:



- test 2 (Rating : 3)
- new test (Rating : 4)
- hello (Rating : 2)
- 02 (Rating : 1)

oo5

Now for these reviews lets create a detail page to show more about review.

At first let's create a html for this page

```
\feedback\templates\reviews\detail_review.html
```

Then view

```
\feedback\reviews\views.py
```

```
class DetailReview(TemplateView):
    template_name = "reviews/detail_review.html"
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        reviews = Review.objects.get(pk=kwargs["ReviewId"])
        context["selected_review"] = reviews
        return context
```

we get id of reviews with kwargs["field_in_URL"]

then we query with this id.

```
\feedback\reviews\urls.py
```

```
urlpatterns = [
    ...
    path("reviews/<int:ReviewId>", views.DetailReview.as_view(),
        name="reviews_page")
]
```

and in html file

```
\feedback\templates\reviews\detail_review.html
```

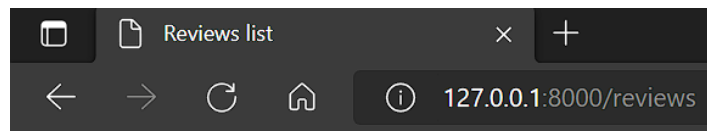
```
{% extends "reviews/base.html" %}
{% block title %}{% endblock %}
{% block content %}
    <h2>{{ selected_review.user_name }}</h2>
    <p>rating : {{ selected_review.rating }}</p>
    <p>{{ selected_review.review_text }}</p>
{% endblock %}
```

Now for add a link in list to go on detail page.

```
\feedback\templates\reviews\review_list.html
```

```
{% for review in reviews %}
    <li>
        <a href="{% url 'reviews_page' review.id %}">
            <p>{{ review.user_name }} (Rating : {{ review.rating }})</p>
        </a>
    </li>
{% endfor %}
```

Output:



- [test 2 \(Rating : 3\)](#)
- [new test \(Rating : 4\)](#)
- [hello \(Rating : 2\)](#)
- [02 \(Rating : 1\)](#)

test 2

rating : 3

some text field

006

As you see in ReviewList view we render a list of fields from database so django has a built-in tool for that.

For using this tool, we need import it

`\feedback\reviews\views.py`

```
from django.views.generic import ListView
```

then let's change the view

```
class ReviewList(ListView):  
    template_name = "reviews/review_list.html"  
    model = Review
```

as you see we have a new variable called model in this var we input the model we want to use it.

Now we are done let's test it.



But as you see we don't get anything why? Because in for loop we don't have a reviews variable we need to change this to object_list

`\feedback\templates\reviews\review_list.html`

```
{% for review in object_list %}  
...  
{% endfor %}
```

Now its working. But this name is not very good so we can change it by add context_object_name.

`\feedback\reviews\views.py`

```
class ReviewList(ListView):  
    ...  
    context_object_name = "reviews"
```

and now we can change the object_list to reviews

`\feedback\templates\reviews\review_list.html`

```
{% for review in reviews %}  
...  
{% endfor %}
```

Now its work.

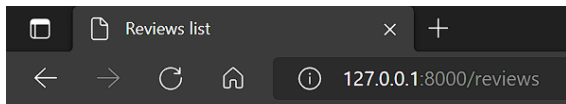
But if we want to show reviews with rating higher than 3 for this, we can do this:

`\feedback\reviews\views.py`

```
class ReviewList(ListView):  
    template_name = "reviews/review_list.html"  
    model = Review  
    context_object_name = "reviews"  
    def get_queryset(self):  
        base_query = super().get_queryset()  
        data = base_query.filter(rating__gt=3)  
        return data
```

with this way we can query some data form database.

Output:



- [new test \(Rating : 4\)](#)

But a want to show all the review so I comment this part.

oo7

now if you remember the detail review view, we can do something else for that. We have a detailview for get one item so, at first, we need import it

```
from django.views.generic import DetailView
```

then for view we can do this

```
class DetailReview(DetailView):
    template_name = "reviews/detail_review.html"
    model = Review
```

then django automatically do this for us (it's very easy)

but in URL we need change something.

```
\feedback\reviews\urls.py
```

```
urlpatterns = [
    ...
    path("reviews/<int:pk>", views.DetailReview.as_view(), name="reviews_page")
]
```

As you see we need change the Dynamic Segments (<int:here>) to pk. then django can find it.

but how django find out our name in template?

Django find it by lower case of model name or if we change it to object.

So, we change the name to review

```
\feedback\templates\reviews\detail_review.html
```

```
{% block content %}
    <h2>{{ review.user_name }}</h2>
    <p>rating : {{ review.rating }}</p>
    <p>{{ review.review_text }}</p>
{% endblock %}
```

Now its works like before.

oo9

ok as you see before we have a lot of class view and so also, we have a view for forms.

So, at first like before we need import it

```
\feedback\reviews\views.py
```

```
from django.views.generic.edit import FormView
```

now let's change the view

```
class ReviewView(FormView):
    form_class = ReviewForm
    template_name = "reviews/reviews.html"
```

as you see we have a new variable named form_class we must input this field with our form. For now, we handle the get request.

Now if user for example input an invalid data django can handle that but if this valid we have error because django doesn't know what it can do after this. So, we add another field for that

```
class ReviewView(FormView):
    ...
    success_url = "/thank_you"
```

now if user input a valid data django redirect user to thank you page

but if we go in review list page, we can't see it because django doesn't save it.

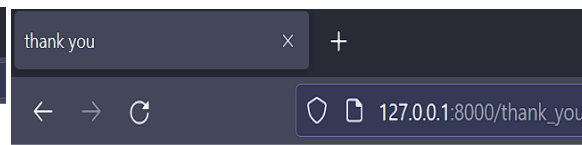
Now for save the data we can do this:

```
class ReviewView(FormView):
    form_class = ReviewForm
    template_name = "reviews/reviews.html"
    success_url = "/thank_you"
    def form_valid(self, form):
        form.save()
        return super().form_valid(form)
```

and now its works like before.

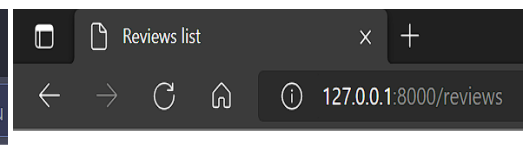
The screenshot shows a web browser window with a single tab titled 'Your review'. The address bar shows the URL '127.0.0.1:8000'. The form contains the following elements:

- Your Name:** A text input field containing 'test formview'.
- Your Feedback:** A large text area containing 'this is a form view'.
- Your Rating:** A dropdown menu showing '5'.
- Send** button.



Thank you!

This works!



o10

Also, we have another thing for form, named create form

```
from django.views.generic.edit import CreateView
```

and then

```
class ReviewView(CreateView):
    model = Review
    fields = '__all__'
```

as you see we don't even need form because django can created. But here we can't do customizations like label or error messages so, like before we can still use the `form_class = ReviewForm`

```
class ReviewView(CreateView):
    model = Review
    form_class = ReviewForm
    template_name = "reviews/reviews.html"
    success_url = "/thank_you"
```

with this way now django save the data for us too. Also, we have update view and delete view they also work like create view.

12

FILE UPLOADS

For start this part I create the basic files (some html and set a simple URL and view for show html file just this) you can download it from here: <https://github.com/academind/django-practical-guide-course-code/archive/refs/heads/file-upload-00-starting-setup.zip>

So, for upload input in html file, we need change some attributes

003

`\feedback\profiles\templates\profiles\create_profile.html`

```
<body>
  <form action="/profiles/" method="post" enctype="multipart/form-data">
    {% csrf_token %}
    <input type="file" name="image"/>
    <button>Upload!</button>
  </form>
</body>
```

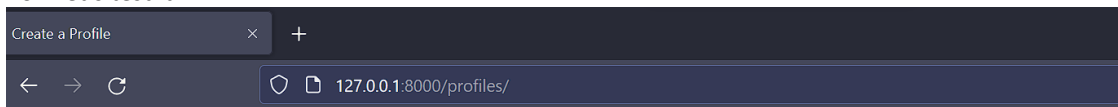
We set the action for set it to our URL and we also set method to post because we want to post some data like photo and the enctype, this value is necessary if the user will upload a file through the form. so now we are good in html lets add some more code to view

`\feedback\profiles\views.py`

```
from django.http import HttpResponseRedirect
class CreateProfileView(View):
    def get(self, request):
        return render(request, "profiles/create_profile.html")
    def post(self, request):
        print(request.FILES["image"])
        return HttpResponseRedirect("/profiles")
```

as you see we get file by `request.FILES["the name of input so for now is image"]`

now let's test it



And in terminal:

Untitled.png

As you see we get the file, but we don't save it we just print name of the file.

004

So, for store the file we can use lots of way but for now I code some basics to understand how it works. For save it we can use python commands like with open, so for doing this I create a function.

```
def store_file(file):
    with open("temp/image.png", "wb+") as dest:
        for chunk in file.chunks():
            dest.write(chunk)
```

I open a image.png with wb+ (this can help us with open binary file like images and then we can write to that file) then for each chunk of photo I write it to file and then file created.

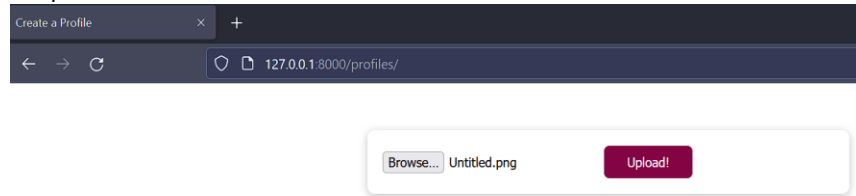
So now we can use this function

```
class CreateProfileView(View):
    ...
    def post(self, request):
        store_file(request.FILES["image"])
```

And then for save the photo we need create the directory called temp in main folder

\feedback\temp

Output:



\feedback\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a----	6/25/2022 7:33 PM	185752	image.png

so, as you see we can store the file but just with .PNG format

oo5

now we can use form for load input tag automated. So, let's create from.py file for that.

\feedback\profiles\form.py

```
from django import forms
class ProfileForm(forms.Form):
    user_image = forms.FileField()
```

now we can use this to our template

\feedback\profiles\templates\profiles\create_profile.html

```
<body>
  <form action="/profiles/" method="post" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form }}
    <button>Upload!</button>
  </form>
</body>
```

Now also we need change some code in view

\feedback\profiles\views.py

```
from .form import ProfileForm
class CreateProfileView(View):
    def get(self, request):
        form = ProfileForm
        return render(request, "profiles/create_profile.html", {
            "form": form
        })
    def post(self, request):
        submitted_form = ProfileForm(request.POST, request.FILES)
        if submitted_form.is_valid():
            store_file(request.FILES["user_image"])
            return HttpResponseRedirect("/profiles")
        return render(request, "profiles/create_profile.html", {
            "form": submitted_form
        })
```

now because we use form.py error handled automatically.

And now it looks like before.

oo6

also, we can use models to save our files

\feedback\profiles\models.py

```
class UserProfile(models.Model):
    image = models.FileField(upload_to="images")
```

the upload_to field tells django where save the files because files doesn't store in database just path of files store in database (also, we need to do migrations.)

Ok for now django save files in base directory of project but if we want change path to uploads folder, we can do this in setting.py

```
\feedback\feedback\settings.py
```

```
MEDIA_ROOT = BASE_DIR / "uploads"
```

So now in view we can use this model

```
\feedback\profiles\views.py
```

```
from .models import UserProfile
class CreateProfileView(View):
    def get(self, request):
        ...
    def post(self, request):
        submitted_form = ProfileForm(request.POST, request.FILES)
        if submitted_form.is_valid():
            profile = UserProfile(image=request.FILES["user_image"])
            profile.save()
            # store_file(request.FILES["user_image"])
        ...
```

Now also we don't need our function because django automatically save it.


```
# def store_file(file):
#     with open("temp/image.png", "wb+") as dest:
#         for chunk in file.chunks():
#             dest.write(chunk)
```

So, I commit it.

Now if we upload some file, this is how it looks like.

```
├── uploads
│   └── images
│       └── MRH.png
```

And in database

Table:  profiles_userprofile		
	id	image
	Filter	Filter
1	1	images/MRH.png

oo7

now also we have a field just for images with this field user just can upload the images, so we have built in validation.

```
\feedback\profiles\models.py
```

```
class UserProfile(models.Model):
    image = models.ImageField(upload_to="images")
```

and we need change the form filed

```
\feedback\profiles\form.py
```

```
class ProfileForm(forms.Form):
    user_image = forms.ImageField()
```

but when we run server, we get an error.

ERRORS:

```
profiles.UserProfile.image: (fields.E210) Cannot use ImageField because Pillow is not installed.
HINT: Get Pillow at https://pypi.org/project/Pillow/ or run command "python -m pip
install Pillow".
```

As you see, Django says, for the image field it needs an external library called pillow, so we need to install it.

pip install pillow

Now it works fine and if a user uploads a file (not a photo) he gets this error:

User image:

- Upload a valid image. The file you uploaded was either not an image or a corrupted image.

No file selected.

oo8

however, we have a special view for this job

```
\feedback\profiles\views.py
from .models import UserProfile
from django.views.generic.edit import CreateView
class CreateProfileView(CreateView):
    template_name = "profiles/create_profile.html"
    model = UserProfile
    fields = "__all__"
    success_url = "/profiles"
```

this view work likes other class views. With this view we can delete our long old view.

oo9

so now if we want to see the photos, we can do this:

let's create a view

```
from django.views.generic import ListView
class UserProfiles(ListView):
    model = UserProfile
    template_name = "profiles/user_profiles.html"
    context_object_name = "profiles"
```

as you see we use list view because we want to return a list of photos to user.

So now for template we use 'for' like this (profile.filed_name_in_database.url).

```
\feedback\profiles\templates\profiles\user_profiles.html
```

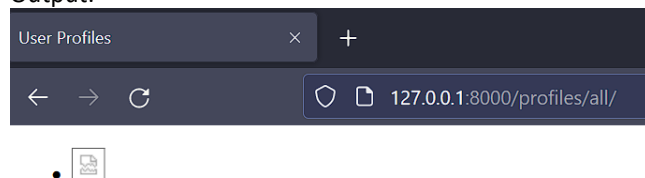
```
<body>
<ul>
    {% for profile in profiles %}
    <li>
        
    </li>
    {% endfor %}
</ul>
</body>
```

And at lest we need set the URL

```
\feedback\profiles\urls.py
```

```
urlpatterns = [
    path("", views.CreateProfileView.as_view()),
    path("all/", views.UserProfiles.as_view())
]
```

Output:



Html:

```
<li>
    
</li>
```

As you see we get the URL but when we want to open it, we get 404 error.

o10

django automatically lock all the folder because of security reasons. So, we need say to django these files are ok to showing up.

```
\feedback\feedback\settings.py
```

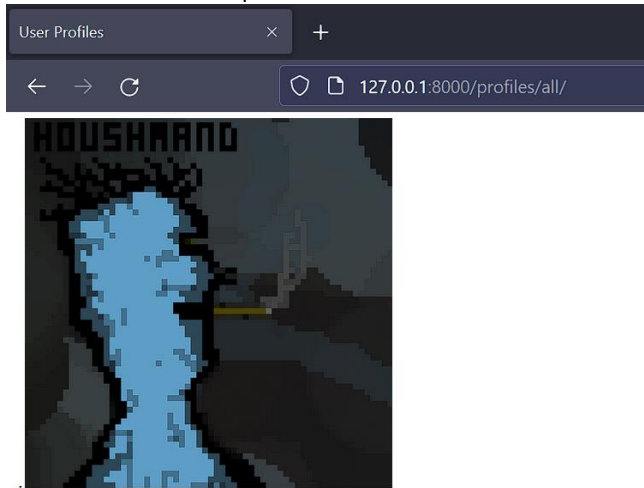
```
MEDIA_URL = "/user-uploads/"
```

Now the images URL start with this

```
\feedback\feedback\urls.py
```

```
from django.conf.urls.static import static
from django.conf import settings
urlpatterns = [
    ...
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

At first, we need import setting and static
Then we called static, and we input these parameters.
static ("URL of MEDIA", "path of MEDIA")
now we can see our uploads



URL of this photo
<http://127.0.0.1:8000/user-uploads/images/MRH.png>

https://github.com/houshmand-2005/hash_neco/tree/File_Uploads_12

13

SESSIONS

The sessions are between client and server and, session framework lets us to store and retrieve arbitrary data on a per-site-visitor basis. For more information you can see this: **003 What are Sessions**

004

at first, we need take look to our project to see are we install sessions (by default its install)

`\feedback\feedback\settings.py`

```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
]
MIDDLEWARE = [
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',
]
```

Also, we can set how long a session must be alive by this command

```
SESSION_COOKIE_AGE = 120
```

By default, its two weeks, so I don't want to change it, so I delete it.

005

so, for test sessions I want add a button for each review to user set it to his or her favorite review so for this I add a Button

`\feedback\templates\reviews\detail_review.html`

```
<form action="/reviews/favorite" method="POST">
    {% csrf_token %}
    <input type="hidden" name="review_id" value="{{ review.id }}">
    <button>Favorite</button>
</form>
```

I add a hidden input to django knows which reviews it's my favorite.

So now also we need URL for this

`\feedback\reviews\urls.py`

```
urlpatterns = [
    ...
    path("reviews/favorite", views.AddFavoriteView.as_view()),
    path("reviews/<int:pk>", views.DetailReview.as_view())
]
```

As you see we most set URL before the detail review.

And for view:

`\feedback\reviews\views.py`

```
from .models import Review
class AddFavoriteView(View):
    def post(self, request):
        review_id = request.POST["review_id"]
```

as you see we get the detail of favorite review but how we can store in session.

006 and 007

```
class AddFavoriteView(View):
    ...
    request.session["favorite_review"] = review_id
    return HttpResponseRedirect("/reviews/" + review_id)
```

Now we set the session with the favorite review then we redirect user to selected favorite review.

008 and 009

now we want if user click on favorite show some text and hide the button.

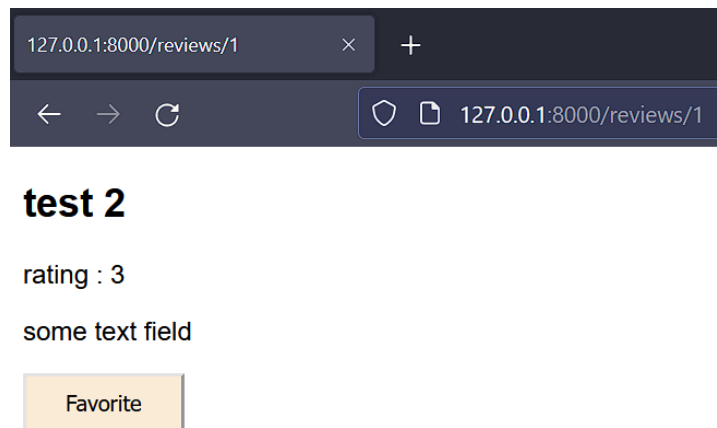
```
class DetailReview(DetailView):
    template_name = "reviews/detail_review.html"
    model = Review
```

```
def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    loaded_review = self.object
    request = self.request
    favorite_id = request.session.get("favorite_review")
    context["is_favorite"] = favorite_id == str(loaded_review.id)
    return context
```

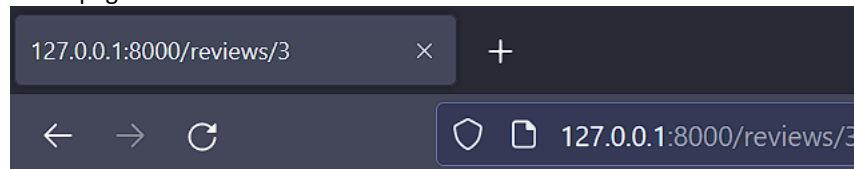
As you see we get information of use visit page with self.object then we get the request user sent and with that we can get the session now if user clicked on the button we have a key here and then if the key we get from session is equal to key of page user see we pass true. So now in html file we can handle it.

```
{% if is_favorite %}
    <p>This is my favorite!</p>
{% else %}
    <form action="/reviews/favorite" method="POST">
        {% csrf_token %}
        <input type="hidden" name="review_id" value="{{ review.id }}">
        <button>Favorite</button>
    </form>
{% endif %}
```

Now let's test it

	
---	---

Other pages:



hello

rating : 2

this is a another test

IMPORTANT NOTES

So, you finished the book, what can you do next?

You can learn more and dipper with documents and these 2 parts of django course by mosh

<https://codewithmosh.com/p/the-ultimate-django-part2>

<https://codewithmosh.com/p/the-ultimate-django-part3>

if you want to know about deploy you can see S15_Deployment.

also, you can learn DRF (django rest framework) and learn about docker because if you want to deploy your project its very useful.

Also, you can learn about Linux and Fast API.

This book is just a booklet of django not a full book so there may be problems. I write this book because I want to learn deeper and when I forget a part of django I look to this book and then I can remember it.

I'm not English, so there may be a lot of grammar and spelling problems, so if you can help me with this go to my GitHub.

My GitHub ID: <https://github.com/houshmand-2005/>

My Telegram ID: https://t.me/Houshmand_Am

By: Amir Mohammad Houshmand

From Apr 23, 2022, to June 27, 2022

django

Django makes it easier to build better web apps more quickly and with less code.

WHY DJANGO?

With Django, you can take web applications from concept to launch in a matter of hours. Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

If you want to learn django you can start with this book

Amir Mohammad Houshmand