DJANGO

Date __/__/__

→ created by Adrian Holovaty and Simon Willison in 2003 when they worked at the Lawrence Journal World newspaper

→ released under BSD license in 2005 and in 2008

→ good online documentation

→ Uses : a) for creating web applications

b) it provides rules, structures and functionality that allows us to use Python code and libraries on the back end of our web application.

c) Django can interact with our web application to send information to the user of the web application.

→ Why use?

- → fast development
- → common features included
- → very scalable
- → very versatile with Python.

d has a lot of built in Python modules that take care of really common web application features

Date __/__/__

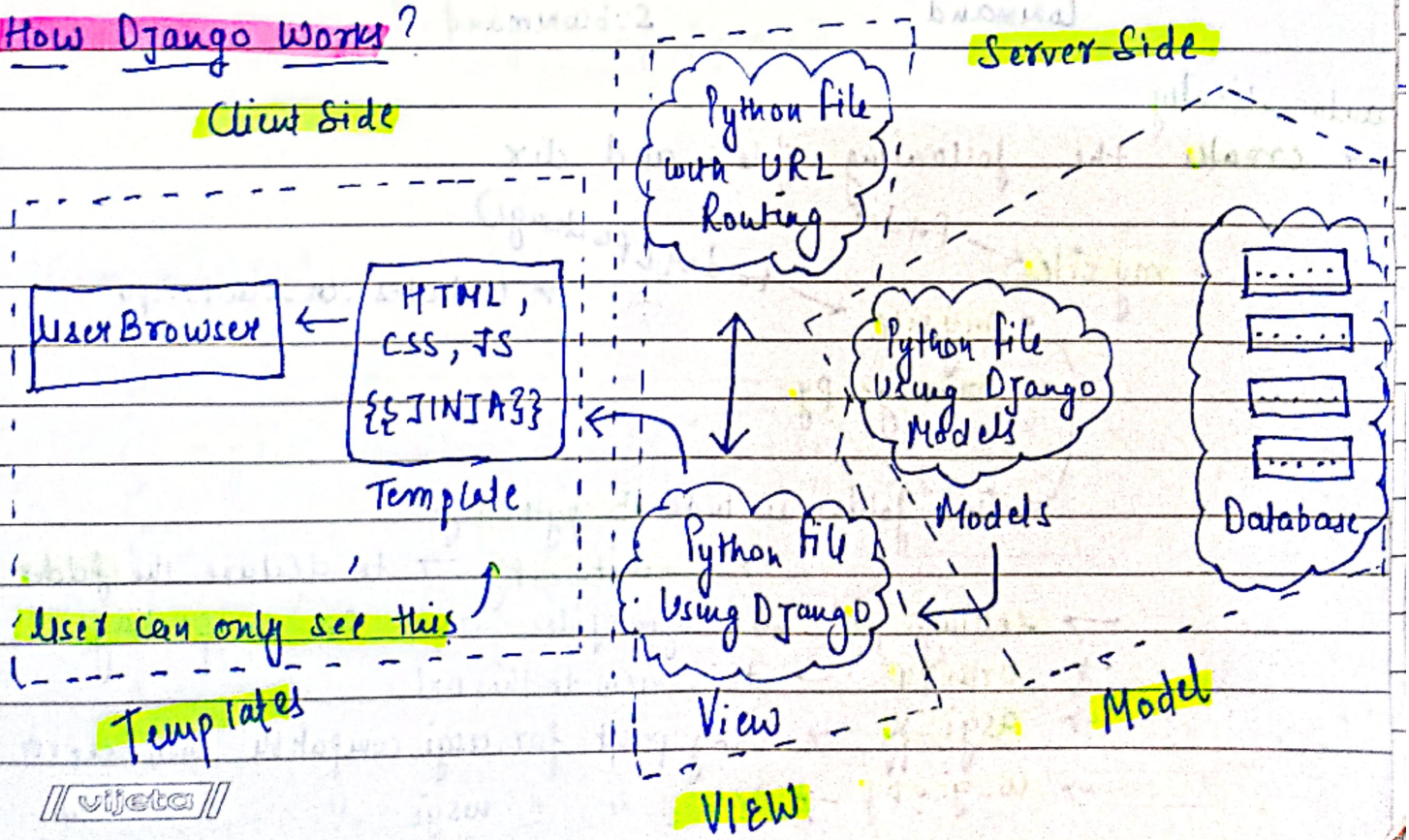
Who use Django?

→ Youtube
→ Instagram
→ Spotify
→ Pintrest
→ Dropbox
→ EventBrite

Key Features of Django?

→ follow MTV (Model - Template View) Structure
→ ORM → Object relational Mapper
→ Models
→ URL and Views
→ Templates

How Django Works?



Date / /

→ Django str also have many features such as authentication and administration access.

Drawbacks

→ heavily reliant on idea of Models

→ add the requirement of understanding Models and setting them up for views

Create a new django

→ in terminal

```
django-admin startproject mysite
     ↓           ↓          ↳ name of the proj
  command    subcommand
```

automatically

→ create the following files and dir

- mysite ← folder
  - mysite ← folder (package) ↳ includes .or __init__.py
    - mysite
    - manage.py

↳ This folder includes 5 python files

→ __init__.py → to declare the folder as an package

→ settings.py → config file

→ urls.py → this view to this url

→ asgi.py → entry point for asgi compatible web server

→ wsgi.py → " " " wsgi " " "

Date __/__/__

How to run the server?

→ go to the directory, which includes manage.py

→ python manage.py runserver
by default, it starts the server on port 8000

→ if you want to change the port running

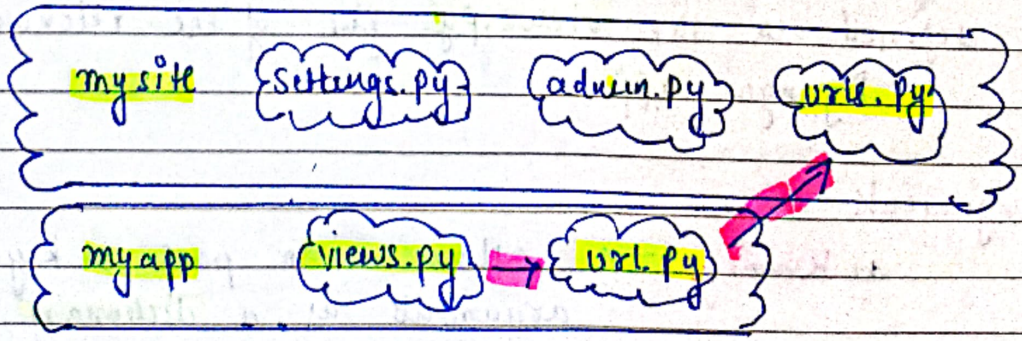
python manage.py runserver 8000

How to create a new app?

python manage.py startapp app_name

↳ if this command throws an syntax error, try using python3

Django Project



means - which url shows which thing

Date / /

views u like the file which include bunch of HTML and css code just like components in React.

Views and URL Overview

views dictate what information u being shown to the client and URL dictate where that infor~ is shown on the website.

Views/URL pairing is a webpage.

- Connecting a view to a URL with path(), it takes two parameters.

1) route → string code that contains the URL pattern

→ django will scan the relevant urlpatterns list until it finds a matching string route (eg. "app/")

2) view → once a matching route is found, the view argument connects to a function or views, typically defined in the views.py file of the relevant Django app.

Optional arguments

3) kwargs → allow us to pass in keyword arguments as a dictionary to the view

↳ we'll learn later

Date __/__/____

4) name → allow us to name a url in order to refer
its a powerful feature that allows us to simply
reference views and url route through the
use of a custom defined name.

Function Based Views (see the practical of connecting to url routes)

1) create your app
2) add urls.py inside the copp folder or package
3) go to views.py.
from
- add a import django.http.response import HttpResponse
4) for function based views
→ add a function with a parameter
for eg:- def simple_views(request):
return HttpResponse("Simple Views")
5) for connecting to your app-based urls.py
→ go to urls.py (app-based)

→ Import 2 things
1) path → from django.urls import path
2) views.py or your custom view
↳ from . import views
↳ dot means just look
in the same folder youre
currently in.

Date / /

6) Create a list always named as urlpatterns

urlpatterns = [path('', views.simple_view)]

↓

it is set to be the empty string

The reason is because, in my project level urls.py file, that's where am actually going to define what these views are going to fall under

7) Now to connect to the project level urls.py
- go to that file
- add second path to the existing urlpatterns

path('first_app/', include('app_name.urls'))

↓

import it by

from django.urls import include

What if you add a string in route or urls in app based

for eg:-

path('/simple', views.simple_view)

└ app based

path('/first_app', include('app_name.urls'))

↳ project based

→ domain.com/first_app/simple

↓ project ↓ app based

Date __/__/__

proj based url routes → effect or attach directly to the domain

app " " " → " , " " " " "

proj based url

Adding Homepage

in project urls.py file

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

Dynamic Routing Logic

if you want to grab the url entity what user have entered in the url.

To use dynamic routing

→ views

```
def main_view(request, topic):
    return HttpResponse(topic)
```

→ urls.py

path('<topic>/', views.main_view)

path converter → it will convert it into the dellared datatype

for eg 3 to '3'

path('<str:topic>/', vie...)

Date __/__/__

ResponseNot found and 404 Page

use try, except

view → def main_views(request, topic)

```
try:
    result = article[topic]
    return HTTPResponse(article[Topic]

except:
    return HttpResponseNotfound("Allocated Resource Not found")
```

you can still use HTTpResponse but it would be in except
a good practice to do so

404

↳ except

```
raise HTTP404('Allocated Resource Not found')
```

if debug == true, you would see your message with other stuff too
, so go to settings.py file to make it false and allowed host to your ip or localhost

Never make it Debug=true in production, coz it can be security threat

Date __/__/__

HTTP Redirects

for redirects, add your logic to views.py, and return the redirect

return HttpResponseRedirect ('/goals/'+ redirectstring)

Reverse function in django

from (import) django.urls import reverse

in urls.py

path ('<str:topic>/', views.main_view, name='topic-page')

than in views.py

```
topic='finance'
url = reverse('topic-page', args=[topic])
return HttpResponseRedirect(url)
```

Connecting a view to template

1) creating a template
   - define a folder inside your project named template
   - create a html file
   - add your html code

Date / /

(ii) defining or connecting it to views

```
def simple_view(request):
    return render(request, 'first_app/example.html')
```

dir structure

- Project
  - project
  - app
  - template
    - first_app
      - example.html

but how could django know, it is from template folder

Hence it throws an error

To tell django, to search it from this directory path, we need to assign the path in settings.py file

in Template list, you can see dictonaries, which have a key of DIRS, provide the path of template folder in the form of list

you can also see BASE_DIR which is using pathlib

1) __file__ :- takes the current location of the file which you executing
2) .parent :- go one directory back just like cd.. in linux

for eg :- home/kali/main/p.txt → home/kali/main

Date __/__/__

Template Directories

1) create an app and view and bind the url routes. also with the project based.

2) run the command

```
python3 manage.py migrate
```

3) Now we want to tell the project that we make this app and we want to get it installed.

4) for that go to app.py where you can see the name of your app in camel casing name with a function

Snake casing :- written in lower case and seperated by (-)
eg:- my-snake-cased-name

Camel casing :- no space and (-), each word begins with capital letter (except first one)
eg:- mySnakeCasedNome

copy the function name and go to settings.py, in installed app list, add your app like this

```
'my_apps.apps.MyAppConfig'
```

- my_apps → folder name
- apps → apps.py
- MyAppConfig → function-name

Date / /

5) save the changes

6) now run → python3 manage.py makemigrations my-app

since we don't have any models, it will say like no changes detected in app.

7) again migrate → did in 2nd step

8) Now project is aware of our app and now we dont need to specify our template folder in Template list in settings.py.

9) Now created a template folder inside your app folder, and inside template, create one folder named as the app_name and inside that add your template

10) give the path like prev in function

def return render(request, 'first_app/example.html')

Date __/__/__

Variable in templates or how to use variable in templates

We can only use variables in the form of dictonaries

for eg::

```
def variable_views(request):
    dicton = {'name':'vishnu', 'hobbies':['singing', 'dancing']
              ,'examp_dict':{'1':'value'}}

    return render(request, 'my_app/variable.html',
                  context = dicton)
```

↑ for connecting to a template

To use in a template

variable.html

```
<h1> Name : {{name}} and hobbies are
{{hobbies.0}}, {{hobbies.1}}
<h1>
<p> My dictonarie {{example_dict.1}} </p>
```

How to do comment

```
<h2> {# This is a comment #} </H2>
```

in template language

, we use '.' but in python we use

hobbies[0]

Date / /

Template language name is Django Template language or Jinja

Filters :

{{ first_name | upper }} → upper case
" | length }}
" | lower }}

you can search for "built-in template tags and filters"

Tags :- for loop

```
<ul>
{% for hobby in hobbies %}
    <li> {{hobby}} </li>
{% endfor %}
</ul>
```

→ tells the python to treat it as a tag

telling end of the for loop

if - else, else if

cond^n

```
{% if logged in %}
<h1>  <h1>
{% else %}
<h1>
<% endif %>
```

if cond^n

Date __/__/__

spacing matters in if-else

for eg {% if name=='vishnu' %} ✗ | {% if name == 'vishnu' %} ✓

No space | spaces are must

Tags and URL Names in template

for redirecting to other url in the html, like a link to other url, we follow some steps

in your app-based urls.py

1) we had to register with django, this app name

app_name = 'my_app' → app name

path('variable/', views.variable_view, name='variable'),

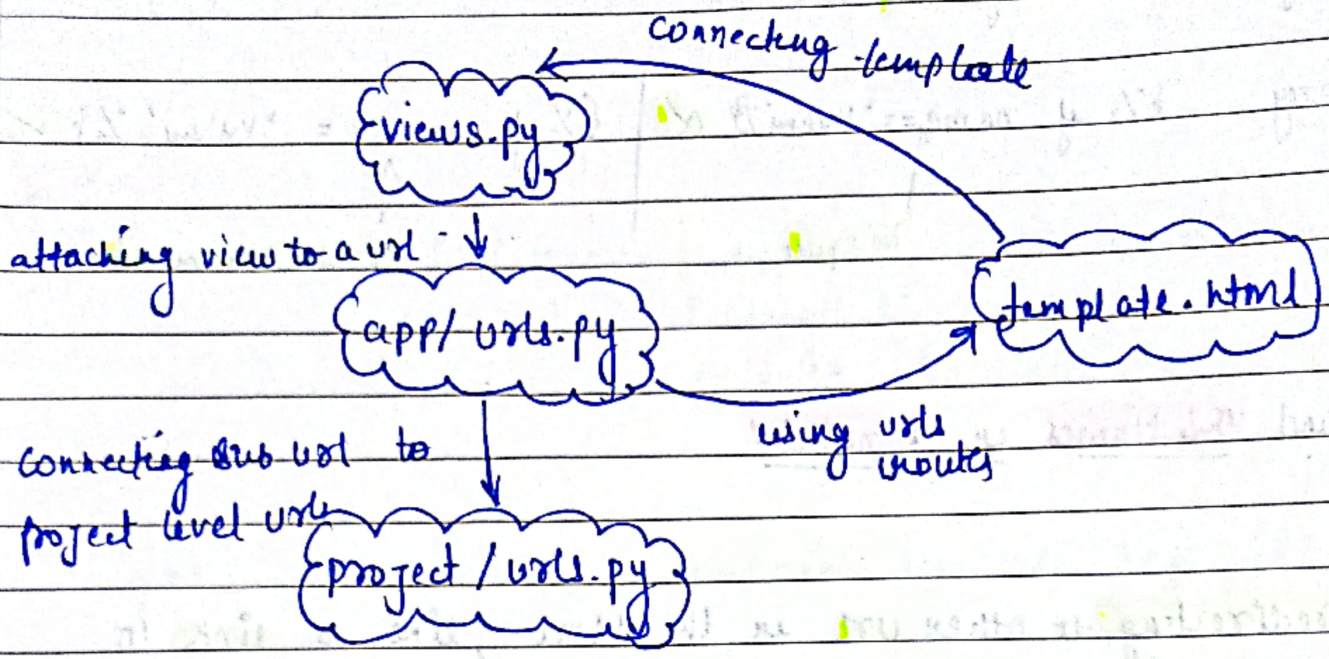
2) to use it in our template

<a href = {% url 'my_app:variable' %} → registered name space; → Keyword

<button> Login </button>

→ url route name

Date / /



Template Inheritance

header.html

```
<Doctyp  >
    <html>
    <head>
    <body>
    <h1> Outside of block </h1>
        {% block homepage %}

        {% endblock %}
    <h> Outside of block</h1>
    </html>
```

example.html (no need of <html> tags head, body as it is already present in the base class)

Date __/__/__

b

{% extends 'my_app/header.html' %} ← folder inside your installed app template
{% block homepage %}
Header Homepage
{% endblock %}

The only code no need of giving body or head tage

in browser (example.html)

Outside of block
header Homepage
Outside of block

Custom Error Template

for attaching a error template to the site level when user goes to a wrong url.

1) create ~~url~~ a template folder inside your project
2) and add a 404.html file
3) Go to settings.py
4) Do the following changes

Debug = False
Allowed Host = [*]

5) you ready to go.

Date / /

How to add name of the 404.html template to other name?

→ you can still use if you don't want to use 404.html

→ change the name of the template like 404.html to error.html
→ in myside, create a views.py file
→ inside that add a function like below

```
def my_custom_404_view(request, exception):
    return render(request, 'fourZerofour.html', status=404)
```

→ in urls.py (project-based)

add this string

```
handler404 = "my-side.views.my_custom_404_view"
```

Static files: to use the image and css file to your template we use static files

1) create a static folder (wherever inside app or project)
2) add the css or static file
3) check in settings.py
   in installed app
   check contrib.staticfiles is there or not
   check for static_url is there or not
   static_url = 'static'
   # my_app/static/my_app/django.css
   define this → you'll see in later why we need to

Date / /

4) Now we need to link the static file to our templates for that we have two ways.

first way: Load the static tag

{% load static %}

<link rel='stylesheet' type='text/css' href="../../static/my-app/variable.css"

<link rel='stylesheet' href='{% static 'my_app/variable.css' %}?{% now 'U' %}">

check in prev page we had static url set, so it will search inside the static folder for myapp and then css

{% now 'U' %} used to display the current timestamp for unix, it is appended as query parameter to the CSS file URL. When the CSS file is modified, the timestamp changes and the browser fetches the updated version instead of using a cached copy

for eg: ../../static/variable.css

↳ browser only reloads the css file once, when the user reload and you made some changes to your CSS, the browser still take it

Date __/__/__

from cache.

so now everytime user reload, we got the different link url

→ <link ... href = /app/variable.css ? 167890.

/app/variable.css ? 16790

so hence with every reload it ~~fetch~~ load the css file

This technique is useful during development or when making frequent updates to your css file.
It help ensures that the browser always retrives the latest version

Second Way :- Normal Way

→ no need to add, just give relative path.

<link rel = 'stylesheet' href = '../../static/my-app/variable.css'>

Django - MODELS, DATABASE, QUERIES

Date / /

Models :- allows us to interact with a database with Python and Django.

includes key interaction with a database CRUD

- Create
- Read
- Update
- Delete

SQL vs NoSQL

- Sql stores data in a tabular format
- for eg

id	name	color
0	John	blue
1	Marry	red

- NoSQL stores data in a key/value pair format
  NoSQL such as MongoDB

Date / /

Which is better? SQL or NoSQL

→ both are different

→ Djang Models works better with SQL based format.

→ To switch for NoSQL, you should check does it actualy provides a major advantage to your project.

→ we are gonna use sqlite

→ if your website hits 100K hits traffic perday, sqlite gonna works fine with that

How to create Models?

1) Create your project

2) create the app

3) once the app is created, run migrations

```
python3 manage.py migration
```

What does it do?

→ it will create the take a look at your default installed app or any apps you

Date __/__/__

add to this list and theres going to create any neccesary database tables according to the settings that you had under database settings

you can see in settings.py file in DATABASES dictionary

→ it also creates the migration table. Django creates a table named 'django_migration' in your database to keep track of which migration have been applied

→ this table used to ensure that each migration is applied only once and to manage the migration History.

4) After migrations, go to models.py

5) Add the class like this

```
class Patience (models.Model):
    first_name = models.CharField(max_length=30)
    last_name = "" . "" ( "" = 30)
    age = "" . IntegerField()
```

for checking more of datatypes, go to Model field reference in django official documentation

Date / /

3 - Migrate Based Command

1) make migrations
2) migrate
3) sqlmigrate

- python manage.py makemigrations app

→ it actualy creates, (but does not run) the set of instruction that will apply changes to the database

specified in models.py

→ def app (eg: auth, admin) already have their default sql make migration code ready.

→ You can see the migration code files under
- app
  - migration
    - 0001_initial.py

- python manage.py migrate

→ runs any existing migrations (typically created through the makemigrations command)

→ runs the file under the migrations directory.

Date __/__/__

- python manage.py sqlmigrate app 0001

  → if you wanna see what the SQL code looked like, you could run sqlmigrate command to view it

  ↳ no. associated with the specific migration python file

a backend

→ help in case you had SQL based developer who wanted to see what actual code or SQL code was actually being run to your database.

Very first migrate command we run as executing the default makemigrations that was already created for you upon creating the project.

Steps for migrations

1) Initial migrate command.

   that's gonna create the database for you and also. make the migration necessary for things like authentication and administration.

2) Create app and models

3) Register app in installed_apps in settings.py

Date / /

4.) Run makemigrations for new apps

5.) Run migrate for new migrations.

Continuing.... (How to create models)

6) python3 manage.py makemigrations office

(In terminal)

office\migrations\0001_initial.py

- Create model Patient

To see the what sql command does it gonna run.

python3 manage.py sqlmigrate office 0001

In terminal

```
CREATE TABLE "OFFICE_PATIENT" (
)
```

Date __/__/__

Data Interaction: Creating and Inserting

→ inserting data is super easy.

→ just create a new instance and than call .save() method on the obj to create and insert call to the SQL database.

→ let's run

```
python3 manage.py shell
```

it's start a python shell session with all the necessary django configs and modules are already imported.

→ import the model

(in small-case)

```
from app_name.models import MyModel
```

in my case " `Offica.models import Patient`

→ `carl = Patient(first_name='Vish', last_name='Tiw', age=30)`

→ `carl.age` ⟶ 30

→ `carl.save()` → it runs the insert command

Date / /

We are doing in 2 steps
1) creating
2) saving

If you don't wan to do in two steps
you can do this

→ Patient.objects.create(first_name='sushan', last_name='smith', age=30)

↳ object created and saved in only 1 step

→ You can also create multiple object at once

→ mylist = [Patient(first_name='adam', last_name='smith', age=40), Patient(first_name='karl', last_name='marx', age=40)]

→ Patient.objects.bulk_create(mylist)

We are using MyModels.objects like in query

it is actualy the Django Model Manager

→ this manager can them actualy read the database through the use of methods calls, like .all() and .get() and narrow down results with .filter and .exclude.

Date / /

How to view the records ?

→ For viewing all

- Patient.objects.all()
- ↳ returns a list like this

<QuerySet [<Patient: Patient object (1)>,]>

for grabing specific, you can provide indexes

Patient.objects.all()[]

but the data is not human redable

By default, when you access a specific object from the queryset, it returns the object itself rather than a human redable representation.

Queryset hold the bunch of objects in the form of list and we know that if we have a object of class, and you are gonna print that and you want a specific string representation., we use __str__

in your class add __str__

Date __/__/__

Django's Queryset has a default __repr__() implementation that includes the string representation of each object. By default, it uses the '__str__()' method of the model to generate the string representation of each object.

Hence, that's why we got the string representation of object in the Queryset list.

```
def __str__(self):
    return f'{self.first_name} {self.age}'
```

now you'll be able to see the output.

→ After running the migration on app, we got some file like __0001_initial.py, in that we can check in fields, you can see on id column which tells that it will be always unique like a primary key that means, if you just insert two object with same data, it doesn't gonna duplicate that. As it have an id key associated with that which is uniquely identified.

Date __/__/__

Data Interaction : filter and Get

→ get always gonna return one record and if you provide a condⁿ which gonna return more than one record, it will throw an error

→ To use

```
Patient.object.get(first_name = 'carl')
```

↳ gonna return the record which have carl as a first name

```
Patient.object.get(pk = 1)
```

pk — primary key

↳ gonna return the very first record which is entered.

filter

→ it can return multiple records

for eg:- `Patient.objects.filter(last_name = 'Gallagher')`

↳ gonn a return every record which met the condⁿ

```
Patient.objects.filter(last_name = 'Gallagher').filter(age=34)
```

or

```
" " " (" " ", age = 34)
```

Date / /

For using operators like and & OR, we need to import Q

from django.db.models import Q

→ Patient.objects.filter(Q(last_name='Gallagher') | Q(age=40))

don't forget to provide the q for another filter.

& → and

Field Lookups

for more complex filtering operations, we use field lookups with a filter() call.

Imp: go to the official django documentation for all field lookups

Patient.objects.filter(last_name__startswith='Gi')

↳ double underscore

" " " (age__in=[10, 20, 40])

" " " (age__gte=30)

↳ greater than equal to

Date __/__/__

Go for QuerySET API reference for checking all kind of methods which return query set other than get and filter

like order by.

Data Interaction : Updating Models

Suppose, if you want to add another coloumn to your existing database.

→ go to models.py in your model class, add the coloumn like this.

heartrate = models.IntegerField()

↓

but it will gonna throw an error as we had the existing records and for that we had to provide some default value to that or may be null we can't left that empty.

heartrate = models.IntegerField(default = 60)

Only one

or

null = True

↳ will give null value to the existing records.

Validators : for validating or to use validators, we need to import some validators

Date __/__/__

```
from django.core.validators import MaxValueValidator
                                 , MinValueValidator

heartrate = models.IntegerField( validators = [MinValueValidator(1)
                                 , MaxValueValidator (200)]
```

don't forget to make migrations and migrate it.

How to update a record in a database?

→ assign the object name to that record

```
lip = Patient.object.get(first_name = 'Philip')
lip.first_name = 'Lip'
```

↳ it will change that

How to delete a record on a database?

→

```
lip.delete()
```

Date __/__/__

Connecting Templates and Database Models

1) create a view

```
def list_patient (request):

    all_patient = Patient.object.all()
    context = {'patient' : all_patient}
    return render (request, 'office/list-html',
                                    context = context)
```

2) in your list.html

{{ patient }} → gonna return ~~Queryset list~~

```
<ul>
    {% for i in patient %}
        <li> {{ i.first_name }} </li>
    {% endfor %}
</ul>
```

Date __/__/__

Note: Whenever importing models to views, always add a .

for eg: from . .models import ModelName

If you are confused on how to display or add changes to the database via web

↳ always you have to connect your models into your view and pass it via content in a dictionary

Date __/__/__

form

in list.html

```
<form action="" method="POST">
    <label for="brand">Brand</label>
    <input id="">

    <label for="year">Year</label>
    <input id="">

</form>
```

{% csrf_token %} important

Now in your views.py

```
def add(request):
    print(request.POST)
    return render(request, 'cars/add.html')
```

Now what will happen is, if user reload the page (add.html) add function will execute

When you first time reload, you got the empty dictionary cuz you don't fill the form

after that if you fill the form and click on submit button, it will reload the data will be captured on the request.post

Date / /

<QueryDict: {'csrfmiddlewaretoken': ['...'], 'brand': ['toyota'], 'year': ['2019']}>

for adding to the database via templates, create a form in template first

```
if request.POST:
    brand = request.POST['brand']
    year = int(...)['year']

    Cars.Objects.create(brand=brand, year=year)
    return redirect(reverse('cars.list'))

else
    return render(request, 'cars/add.html')
```

for deleting

```
if request.POST:
    pk = int(request.POST['pk'])
    try:
        Cars.objects.get(pk=pk).delete()
        return redirect(reverse('cars:list'))

    except:
        print('Pk Not found')
        return redirect(reverse('cars:delete'))

return render(request, 'cars/delete.html')
```

Date / /

Django Administration

- creating a superuser

```
python3 manage.py createsuperuser
```

Django Admin and Models

How to register your model to the admin?

→ go to admin.py

→ import your model, and always import like this or you'll get an error

```
form .models import Cars ✓
```

or

```
from cars.models import Cars ✓
```

↓ don't do

```
from models import cars ✗
```

→ # Register

```
admin.site.register(Cars)
```

Now you'll be able to see your database in admin site and able will to add or delete the entries

Date / /

~~dar~~

if you want to add some changes or wan to change how these thing look, do it like this for eg.

```
class CarAdmin(admin.ModelAdmin):
    field = ['year', 'brand']

admin.site.register(Car, CarAdmin)
```

it will change the order of your form whenever you editing or adding an record

Just like this, you can do more stuff in this

Date __/__/__

Django form

Get: request data from a specified resource

Post: " to send data to a server to create / update a resource.

csrf: when an attacker tricks a victim into performing unwanted actions on a website where the victim is authenticated

django creates these csrf tokens for us automatically with a simple tag call.

{% csrf_token %}

Django form Class

- create new project and app
- connect templates, views and urls.
- create a forms.py file
- " " django form class inside forms.py
- connect django form to view for context insertion inside template.

Date / /

Create forms.py

```
from django import forms

class ReviewForm(forms.Form):
    first_name = forms.CharField(label='First Name',
                                 max_length=30)
    last_name = forms.CharField(label='Last Name',
                                max_length=30)
    email = forms.EmailField(label='Email')
```

Now in your ~~template~~ views

```
def rental_views(request):
    if request.method == 'POST': # means if form is submitted
        form = ReviewForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data)  cleaned-data
            return redirect(reverse('cars:thank_you.html'))
    else:
        form = ReviewForm()

    return render(request, 'cars/rental_reviews.html', context
                  = {'form': form})
```

Date __/__/__

form = ReviewForm(request.POST)

↓ and it means the form is filled

↓ it stores the data when you submitted the form

form = ReviewForm()

↓ means form is empty

form.is_valid()

↓ check for the validity

Note: if a user fills the form and ~~did~~ submit the forms, and there's no redireds, and than user refreshes the page, it resubmits the form and make a POST request (in views)

template.html

```
<form method = 'POST'>
    {% csrf token %}
    {{ form }}
</form> <input type = "submit">
</form>
```

Date __/__/__

Template Rendering

you can also provide some adjustment to your code

foreg

{{ form.as_p }}

(it will add the label and input inside a paragrap

you can also use div

→ {{ form.first_name.label_tag }} {{ forme.first name }}

This will give label name

foreg: in forms.py

first_name = forms.CharField(label='First_Name Please write your review here')

gonna give the input field

First Name: [ ]

→ {% for field in forms %}

{{field.label_tag}} {{field}}

it will etorate over each field and field is equal to field.~~is~~ form.first_name after iteration the next field

Date / /

Widget and Styling :-

Each field has some arguments and one of this is widgets

review = forms.CharField(label='Please write your review here', widget = form.Textarea())

How to style widget?

you can add attrs

widget = form.Textarea(attrs = {'class': 'myform'}))

you can also add attributes which are available to that Text area

for eg

wid... = fo... TextArea(attrs = {'row': 2, 'colocumn': 2})

Model form

you can create form based on the fields present at your models

follows the steps to do it

Date __/__/__

go to forms.py

```
from django import forms
  # import model
from car.models import Review
  # import ModelForm
from django.forms import ModelForm


class ReviewForm(ModelForm):
    class Meta:
        model = Review
        field = ['first_name']
```

fields which you want to display in the form

go to views.py

```
def rental_review(request):
    if(request.POST)
     if request.method == 'POST':
          form = ReviewForm(request.POST)
          if form.is_valid():
              form.save()
              return redirect(reverse('
                  cars:thank_you'))
```

it will save in db.

use:

```
    print form = ReviewForm()

return render(request, 'cars/rental_review.html'
              , context = {'form': form}
```

Date __/__/__

Modelforms Customization

whatever I m doing am not going to define class Review form and Meta again and again, so pls keep in mind.

for changing the label name

```
class ReviewForm(ModelForm):
    class Meta:
        model = Review        # Model Name
        fields = "__all__"
                 # → all fields
        labels = {'first_name': 'Your First Name',
                  'stars' : 'Ratings'}
```

Showing error

→ django automatically shows a default error while using ModelForms

for eg: in models.py

```
stars = models.IntegerField(validators = [MinValueValidator(1), MaxValueValidator(5)])
```

Date __/__/__

Now if a user enters a value more than 5 or min than 1 django automatically gonna show the error

How to give the custom message?

```
error_message = {'stars': {'min_value': 'Min value is 1',
                           'max_value': 'Error Max'
                           }}
```

Date __/__/____

Django Class Based Views

Why to use?

come with many pre-built generic class view for common task

TemplateView:- it connects a view to a template, just like render to

```
def home_view(request):
    return render(request, 'classroom/home.html')
```

By class based view

```
from django.views.generic import TemplateViews

class HomeView(TemplateView)
    template_name = 'classroom/home.html'

class ThankYouView(TemplateView)
    template_name = 'classroom/thankyou.html'
```

Now in your url path, it takes view as a function so for making it to function

```
path('home/', HomeView.as_view(), name='home')
```

Date __/__/__

FormView

1) You create a form.py and add your form
   Once you made the form and ready to go

2) go to views, import your form
   and import FormView

```
from django.views.generic import FormView

class ContactFormView(FormView)

    form_view = ContactForm
```

connecting to the form class

```
    # if success, form submitted sucessfully

    sucess_url = "classroom/thank_you"

    def form_valid(self, form):
        print(form.cleaned_data)
        return super().form_valid(form)
```

def form_valid(self, form): → if form is valid

super().form_valid(form) → ContactForm(request.POST)

Date __/__/__

use reverse_lazy instead of reverse in success url

success_url = reverse_lazy ('classroom: thank_you')

CreateView

↳ it creates a form with the model fields and save it

1) create a class

```
class TeacherFormView (CreateView):
    model = Teacher
    success_url = reverse_lazy ('class: thank_you')
    field = '__all__'
```

↳ it will automatically save it to the db

2) go to the template

→ you don't need to specify the template in TeacherFormView

→ it automatically takes and match the pattern like

model_form.html

for eg: if your model name is Teacher, it gonna take teacher_form.html

Date / /

in teacher_form.html

```
<h1> Teacher Form </h1>
<form method = 'POST' >
    {% csrf_token %}
    {{ form.as_p }}
    <input type = "submit" >
</form>
```

{{form}} → is a placeholder in the

in urls.py

```
path('teacher_form', TeacherFormView.as_view(),
     name = 'teacher_form')
```

List View :- list the object in the database

```
class TeacherListView (ListView):
    model = Teacher
    # model_list.html
```

in teacher_list.html

```
<h1> Teacher List </h1>
{% for teacher in object_list %}
    <li> {{ teacher }} </li>
{% endfor %}
</ul>
```

Date / /

you can change the 'object_list' name to another name

model = Teacher
context_object_name = 'teacher_list'

{% for teacher in teacher_list %}

You can also alter the list based on query

by default, it sending all objects

queryset = Teacher.objects.all()

↓

you can overwrite this and get the list

queryset = Teacher.objects.filter(first_name = 'Vishnu')

↓

don't use get, as it doesn't return a list or queryset

Detail View :- used to display detail of a particular teacher

class TeacherDetailView(DetailView):
model = Teacher
# teacher_detail

Date / /

important, always should be PK

in urls.py

path ('teacher_detail/<int:pk>', TeacherDetailViews.as_views()
, name = 'teacher_detail')

now in template_list.html

{% for - - %}

{{teacher}}

now in template_detail.html

<h1> Teacher Detail </h1>
{{teacher}}

Update Views :-

First Way :- By Query, you provide an query, based on that it update the record

class TeacherUpdateView (UpdateView):
model = Teacher
fields = ['last_name']
success_url = reverse_lazy (' ; ')
template_name = 'class/..'

or def get_object (self, queryset = None)
queryset = Teacher.objects.get (first_name
= 'Vishnu')

Date __/__/__

return queryset

in urls.py

path ('teacher_update_form/', TeacherUpdateView.as_view()
, name = 'teacher_update')

in template

<form method = "POST" >
{% csrf token %}
{{ form.as_p }}
<input type = "submit" >
</form>

Second way → remove the func" inside your class
remove temp

it gonna share the form which is already there
in creat view

in urls.py update_teacher
path ('teacher_detail / <int:pk>', TeacherUpdateView
.as_view')

Date / /

in teacher_list.html

{% for teacher in teacher_list %}

<a href = "/classroom/update_teacher/{{teacher.id}}
">
Update Information

{% endfor %}

Delete View → to delete a particular object from the database

first way - by a query

eg: ~~class DeleteView~~

class TeacherDeleteView(DeleteView):
model = Teacher
success_url = reverse_lazy('classroom:
teacher_list')
template_name = 'classroom/teacher_delete
.html'

def get_object(self, queryset=None)
queryset = Teacher.objects.filter(first_name
return queryset

def get_context_data(self, **kwargs):
context = super().get_context_data(**kwargs)
Context['teacher'] = self.get_object()

Date / /

return context

we had to pass the context, so that we can use in our template

In teacher_delete.html

```
<h1> Delete Teacher </h1>
<h2> {{ teacher.0.first_name }} </h2>
<form method = "POST">
    {% csrf_token %}
    <input type = "submit" value = "Confirm Delete" />
</form>
```

Second way: by primary key

form → confirm delete button
default template name → model_confirm_delete.html

delete both function and template name

in urls.py

path('delete_teacher/<int:pk>', TeacherDeleteViews.as_view()),

in teacher_confirm_delete.html → same as in first way

in teacher_list.html →

```
<li>
<a href = "/classroom/delete_teacher/{{ teacher.id }}">
    Delete {{ teacher.first_name }}
</a> </li>
```

Date / /

Redirect View → extremely helpfull
for eg if you want to redirect the user to a diff web page, like where user in the www.example.com/ . it actually redirects user to www.example.com/app

from
import django.views.generic import RedirectView

path('catalog', include('catalog.urls'))
path('', RedirectView.as_view(url='catalog/'))

Date __/__/__

User Authentication and Sessions

Models :- foreign key is used to create relationship b/w 2 database model, it allows you to establish a connection b/w two models by referencing

```
class Language (models.Model):
    name = models.CharField (max_length = 300)
class Genre (models.Model):
    name = "                "
```

```
class Book (model.Model):

    title = models.CharField (max_length = 200)
    author = "          "  (  "  , 200)
    language = models.ForeignKey ('Language', on_delete
                                  = models.
                                  SET_NULL
                                  , null = True)

    genre = models.ManyToManyField (Genre)
```

```
class BookInstance (models.Model):
    id = models.UUIDField (primary_key = true
```

Date __/__/__

its diffi cult to do with shell or in code
so you can add your all models to admin

for creating a book, you should already have some genre, or language so that you can select it while creating

Showing details on templates

views

```
def index(request):

    num_book = Book.objects.all().count()
    num_instance = BookInstance.objects.all().count()

    num_instance_available = BookInstance.objects.filter
                             (status='a').count()

    context = {'numbook': numbook,
               'num_instance': num_instance
               }

    return render(request, 'catalog/index.html',
                  context=context)
```

Date __/__/__

User Authentication with Django Users

create a group from the admin
add a user to that group.

add the path in url (project-level)

path('accounts/', include('django.contrib.auth.urls'))

create a template folder

template/registration/login.html

in settings -specify the template in settings.py file

os.path.join))

in login.html

{% if form.errors %}
 password or username incorrect
{% endif %}

if user enters wrong password

in settings.py → add this

LOGIN_REDIRECT_URL = '/catalog/create_book'
↳ it will take you to the url after logging in

means user trying to get a restricted page

Date / /

```
{% if next %}
    {% if user.is_authenticated %}
        <p> Dont have permissions</p>
    {% else %}
        <p> Please login to see this </p>
    {% endif %}

<form method = "post" action = "{% url 'login' %}">
    {% csrf_token %}
    {{ form }}
    <input type = 'submit' value = "Login" />
    <input type = 'hidden' name = next value =
                                    "{{ next }}" />
</form>
```

The next parameter is typically used in django to redirect the user to a specific page after a successful form submission or authentication. By including

User authentication on Views :-

* use of if-else

```
{% if user.is_authenticated %}
    <h2> Welcome {{ user.get_username }} </h2>
        <p> logged in </p>
{% else %}
    <Not logged in>
    <but <a href = "{% url 'login' %}
        next = {{ request.path }}"
        login
    </a>
```

Date __/__/__

{{ user.get_username }} → written the logged in user name

<a href = " {% url 'login' %}>

↑ it will take you to login page, but after you login, it will take you to the default login redirect url mentioned in settings.py file.

<a href = '{% url 'login' %}?next = {{ request.path }}>

↳

in searchbox localhost/login?next = /catalog/index

will redirect you after login.

second way :- for function based view

from django.contrib.auth.decorators import login_required

@login_required
def simple_view(request)
return render(request, 'catalog/myview.html')

-- you can't access the page without logging in

Date __/__/__

for class-based view

```
from django.contrib.auth.mixins import
                                    LoginRequired
                                    Mixin
```

```
class BookCreate(LoginRequiredMixin, CreateView):
    model = Book
    field = __all__
    success_url = reverse_lazy('catalog:index')
```

User Registration form :-

```
from django.contrib.auth.forms import UserCreation
                                        form
```

```
class SignUpView(CreateView):
    form_class = UserCreationform
    success_url = reverse_lazy('catalog:index')
    template_name = 'catalog/signup.html'
```

(log in)

In signup.html

```
<form method='POST'>
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="signup">
</form>
```

Date __/__/__

User Specific Page :-

① Foreign key connection b/w the user and the model.

for eg:

from django.contrib.auth.models import User

in models.py

```
class BookInstance (models.Model):
    id = ...
    book = ...
    imprint = ...
    due_back = ...
    borrower = model.ForeignKey (User, on_delete
        = model.Set_Null
        , null = True,
        blank = True)
```

in views.py

```
class CheckedOutBookByUserView (LoginRequire
                                Mixin,
                                ListView):
    model = BookInstance
    template_name =
    paginate_by = 5

    def get_queryset (self):
        return BookInstance.objects.
            .filter(borrower
                = self.request
                .user)
```

Date / /

in profile.html

model_list → as the context object
for generic list view

{% for book in bookinstance_list %}
 {{ bookinstance.id }}
{% endfor %}

Date / /

Linode Deployment

connect to ssh

go to var / www / DjangoAPP

go to github.com
create a repository

getting personalized token
- go to settings
→ go to developer settings / Personal_access_token
→ fill the form
→ Scope → select everything under repo.

copy the command line which is there when you create a repository

→ give username
→ password → paste the token

Now adding your project
git add .
git commit . -m "my second commit"

git push

Your project is uploaded