→ Burrito
→ Fushy Burrito
→ 3
↳ update → 50 (we can use admin Panel)

How Data Retrieval works in Django

Retrieving objects

How to retrieve data from database -

Queryset → (A queryset is nothing just an Collection of objects stored in database)

Queryset Manager Model

Example:

Queryset

Item

Model → Item
Manager → objects
Method → all()

>> Item.objects.all()

Reading Data from Database:

Server: Display on index page of food

in views.py (food)

Reasons ??

namespacing

* urls.py

when we are working on multiple apps

```
app_name = 'food'

<a href="{% url 'food:detail' item.id %}">
```

Black pip install Black
git --version

Completing the detail view :-

```
def detail(reques, item_id):
    item = Item.objects.get(pk=item_id)
    context = {
        'item': item,
    }
    return render(request, 'food/detail.html', context)
```

templates
↳ food
↳ detail.html

```
<h1> {{ item.item_name }} </h1>
<h2> {{ item.item_desc }} </h2>
<h3> {{ item.item_price }} </h3>
```

food/1 2 3 4

Not manually

for clicking we can use <a> tag

def index (request): ① from django.http import HttpResponse
    return HttpResponse ("Helo. world")

def (links Particular View to the Particular Url)

from django.urls import Path, include
from . import views

urlpatterns = [
    Path(" ", views.index, name='index'),
    'item/', views.items, name = 'item'),
]

'hello/'

new urls.py

urls.py

urlpatterns = [
    Path('admin/', admin.site.urls),
    Path ('food/', include ('food.urls')),
]

→
def items (request):
    return HttpResponse ("This is an item view")
HttpResponse ("<h1> This is another view </h1>")

Database & Models –

Blueprint to create table

Pre-installed / configured → SQLite3

App → registration ('food.apps.FoodConfig',) apps.py

models.py

from djang.db import models

class Item (models.Model): ⑤
    item_name = models.CharField (max_length = 200)
    item_desc = models.CharField (max_length = 200)
    item_price = models.IntegerField ( )

→ we have made some changes → Convey it
→ Python manage.py makemigrations food
→ Python manage.py sqlmigrate food 0001
→ Python manage.py migrate

Formating — Tags

```
{% for item in itemlist %}
    <ul>
    <li>
        {{ item.id }} -- {{ item.item_name }}
    </li>
    </ul>
{% endfor %}
```

Variable

~~return render(request, 'food/index.html', context)~~

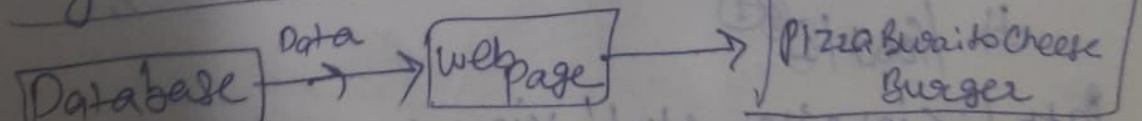
from django.shortcuts import render

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

Why we need Templates

Templates in Depth —

Why do we need templates?



- Pizza
- 3 -- Burrito
- 4 — Cheese Burger

For looping

What we will learn :-

- Introduction to Django. ✓
- Python & Django Installation. ✓
- Food Menu API (views, models, templates, authentication)
- Rest API (Django Rest Framework).
- Pagination, Search, Filtering.
- E-Commerce Site (Shopping Cart, Checkout, Search, Order Placement).
- Admin Panel Customisation.
- Web Based CV/Resume Generator.
- Web Based Link Scraper

index.html

```
{% for item in itemlist %}
    <ul>
    <li>
    <a href="/food/{{item.id}}"> {{item.id}} -- {{item.item_name}} </a>
    </li>
    </ul>
{% endfor %}
```

Django Template Language :-

- ~~not Python~~
- ~~not html~~
- it is DTL

- Templating Engine?
- Jinja2 - Templating Engine
- DTL - Django's Default Engine

Variable in curly braces

{{item.id}}

Removing hard Coded URLs -

name = 'detail'

```
<a href="{% url 'detail' item.id %}">
```

a = a.save()

a.id

a.pk

b = Item(item_name = "Burger",
item_desc = "Cheesy Burger",
item_Price = 10)

b.save()

b.id

b.pk

Item.objects.all() exit()

Python manage.py createsuperuser

Username :- Name
Email address : abc@abc.com
Password : Superuser

```
def __str__(self):
    return self.itemname
```

Re model Register

Sachin
Pqr@Pqr.com
Superuser

Pip freeze

pip install virtualenv

cls

virtualenv environmentname
e.g venv

Python -m virtualenv env

env\Scripts\activate

env\Scripts\deactivate

templates

```
from django.template import loader
loader.get_template("food/index.html")
template
context = {

}
return HttpResponse(template.render(context, request))
```

5.

pip install django

pip freeze

conda create -n newenv Python=3.6 -y
conda activate newenv
conda

Conda env list (10)

Python --version Pip Install Django

Py -m django --version then add to Path

(CMD)

Location → django-admin startproject (mysite)

- init.py → Package

manag.py → Interaction (It allows us to Interact with django Project.)

- settings.py → Settings & Configuration.

urls.py → Urlpattern matching.

Python manage.py runserver (4444)

Python manage.py startapp food

Python manage.py runserver 4444

from . import views

from django.urls import path, include

Python manage.py migrate

from django.db import models

Python manage.py makemigrations food

Python manage.py Sqlmigrate food 0002

Python manage.py Shell

Item.objects.all() Import Item

a = Item(item_name = "Pizza", item_desc = "cheesy pizza", item_price = 20)

views.py

```
from django.template import loader

def index(request):
    itemlist = Item.objects.all()
    template = loader.get_template('food/index.html')
    context = {

    }
    return HttpResponse(template.render(context, request))
```

for data

(remove boiler plate)

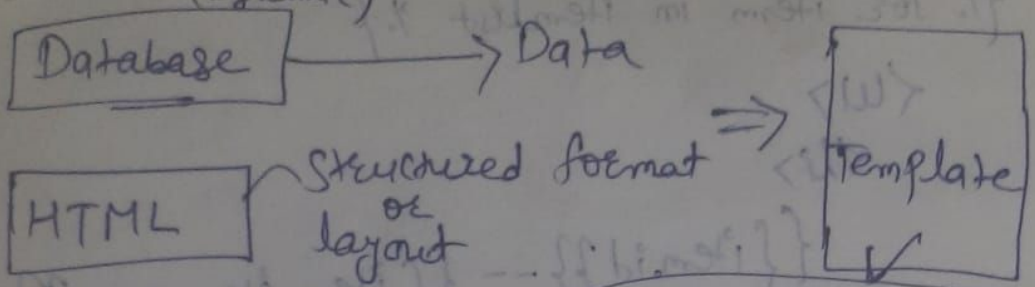
```
context = {
    'itemlist': itemlist,
}
```

in index.html — templates syntax

```
{% for item in itemlist %}
    {{item.id}}-{{item.item_name}}
{% endfor %}
```

for looping

templates do? (8)



We Can Combine both Static & Dynamic

Let's create a template :-

Creating the Detail View

In views.py

```
def detail(request, item_id):
    return HttpResponse("this is item no/id: %s" % item_id)
```

In urls.py

```
# /food/
path('', views.index, name='index'),
# /food/1
path('<int:item_id>/', views.detail, name='detail'),
```

>> Item.objects.all() (3)

(*) String representation

```
def __str__(self):
    return self.item_name
```

first exit(), then re-enter, then again import

```
from food.models import Item
>>> Item.objects.all()
```

Django Admin Panel & Creating Super-User

Python manage.py createsuperuser
Username : Sanjay
Email address: abc@abc.com
Password : superuser
again (4)

server-start

/admin

for include: admin.py (5) (6)

from .models import Item

Register your Model:

admin.site.register(Item)

views.py (food)

```
from .models import Item

def index(request):
    item_list = Item.objects.all()
    return HttpResponse(item_list)
```

Admin Panel → add Something (InCognito)

Cheese Burger

This is a Burger

so

(Color formatting)

Templates → (HTML dynamic)

Django Templates -

Django TEMPLATES engine

in views.py

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

templates

→ food

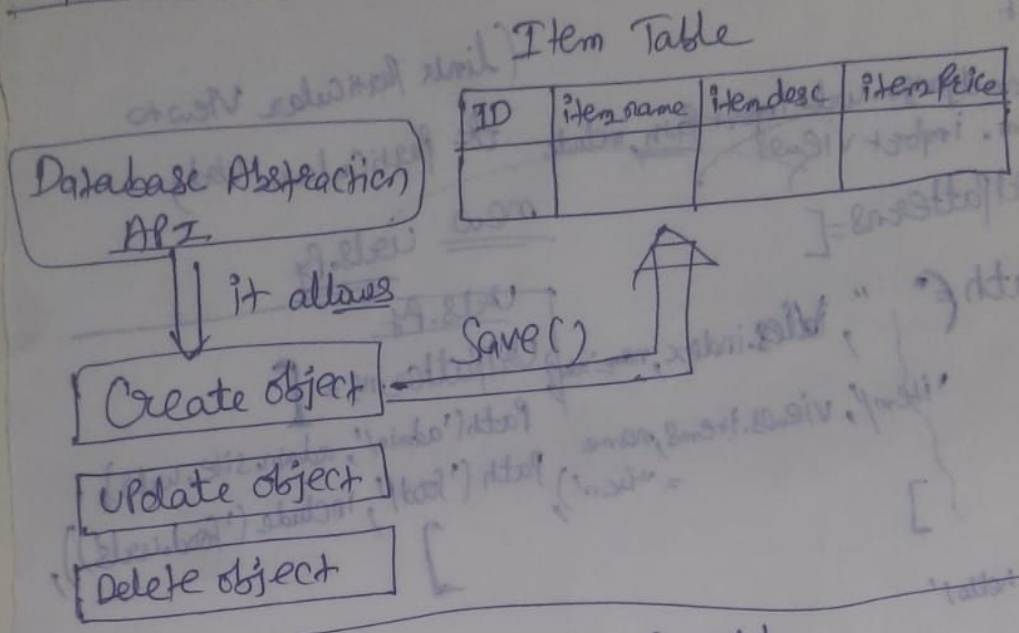
→ index.html

(1)

```
<body>
    <h1> This is just a template </h1>
</body>
```

Storing data in the database -

Steps involved



Using the interactive shell

```
Python manage.py shell
>>> from food.models import Item
>>> Item.objects.all()
>>> a = Item(item_name="Pizza", item_desc="Cheesy Pizza",
                item_price=20)
>>> a.save()
>>> a.id
>>> a.pk
>>> b = Item(item_name="Burger", item_desc="Cheesy Burger"
                item_price=20)
>>> b.save()
>>> b.id
>>> b.pk
```

(2)

```
from django.shortcuts import render
from django.http import HttpResponse
from django.urls import path
from .views import homepageview
from django.contrib import admin
from django.urls import path
from django.urls import path, include
```

conda env list

```
git init
git status
.gitignore
                requirements.txt
Pip freeze requirements.txt
git add -A
git Commit -m "Initial Commit"
```

What are Static files ? (Q)

Files such as images, JavaScript or CSS
How does Django know where the Static
files are located ?

```
INSTALLED_APPS = [
    'django.str
]

STATIC_URL = '/static/'
```

{% for item in itemlist %}

{{item.id}} -- {{item.item_name}}

{% endfor %}

{% for item in item_list %}

```
<ul>
    <li>
        {{item.id}} -- {{item.item_name}}
    </li>
</ul>
```

{% endfor %}

~~from django.shortcuts import render~~

```
from django.template import loader

def index(request):
    item_list = Item.objects.all()
    template = loader.get_template('food/index.html')
    Context = {'item_list': item_list,}
    return HttpResponse(template.render(Context, request))
return render(request, 'food/index.html', Context)
```

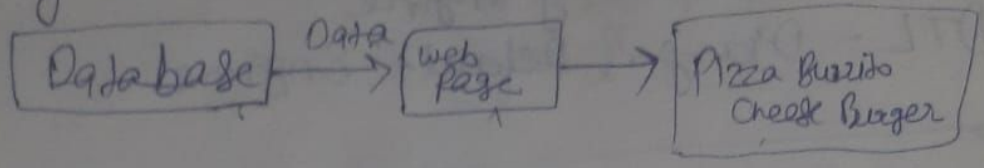
- Django Template language
- Templating Engine?
- Jinja2 - Templating Engine
- DTL - Django's Default Engine

Removing Hardcoded URLs

Namespacing

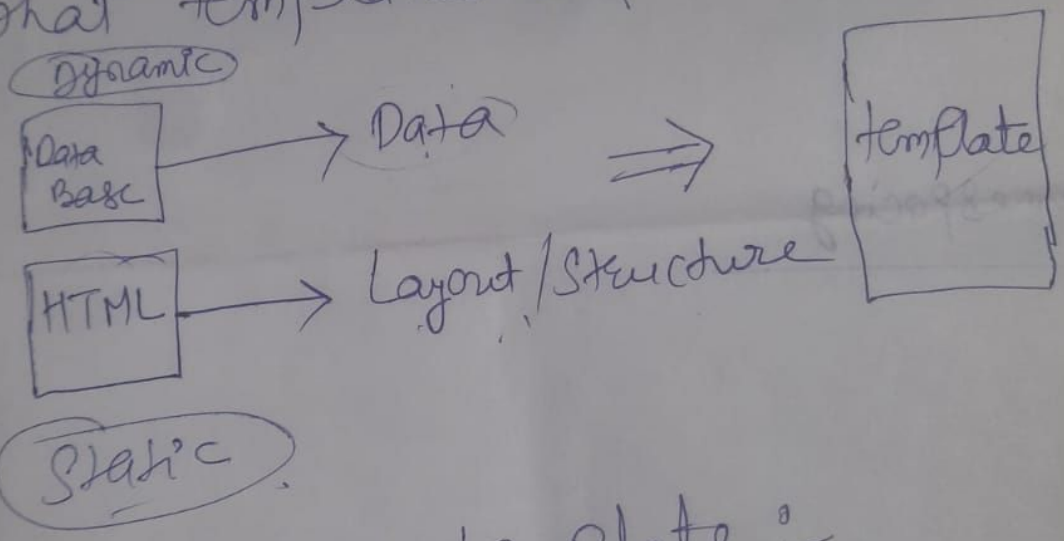
Templates in Depth :-

* why do we need templates?



- 1 - Pizza
- 3 - Burrito
- 4 - Cheese Burger

what templates do?



Let's Create a template :-

Django Template language

tags

```
{% for item in item_list %}
    <ul>
        <li>{{ item.id }} -- {{ item.item_name }}</li>
    </ul>
{% endfor %}
```

Variable

Storing data in the database. (1)

Steps Involved

item Table

ID	item_name	item_desc	item_price

Database Abstraction API

- Create Object
- Update object
- Delete object

Save()

Retrieving objects

How to retrieve data from database

Queryset Manager Model

Example: Queryset

Item

Model → Item

Manager → objects

Method → Item.object.all()

all()

Databases & Models in Django

What are models? (R)

Large websites needs large amount of data to be stored.

=) Data for websites is stored in a database.

(3) Data in a database is stored in form of tables / database tables.

(4) Models allow us to create database tables.

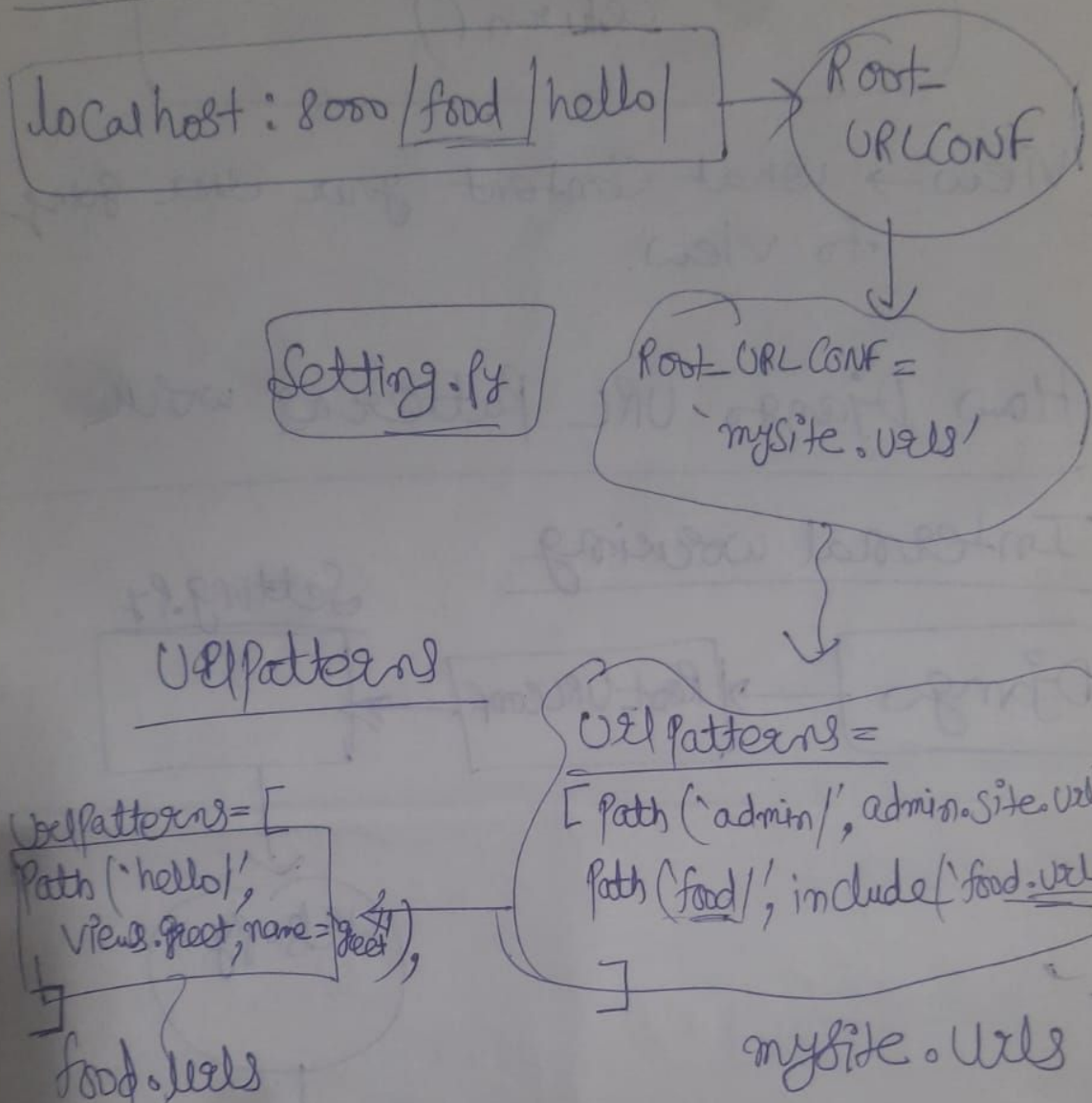
(5) Models are the blueprint which can be used to create database tables.

(6) Models are nothing but classes in Python.

(7) Models are created in the models.py file.

For example:- To create a model Student, simply create a class named Student.

Let's take our previous example ③



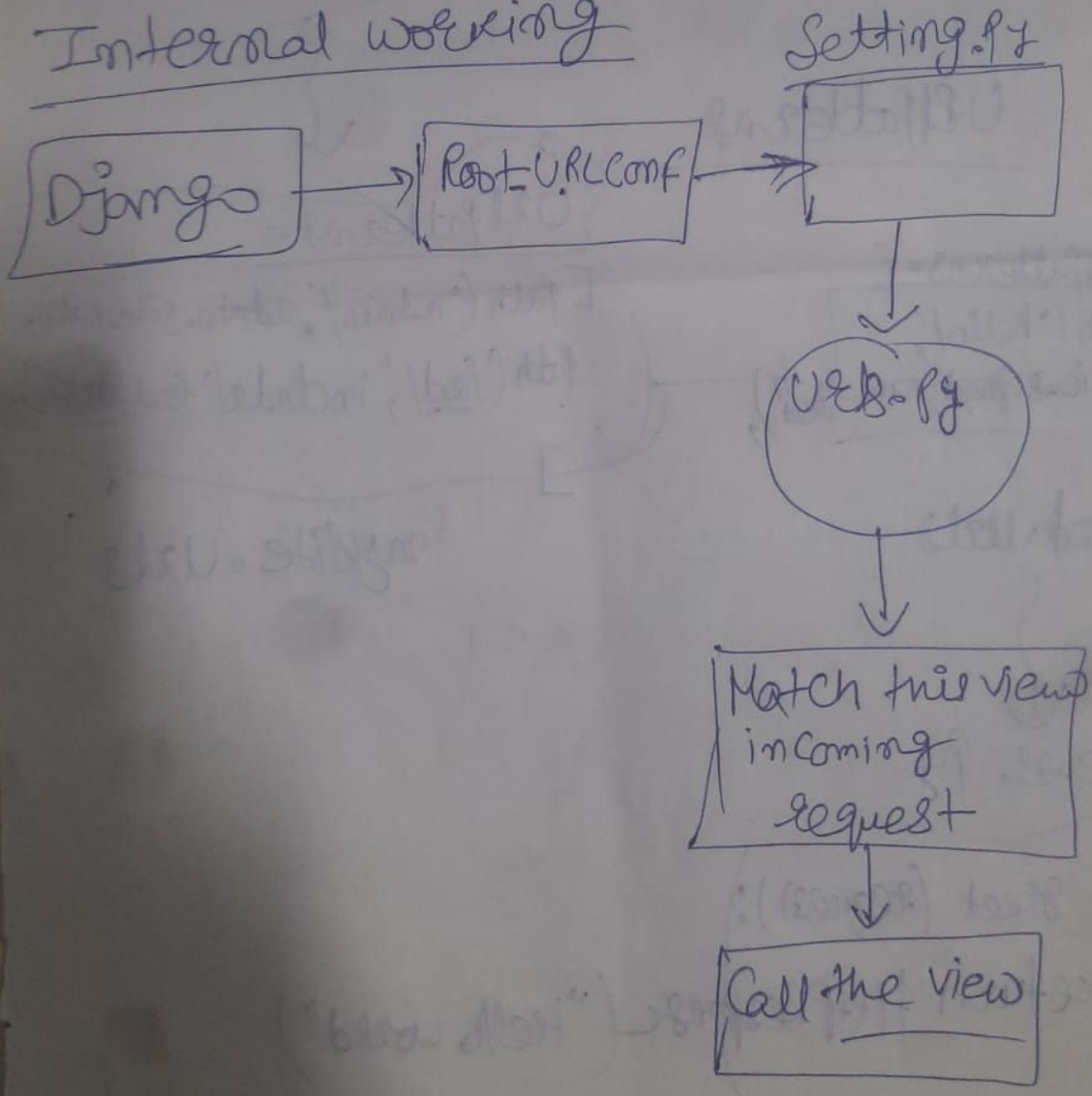
```
def viewname(request):
    return()
```

(1)

View → what Content your user going to view.

How Django URL Patterns work

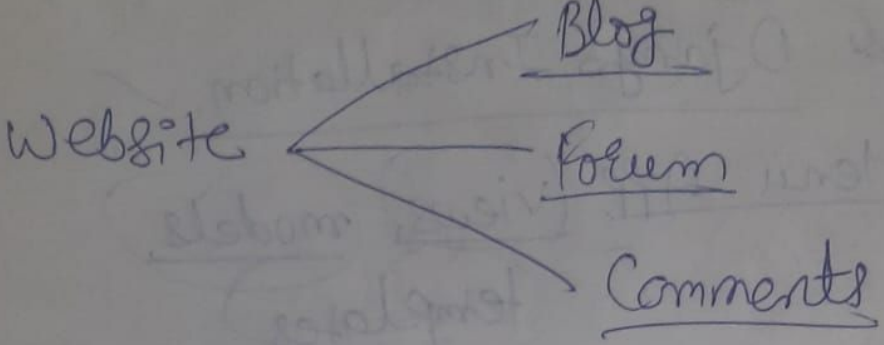
Internal working



How Django APPS actually work

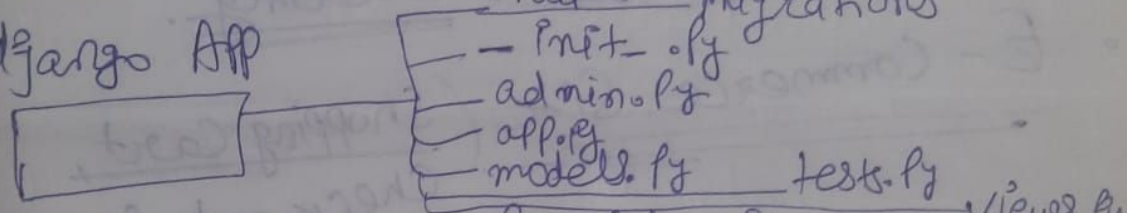
What is an app?

(h)

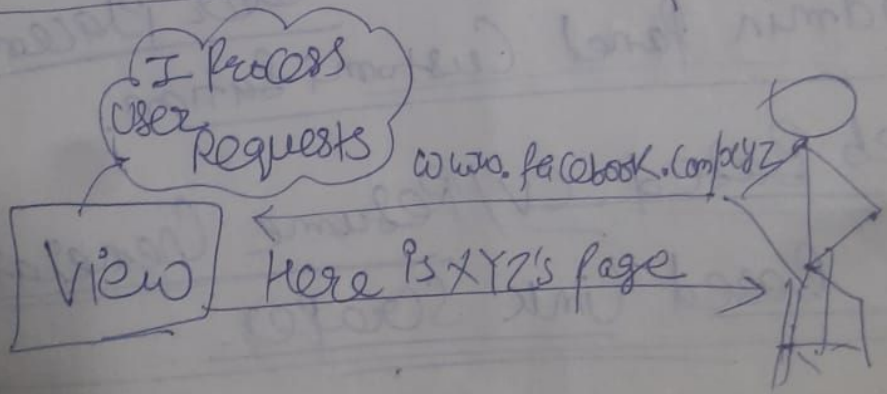


How apps look like in django

apps → It is a Collection of other files

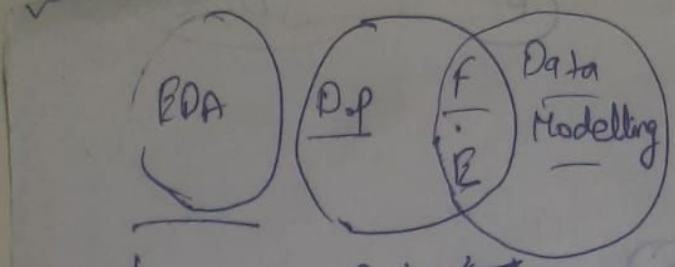


What is a views.py file?



How to write a view

written as a python Function.
This can be written inside the views.py file



Data Understanding → Data Transformation ⇄ Modelling

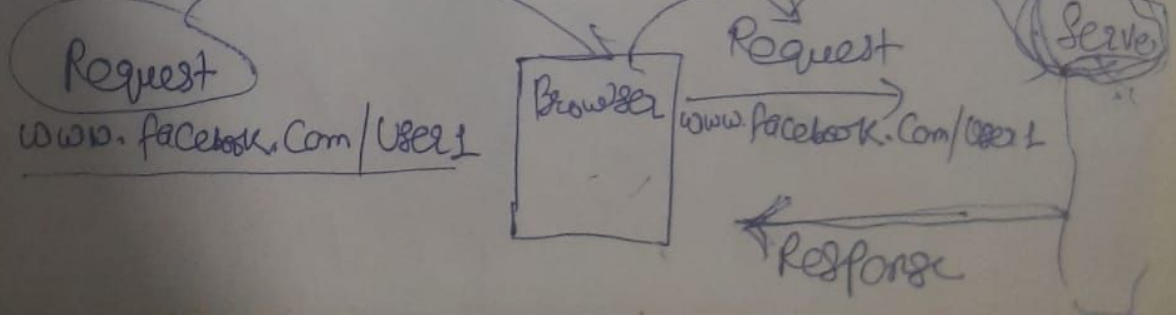
Django

A free and open source web Application framework

A framework is nothing but a Collection of Components which help us build sites faster.

It is a backend framework

Has automatic admin interface which supports CRUD.



Prerequisite & Inside the Course

Django

Understanding the indentation and Syntax of Python or prior Django knowledge is required

Programming Control Structures like Conditional loops.

HTML, CSS and Javascript are the fundamental building blocks of web development. Hence you must possess good knowledge about it.

What's inside the Course?

1. Setup & Understanding Django Projects
2. URLs, Routes & Views
3. Templates & Static Files
4. Data, Models & Relationships
5. Working with Forms
6. Class-based Views
7. File Uploads
8. Sessions
9. Cookies
10. Many Small Examples
11. A Real Project (Building a Blog)
12. Frontend + Admin Area
13. In-Depth Deployment Guide
14. E-Commerce Website

What is a framework?

- A framework is a particular set of rules, ideas or beliefs which you use in order to deal with problems or to decide what to do.

What is a django?

- Django is basically a high level Python web framework that enables the rapid development of web applications.
- Open-Source Python frameworks.
- Follows the model-view-template (MVT) architechural pattern
- Most Popular Python frameworks.

Why Django?

- Its fast & simple
- It's secure
- It suits any web application project
- Its well-established
- MVT support and object-oriented approach
- Built-in Authentication and Authorisation
- Packaging system.

Which companies uses django?

- YouTube
- Instagram
- Spotify
- Disqus
- The washington post
- Bitbucket

Installing Python & Django

Pip Freeze

Pip install django

Creating a Django Project -

Starting a Development Server

Port Change -

Python

MVT (A)

(Model View Template)

Django MVT-

- The MVT (Model View Template) is a Software design Pattern.
- It is a Collection of three important Components Model view and Template.
- Model : Model is going to act as the interface of your data
- View : The view is the User interface - what you see in your browser when you render a website
- Template : A template Consists of static parts of the desired HTML output as well as some special Syntax describing how dynamic Content will be inserted.

Flow Chart of MVT