Django

Django architecture :

Generally most of the frameworks follows MVC architecture (models - views - controller) but django follows MVT architecture (models - views - Templates)

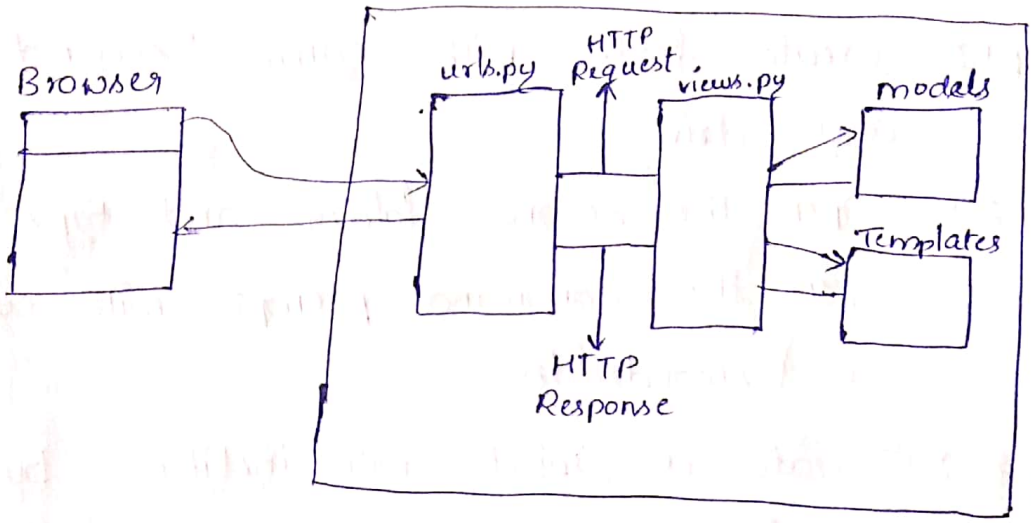
MVT :

M - Models - Database coding - M - Model

V - views - Business coding - C - controller

T - Template - Representation coding - V - views

Both MVC and MVT will perform same operations Both the difference is in the names.



Generally the project flow starts from the browser and it will go to urls.py file and find the corresponding view as per the requested url.

The Http request will go to views.py file and perform the operation in the view, as per the view design the request will go to models.py file and templates file and take the required data.

The Http response will come and display on the browser.

Scanned by CamScanner

Application: An application is specific task in the project we can create 'n' no of applications in the project if we want our website to handle the same type of task we require only one application, if we want our website to handle different kinds of un-related data then we can create different kinds of application

We use startapp subcommand to create django application startapp subcommand is part of manage.py file.

Step1: create folder with name 'second folder' in any drive.

Step2: Open the second folder and type cmd to open the command prompt with current location

E:\secondfolder >

Step3: create a project with institute by using start project sub command.

E:\secondfolder > django-admin startproject institute

Step4: change the path to manage.py file

E:\secondfolder > cd institute

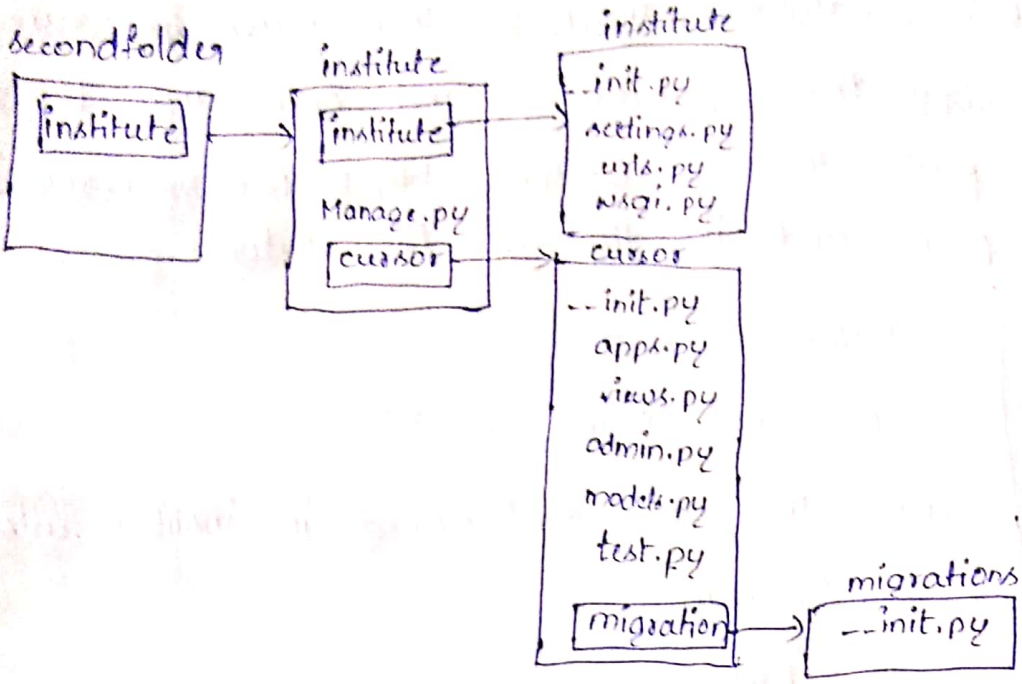
E:\secondfolder\institute >

Step5: Create an application with name cursor by using startapp command

E:\secondfolder\institute > python manage.py startapp cursor

Scanned by CamScanner

Now we can see the folder and files like below.



Step6: open the 2nd folder secondfolder in pycharm

File → Open → select secondfolder

Step7: open views.py file and write the code below

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

def home(request):
    x = 'welcome to Django framework'
    return HttpResponse(x)
```

Step8: open urls.py file and write the below code

```
from django.conf.urls import include,url
from django.contrib import admin
from courses import views
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^hello/', views.home)
```

Scanned by CamScanner

Step9: Go to command prompt and run the server

E:\secondfolder\institute > python manage.py runserver

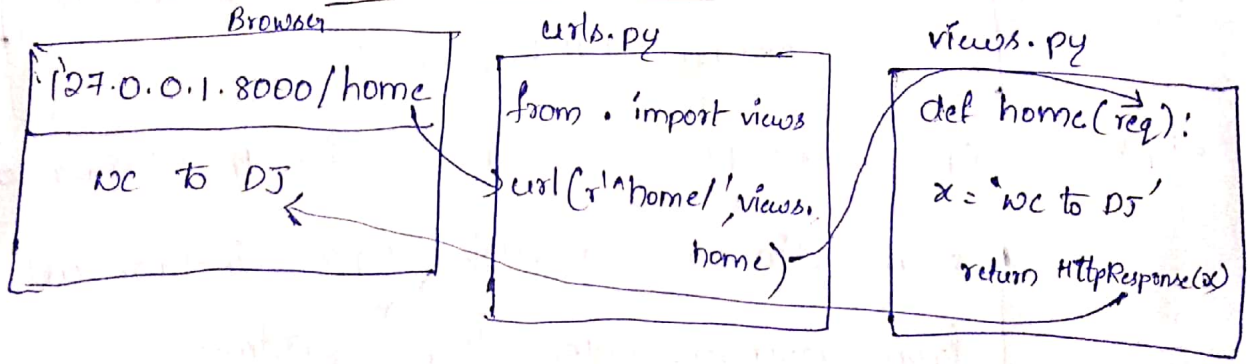
Now copy the Ip address and open any browser and paste the IP address (http://127.0.0.1:8000) in the url path and it displays like below.

① 127.0.0.1:8000
page not found (404)
using the URL conf defined in institute.urls.Django
1. ^admin/
2 hello

Now we have ~~to~~ to add/hello to the existing url and displays like below

(i) 127.0.0.1.8000/hello
welcome to Django framework

Working flow of the project:



Scanned by CamScanner

Views

* A view is a python function which takes HttpRequest and gives HttpResponse
* A view is responsible to perform any kinds of business logic
* A view will interact with models file to manage the database coding, it is also interact with template file to manage presentation coding and it is also interact with urls file to configure mapping between user urls and views.
* We write the views in views.py file which is default file in the application.
* We can write one or more views in single views.py file.

python(func)	Django(views)
def adding(a,b) ← i/p	def home(request): ← HttpRequest
x = a+b	x = 'welcome to DJ'
print(x)	return HR(x)
↓	↓
o/p	HttpResponse

* Now we will create a project by using multiple views in single views.py file

Step1: Create a folder with name 'thirdfolder'

Step2: Open the folder and type cmd to open the command prompt with current location

Scanned by CamScanner

E:\ thirdfolder >

Step 3 : create a project with name 'institute' by using start project command

E:\thirdfolder > django-admin startproject institute

Step4 : Change the path to manage.py file

E:\ thirdfolder > cd institute

E:\ thirdfolder\institute >

Step5 : create an application with name 'cursor' by using startapp command.

E:\ thirdfolder\institute>python manage.py startapp cursor

* We use the command to display files and folders

E:\ thirdfolder\institute > tree/F

Step6 : open the thirdfolder in the pycharm

Step7 : Go to views.py file and write the below code

```
from django.shortcuts import render
from django.http.response import HttpResponse
def home(request):
    x = '<h1> welcome to Durgasoft </h1>'
    return HttpResponse(x)
def services(request):
    s = '<h1> We will provide all courses </h1>'
    return HttpResponse(s)
def contact(request):
    c = '<h1> please go to main office to contact
                                Durga sir </h1>'
    return HttpResponse(c)
```

Scanned by CamScanner

```
def feedback(request):
    f = '<h1>we have got good feedback</h1>'
    return HttpResponse (f)
```

Step8 : Go to urls.py and write the following code

```
from django.conf.urls import include, url
from django.contrib import admin
from courses import views
urlpatterns = [
    url (r'^admin/', include (admin.site.urls)),
    url (r'^home/', views.home),
    url (r'^services/', views.services),
    url (r'^contact/', views.contact),
    url (r'^feedback/', views.feedback),
]
```

Step9 : Open the command prompt and run the server by using runserver command.

Now, copy the IP address/ and paste the IP address in the url path

E:/thirdfolder/institute7 python manage.py runserver

we have to add /home or /contact or/feedback so that we get the corresponding response

127.0.0.1.8000
Welcome to DJ

Every view will take http request and the view will redirect the request in different ways.

Scanned by CamScanner

HHpResponse() :

It will redirect the request to the browser as HHpResponse

views → Browser

Render( ): It will redirect the request to templates files

views → Templates(html files)

Redirect( ): It will redirect the request from one view to another views in the views.py file

view → view

* If we add /services it displays like below

127.0.0.1:8000/services
we will provide all courses

* If we add /contact it displays

← → ⓘ 127.0.0.1:8000/contact
Please go to main office to meet durga

if we add /feedback it display

← → ⓘ 127.0.0.1:8000/feedback
we have good feedback in market

Scanned by CamScanner

Create a project to display a ~~con~~ current date and time when the user sends 'date' request.

Step1 : Create a folder with the name thirdfolder

Step2 : open thirdfolder and type cmd in filepath to open a command prompt with current location

Step3 : create a project with name datepro

E:\ thirdfolder > django-admin startproject datepro

Step4 : change the path and create an application with name dateapp

E:\ thirdfolder > cd datepro

E:\ thirdfolder\ datepro> python manage.py startapp dateapp

Step5 : open thirdfolder in pycharm and go to views.py file and write the below code

```
from django.shortcuts import render
from django.http.response import HttpResponse
import datetime
date1 = datetime.datetime.now()
def dateview(request):
    x = "<h1> The current date and Time is
        {}</h1>".format(date1)
    return HttpResponse(x)
```

Scanned by CamScanner

Step 6 : Go to urls file

```
from django.conf.urls import url
from django.contrib import admin
from dateapp import views
urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^date/', views.dateview),
]
```

Step 7 : Run the server

```
E:\thirdfolder\datepro > python manage.py runserver 8890
```

Step 8 : Copy the address and open in browser

```
<- -> C (i) 127.0.0.1:8000/date
The current date and time is 2019-02-23 17:00:14.52
```

Format function :

It is used to format the "print" function in proper manner (or) in easy manner.

```
name = 'Kesav'
loc = 'hyd'
com = 'wipro'
print('He is', name, 'and he is working in', com, 'in' loc,)
```

(or)

```
print (He is {} and he is working in {} in {}.
        format(name, com, loc))
```

Here the order of values important in the format function

Scanned by CamScanner

If we take in different order we got wrong result like below

name = 'Kesav'

com = 'wipro'

loc = 'hyd'

print ("He is {} He is working in {} in {}.format(com, loc, name))

O/P: He is wipro. He is working in hyd in Kesav

* If we want to get proper result after changing the order values also we should used index math values in place holders.

print ("He is {2} He is working in {0} in {1}.format (com, loc, name))

Introduction to models

Till now, we developed 3 projects around views and urls files

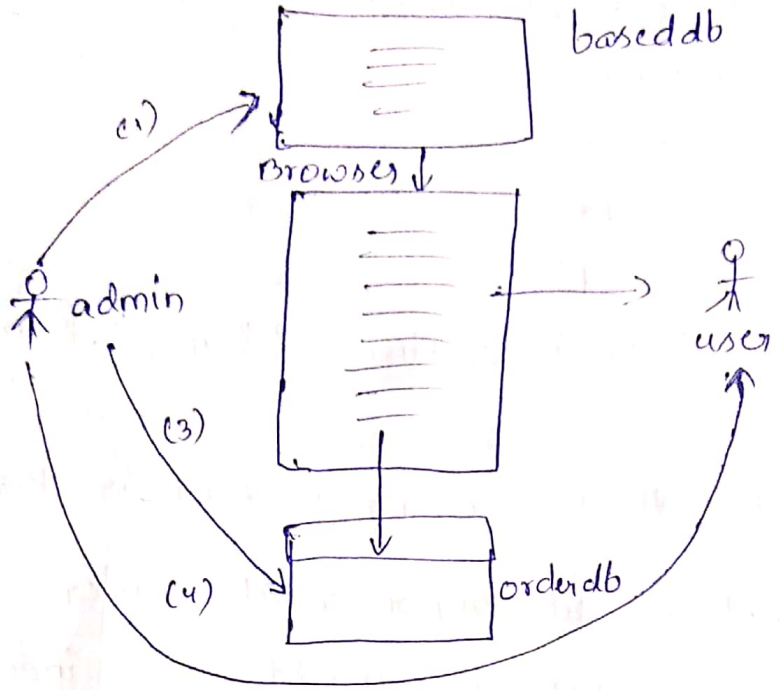
Generally every website must have database that means every django project must have models.py

* if any website using database i.e website is called database driven website

* django is well suited to create database driven websites.

* Both admin and users will use database in the website like below

Scanned by CamScanner



1. Admin or managers will store the data in the database and display on the browser
2. users or customers sees the data in the browser and place the order, this order store in the another database (order_db)
3. Admin (or) manager will keep on check the order_db for any new order
4. If they find any new order then they deliver the product to the user
5. django supports all the databases to create website like mysql, sql, sqlserver, oracle ..
6. whenever we run the server for the first time Django brings the default database sqlite3.

Scanned by CamScanner

7. sqlite 3 is recommended for small projects the projects which are not going to deploy in the production server generally, but in the real-time we always use either mysql or postgresql (or) sql server (or) oracle ..

Configure in mysql

We have to install pymysql module to connect mysql server from Django project.

⇒ We install pymysql through the command prompt.

```
pip install pymysql
```

⇒ go to mysql and create a database

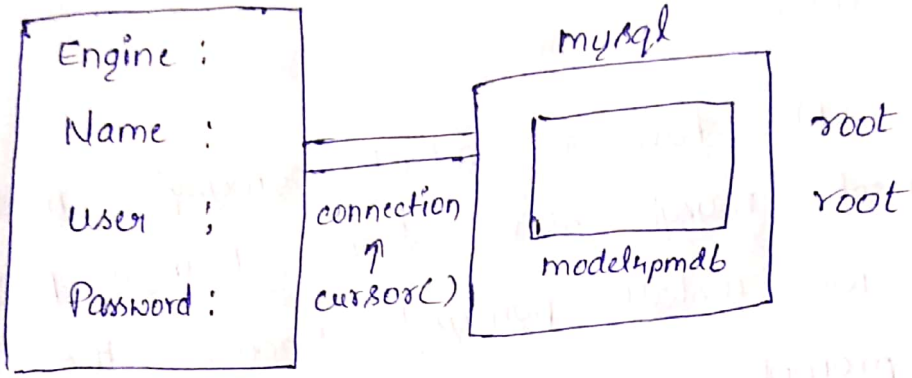
```
mysql > create database model4pmdb;
```

⇒ go to settings file and configure the mysql

```
DATABASES = {
    'default' : {
        'ENGINE' : 'django.db.backends.mysql',
        'NAME' : "model4pmdb",
        'USER' : 'root',
        'PASSWORD' : 'root',
    }
}
```

Scanned by CamScanner

⇒ Whenever we configure the database in Django settings file then automatically a connection will create between Django and mysql. Database like below.



⇒ We can execute the connection to verify whether we have given proper database details or not

Verifying the connection :

⇒ We go to python shell to verify that connection

⇒ We execute the connection by using 'cursor()' like below

c:\ febdjpm\modelpro > python manage.py shell

Scanned by CamScanner

$\boxed{a = connection.cursor()}$ from django.db import connection

⇒ The cursor() will return error. If we configure the database with wrong details.

⇒ If cursor() didn't return any error i.e means we configured the database properly.

⇒ If we configure database properly then we can create a model.

What is a model?

Model is a python class which contains the modelname, one or more field names and corresponding fieldtypes

* Whenever we run the model then the model name can convert as table name and the fieldnames convert as columns and the fieldtypes will convert as datatypes

```
class emp(models.Model)
    eno   = models.IF()
    ename = models.CF(max_length=20)
    sal   = models.IF()
```

eno, ename, sal → fieldname / columns

IF, CF, IF → field types → datatype

* We write the model ~~main~~ models.py file
* models.Model is the basemodel for every userdefined models
* We can write one or more models in single models.py file
* go to models.py file

```
from django.db import models
class emp(models.Model)
    eno   = models.IntegerField
    ename = models.CharField(max_length=50)
    sal   = models.IntegerField()
```

Scanned by CamScanner

=> If we want to create the database table for emp model then we should run makemigrations command and migrate command.

makemigrations

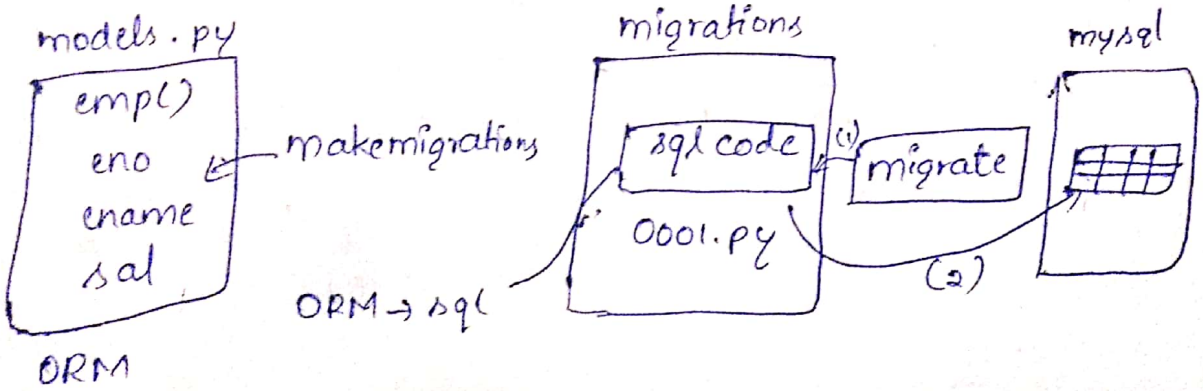
Whenever we run makemigrations then Django will goto models file and it will take the latest migrations are changes and then it will convert from ORM code into sql code and these sql code will save in the migrations folder.

E:\febdj4pm\modelpro > python manage.py makemigrations
migrations for 'modelapp'
0001_initial.py
- Create model emp

{0001_initial.py contains sql code}

migrate command

Whenever we run the migrate command then Django will go to the migrations folder and then it copies the sql code from a specific migration field and then it executes sql code in the database.



Scanned by CamScanner

E:\febdjupm\modelpro> python manage.py migrate

⇒ Now we can go to database and check the tables

mysql> use modelupmdb;

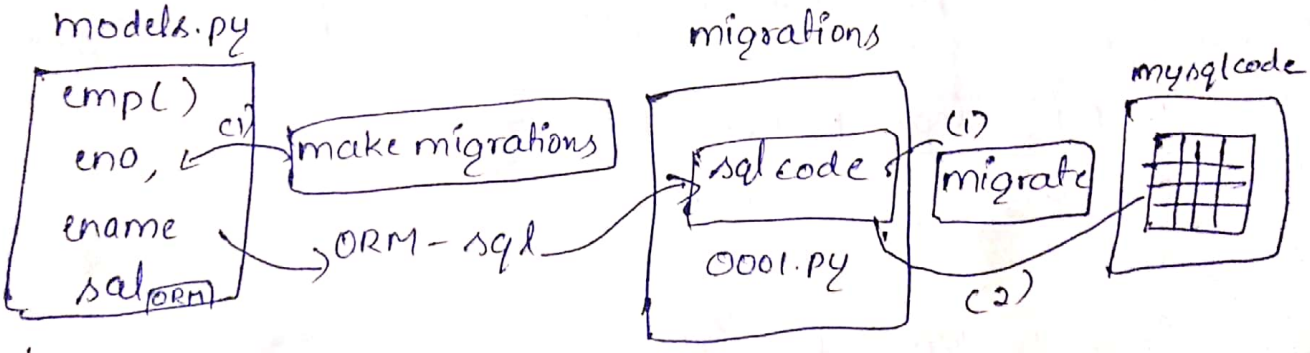
mysql > show tables;

sql migrate command

If we want to see the sql code of any specific migration field then we can use sqlmigrate command like below.

E:\febdjupm\modelpro > python manage.py sqlmigrate modelapp 0001

```
CREATE TABLE `modelapp_emp`( `id` integer AUTO_INCREMENT notnull PK;
        `eno` integer NOT NULL,
        `ename`, varchar(50) NOTNULL,
        `sal`, integer NOT NULL;
```



27/02/2019

Every model has the default field 'id'.

* Id is a "auto-increment" field that means Django will assign the values for id field automatically and the values starts from 1 and increment by 1

* Every field is not null field that means the fields will not allow null values (Every field is an mandatory field)

Scanned by CamScanner

* Id is also a primary key field that means it will not allow duplicate values.
* The table name is the combination of app name and the lowercase of model name by separating with (-) underscore. {modelapp_emp}

CURD operations by using ORM language:

* By default models data is hidden data
* we add string representation to the model so that the data is visible in python shell, admin site and browser

```
from django.db import models
class emp (models.Model)
        eno = models.IntegerField( )
        ename = models.CharField(max_length = 50)
        sal = models.IntegerField ( )
        def __str__(self):
                return self.ename
```

* we can perform the curd operations in different ways.

1. By using python shell
2. By using Admin site
3. By using Browser
4. By using Database

Performing CURD operations by using python shell:

* c - create (inserting data)

we have to import the emp model into python shell

we can insert the data in different ways

1. By using save()
2. By using create()

Scanned by CamScanner

* In save(), we take that data into object & then we save the object by using save()
* If we don't save the object then the data will not insert into
* If we save the object by using save() then internally insert command will execute in the B.
* If we create, we use create() to take the data into an object.
* Don't require to the save the object.
* If we use create() to take data then automatically in the database

E:\febdj4pm\modelpro > python manage.py shell.

```
>>> a = emp (eno =10, 'ename' = 'Sai', sal =10000
        a.save()
    b = emp (eno =11, ename = 'Gopi', sal = 20000)
        b.save()
    c = emp (eno =12, name = 'vijaya', sal =11000) c.save()
    d = emp.objects.create (eno =12, ename = 'satya', sal = 150000)
        d.save()
    e = emp.objects.create (eno =13, ename = 'Kesav', sal = 110000)
        e.save()
```

Note :- Go to database and check the data

MySQL > Select * from modelapp_emp;

id	eno	ename	sal
1	10	sai	10000
2	11	Gopi	20000
3	12	vijaya	11000
4	13	satya	15000
5	14	Kesav	11000

Scanned by CamScanner

R - Retrieve operation (Fetching Data):

We can fetch any specific value or all the values from the table

```
>>> a.ename
    'sai'
>>> b.sal
    20000
>>> c.eno
    12
>>> d.ename
    'Kesav'
>>> x = emp.objects.all()
>>> print(x)
[ <emp:sai>, <emp : nani>, <emp : satya>, <emp : kesav>]
```

U- update(modifying Data):

* We assign the new value to the specific field by using its objectname (variable name) and then we save the object by using save()
* If we save the object after modification then internally update command will executing the database.
* If we don't save the object modification then update command will not executing in the database

```
>>> a.sal = 30000
>>> a.save()
>>> b.ename = 'Gopi'
    b.save()
```

Note: Now we goto database to check the database table

Scanned by CamScanner

mysql > select * from modelapp_emp;

my

id	eno	ename	sal
1	10	sai	3000
2	11	Gopi	2000
3	12	vijaya	11000
4	13	satya	15000
5	14	Kesav	16000

D - Delete (Removing data):

Here we use get method to get a specific record into an object. And then we use delete method to delete that object which has the data

```
>> x = Emp.objects.get(eno=12)
>> x.delete()
>> y = emp.objects.get(ename='Kesav')
>>> y.delete()
```

Note: Go to Database and check the data

mysql > select * from modelapp_emp;

id	eno	ename	sal
1	10	sai	30000
2	11	gopi	20000
4	13	satya	15000

28/2/2019: model relationships

* Django models follow RDBMS like any other database
* Relationships are used to reduce the duplicate data or to avoid duplicate data
* Generally every table must have relationship in RDMS

Scanned by CamScanner

Like all other databases, Django models support different types of relationships

1. One-to-One Relationship
2. Many-to-One Relationship
3. Many-to-many relationship

Note: If we want to implement relationships then we required minimum two models.

One-to-one Relationship

In one-to-one relationship, one child record can depend on one parent record

* We use OneToOneField to implement one to one relationship between two models

Step1: foldername : febdj4pm

Step2: ~~app name :~~ project name : onetoonepro

Step3: app name : onetoone app

Step4: database name : oto4pm

Step5: Configure database in settings.py file

```
DATABASES : {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME' : 'oto4pm',
            'USER' : 'root',
            'PASSWORD: 'root',
        }
}
```

Scanned by CamScanner

Step6 : go to models.py file and create models

```
from django.db import models
class Student(models.Model):
    sname = models.CharField(max_length=20)
    doc = models.CharField(max_length=20)
    age = models.IntegerField()
    def __str__(self):
        return self.sname
class Course(models.Model):
    abc = models.OneToOneField(Student)
    cname = models.CharField(max_length=20)
    cfee = models.IntegerField()
    def __str__(self):
        return self.cname
```

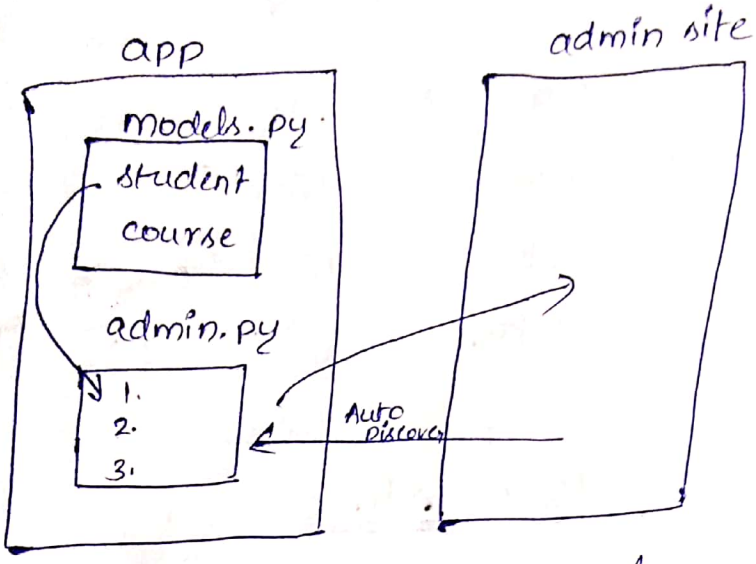
Step7 : Go to admin.py file

whenever we run the server, ah auto-discover() will come from admin site to admin.py file and execute the code which is available in the admin.py file.

if any code is available in the admin.py file then autoDiscover() will execute the code and the result of the code will reflect on admin site by by default admin.py file is an empty file that's why admin site doesnot have anything.

Scanned by CamScanner

AutoDiscover() will go to admin.py file only, but not to any other file in the application, like below



1. Here we import the models from models.py
2. We create admin models for the normal models to load all ~~colu~~ or required columns into admin site.
3. Here we have to register both normal and admin models with admin site, if we don't register with the admin site then the models will not load into admin site

```
from django.contrib import admin
from .models import Student, Course
class AdminStudent(admin.ModelAdmin):
    list_display = ['age', 'sname', 'loc']
class AdminCourse(admin.ModelAdmin):
    list_display = ['cname', 'cfee']
```

→ importing normal models.

→ Creating admin models for normal models to load all or required columns

Scanned by CamScanner

admin.site.register(Student, ~~Administrator~~ StudentAdmin)
admin.site.register(Course, AdminCourse)

→ registering both normal and admin models with admin site

Step8 : run makemigrations command

Go to tools tab and click on run manage.py task, a new window will open,

```
manage.py@onetoonepro > makemigrations
```

Step9 : run migrate command

```
manage.py@onetoonepro > migrate
```

Step10 : We have to create username and password to login to admin site.

* We use createsuperuser command

```
manage.py@onetoonepro > createsuperuser
```

username : django
password : 1234abcd

Step11 : run the server by using runserver command

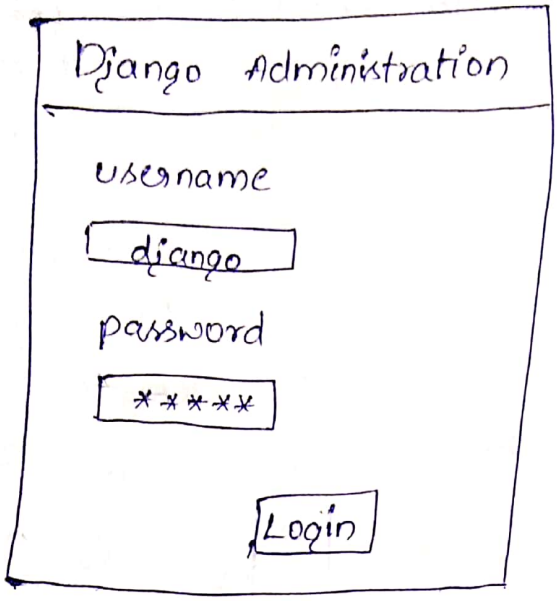
```
manage.py@onetoonepro > runserver 8303
```

Step12 : click IP address link so that a browser at open

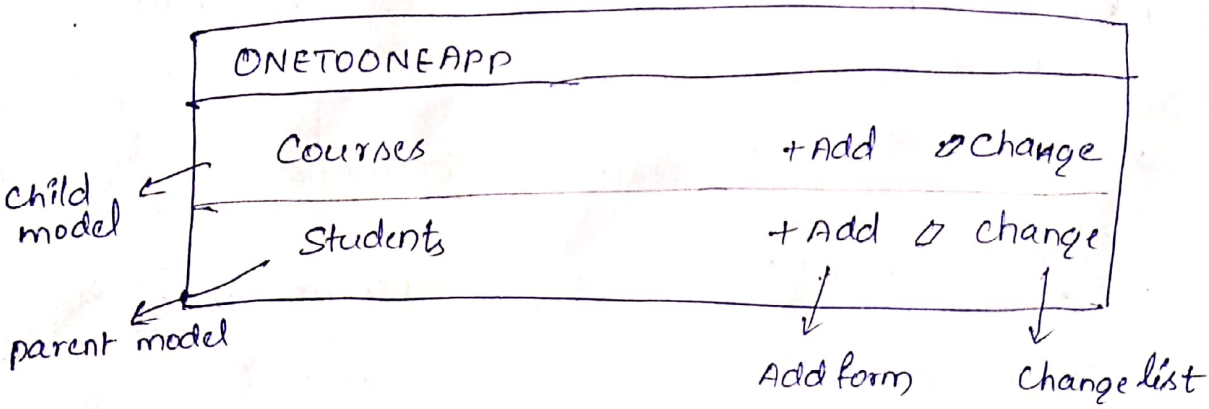
Step13 : Add /admin to the existing URL in the URL path and click enter admin login page will open.

Scanned by CamScanner

like below

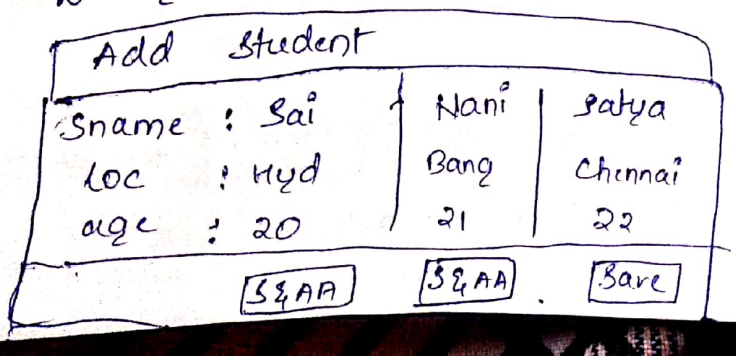


Step14: After giving username and password, if we click on log in button then admin site will open like below



* Add form is used to insert the data
* Change list is used to update, delete and retrieve the data

Step15: Click on Add form of student ~~form~~ model to give the data



Scanned by CamScanner

Step16 : Click on add form of course model to give the data

Add Course

Abc	:	Sai	Nani
Cname	:	python	Django
Cfee	:	2000	2000
		Save	Save

Step17 :- Now go to database and check the tables

mysql > select * from onetooneapp_student ;

id	sname	loc	age
1	sai	hyd	20
2	nani	bang	21
3	satya	Chennai	22

mysql > select * from onetooneapp_Course ;

id	cname	cfee	abcid
1	python	2000	1
2	django	2000	2

Step18 # In python child record is dependent on sai parent record and django child record is depending on nani parent record.

* Sathya parent record doesn't have any dependency.

Scanned by CamScanner

so satya parent record can be deleted or updated directly but sai and nani parent records cannot be updated or deleted directly (generally in the databases)

* If we want to manage the child records when the corresponding parent records are deleted or updated then we go for cascading rules

Cascading rules :

The cascading rules are used to manage the child records when the corresponding parent records are deleted or updated

* Django models support cascading rules as like any other database tables

Django	Database
DO_NOTING	NO_ACTION (It is Default rule)
SET_NULL	SET_NULL
SET(VALUE) ~~DEFAULT~~	SET_DEFAULT (value)
CASCADE (It is default value)	CASCADE

In database, no action is the default rule so we cannot delete or update that parent records

* In Django, cascade is the default rule so we can directly add can delete or update the parent records

* The cascading rules should be applied on the child table because these are used to

Scanned by CamScanner

manage child records.

DO_Nothing

If we set on_delete = models.DO_NOTHING then the parent records cannot be deleted or updated.

1. go to course model and modify abc field like below

```
abc = models.OneToOneField(Student, on_delete=models.DO_NOTHING)
```

2. run makemigrations, migrate and runserver commands
3. go to admin site and refresh
4. goto student model and select sai record and try to delete it
5. It will return error.

SET_NULL :

If you set ON_DELETE = models.SET_NULL then we can delete the parent records but the corresponding child records (abc_id) values becomes null values.

* We have set. null = True also for abc field in course because the abc field will not allow null values by default but SET_NULL gives null values that's why we set null = True

1. go to course model and modify abc field like below

```
abc = models.OneToOneField(Student, on_delete = models.
                                SET_NULL, null = True)
```

Scanned by CamScanner

2. run makemigrations, migrate and runserver commands
3. go to admin site and refresh it.
4. go to student model and select sai record and delete it.
5. The sai record can be deleted
6. go to database and check the tables

mysql > select * from onetoopeapp_student;

id	sname	loc	age
2	nani	bang	21
3	satya	Chennai	22

mysql > select * from onetoopeapp_course;

id	cname	cfee	abc_id
1	python	2000	null
2	django	2000	2

SET_DEFAULT (value)

If we set on_delete = models.SET(3) then we can remove the parent records but the corresponding child tables abc_id value will be replaced by 3.

* The value must be any one of the existing parent id

1. go to course model and modify abc field like below

```
abc = models.OneToOneField(Student, on_delete = models.SET(3),
                           null = True)
```

Scanned by CamScanner

2. run makemigrations, migrate and runserver commands
3. Go to admin site and refresh it.
4. Go to student model and select nani record and delete it.
5. Go to database and check the data

mysql > select * from onetooneapp_student;

id	sname	loc	age
3	satya	chennai	22

mysql > select * from onetooneapp_course;

id	cname	cfee	abc_id
1	python	2000	NULL
2	django	2000	3

CASCADE:

cascade If we set on_delete = models.CASCADE then we can delete the parent records but the corresponding child records also will be deleted.

If we don't specify any cascading rule also then by default cascade rule will work.

1. Go to course model and modify abc field like below

```
abc = models.OneToOneField(Student, on_delete=models.
                                        CASCADE,
                                        null=True)
```

2. run makemigrations and migrate and runserver commands.
3. Go to admin site and refresh it.

Scanned by CamScanner

4. Go to student model and select satya record and delete it

5. Satyarecord will be deleted and also the corresponding django child record also deleted

6. Go to database and check the tables.

mysql > select * from onetooneapp_student;

Empty set

mysql > select * from onetooneapp_course;

id	cname	cfee	abc_id
1	python	2000	NULL

6/3/2019

Many-to-one Relationship

1. Here multiple children can depend on single parent
2. Many-to-one relationship supports cascading rules like one-to-one relationship
3. We use foreign key() to implement many to one relationship because foreign key allows duplicate values

Step1 :- foldrname : febdjupm

Step2 :- projectName : mtopro

Step3 :- app Name : mtoapp

Step4 : databasename : mto4pmdb

Step5 : configure Database in settings.py file

Scanned by CamScanner

```
DATABASES = {
        'default' : 'django.db.backends.mysql',
        'NAME' : 'mto4pmdb',
        'USER' : 'root',
        'PASSWORD: 'root',
    }
}
```

Step6 : create models in models.py file

```
from django.db import models
class Student(models.Model):
    sname = models.CharField(max_length=50)
    sloc = models.CharField(max_length=50)
    age = models.IntegerField()
    def __str__(self):
        return self.sname
class Course(models.Model):
    student = models.ForeignKey(Student)
    cname = models.CharField(max_length=50)
    cfee = models.IntegerField()
    def __str__(self):
        return self.cname
```

Step7 : Go to admin.py file

Scanned by CamScanner

```
from django.contrib import admin
from .models import Student, Course
class AdminStudent(admin.ModelAdmin):
    list_display = ['sname', 'sloc', 'age']
class AdminCourse(admin.ModelAdmin):
    list_display = ['cname', 'cfee']

admin.site.register(Student, AdminStudent)
admin.site.register(Course, AdminCourse)
```

Step8 :- run makemigrations, migrate commands.

Step9 :- run createsuperuser command

username : django

password : 1234abcd

Step10 :- run runserver command, click on the IPAddress link

Step11 :- A new browser will open with current IPADDRESS.

Step12 :- add/Admin to the existing url and click enter, admin login page will open.

Step13 :- Enter username and password and click on login button, admin home page will open.

Step14 :- Click on Add form of student model and enter like below

Scanned by CamScanner

Add Student

Sname :	Sai	Nani	Satya
Sloc :	hyd	Banglr	Chennai
age :	20	21	22
	S&AA	S&AA	S&AA

Step15 : Click on Add form of Course model and enter data like below

Add Course

student :	sai	Sai	Nani
Cname :	python	Django	RestAPI
Cfee :	2000	2000	2000
	S&AA	S&AA	S&AA

Step16 : Now go to database and check the tables

mysql > select * from mtoapp_student;

id	sname	sloc	age
1	sai	hyd	20
2	nani	banglr	21
3	satya	chennai	22

mysql > Select * from mtoapp_Course;

id	cname	cfee	student_id
1	python	2000	1
2	django	2000	1
3	RestAPI	2000	2

Scanned by CamScanner

Step17 :- Here, python and django child records are dependent on a single parent sai record that means multiple child records are dependent on single parent record

→ RestAPI child record is dependent on Nani parent record

→ No child records are dependent on satya parent record.

→ We can manage the child records when the corresponding records are deleted or updated by using cascading rules same like one-to-one relationship

Many-to-Many Relationship

In this relationship, multiple child records can depend on multiple parent records

→ Here we use ~~Many-To-Many Field~~ ManyToManyField to implement manytomany relationship

→ It supports only the defaulet cascading rule i.e cascade ('CASCADE') if we try to perform the remaining cascade rules it will throw error

Step1 : FolderName : febdj4pm

Step2 : ProjectName : mtompro

Step3 : appname : mtomapp

Step4 : databasename : mtom4pmdb

Scanned by CamScanner

Step5 : Configure DATABASE in settings.py file

Step6 : Go to models.py file and write the models.

```
from django.db import models

class Student(models.Model):
    sname = models.CharField(max_length=50)
    sloc = models.CharField(max_length=50)
    age = models.IntegerField( )
    def __str__(self):
        return self.sname

class Course(models.Model):
    student = models.ManyToManyField(Student)
    cname = models.CharField(max_length=50)
    cfee = models.IntegerField( )
    def __str__(self):
        return self.cname
```

Step7 : Go to admin.py file

Note:

```
from django.contrib import admin
from .models import Student, Course
class AdminStudent(admin.ModelAdmin):
    list_display = ['sname', 'sloc', 'age']
class AdminCourse(admin.ModelAdmin):
    list_display = ['cname', 'cfee']
admin.site.register(Student, AdminStudent)
admin.site.register(Course, AdminCourse)
```

Scanned by CamScanner

Step8 : run makemigrations, migrate, createsuperuser, runserver

Step9 : login to admin page

Step10 : Click on addform of student and enter data below

add student

sname :	sai	nani	satya
sloc :	hyd	bangl	Chennai
age :	20	21	22
	S&AA	S&AA	Save

Step11 : Click on addform of Course and enter data below

Add Course

~~Snam~~.

Student :	Sai Nani Satya	Sai Nani Satya	Sai Nani Satya
cname :	python	django	RestAPI
cfee :	2000	2000	2000
	S&AA	S&AA	Save

Step12 : Go to database and check the data

mysql > select * from mtomapp_student;

id	sname	sloc	age
1	sai	hyd	20
2	nani	bangl	21
3	satya	Chennai	22

Scanned by CamScanner

mysql > Select * from mtomapp-course;

id	cname	cfee
1	python	2000
2	Django	2000
3	RestAPI	2000

mysql > Select * from mtomapp-course-student;

id	course_id	student_id
1	1	1
2	1	2
3	2	3
4	2	1
5	2	2
6	3	1

Note :- Django creates an intermediate table to store the relationships.

→ ManyToMany Relationship will not supports all cascading rules, it supports only cascade (CASCADE) rule

→ If we delete sai parent record then corresponding relationships will disconnected from the intermediate table, the child records of sai parent are not deleted

→ If we delete all parent records then all relationships will be disconnected from intermediate table, all child records are available

→ Even though we delete the parent records, the child records are not deleted.

Scanned by CamScanner

Model Inheritance

1. It is the process of deriving the fields from one model to another model
2. Model also supports different types of inheritance
   1. Abstract Model Inheritance
   2. Proxy Model Inheritance
   3. Multilevel Model Inheritance
   4. Multiple Model Inheritance

Abstract model Inheritance :

1. Abstract model is a model which has fields of other non-abstract models.
2. Abstract model will give the fields to non-abstract model at runtime
3. We have to create meta model to make a model as abstract model and we have to set abstract = True in the meta model.
4. We can derive the fields from one abstract model into n number of non-abstract models.
5. Django will not create database table for abstract model because abstract model doesn't have its own fields, it has the fields of non-abstract models.
6. The abstract model should be created before creating non-abstract models
7. We should use abstract model name as a base or parent class to all non-abstract models.

Scanned by CamScanner

so that abstract model it will give fields to non-abstract models

8. If we don't use abstract model name as a parent model for any specific non-abstract model then it will not give the fields.

Step1 :- Foldername : febdj4pm

Step2 :- ProjectName : abpro

Step3 :- App Name : abapp

Step4 :- database name : ab4pmdb

Step5 :- configure database in the setting.py file

Step6 :- Create models in model.py

```
from django.db import models
class Data(models.Model):
    name = models.CharField(max_length=50)
    loc = models.CharField(max_length=50)
    class Meta:
        abstract = True
class Emp(data):
    sal = models.IntegerField()
    def __str__(self):
        return self.name
class customer(data):
    sales = models.IntegerField()
    def __str__(self):
        return self.name
```

Scanned by CamScanner

```
class student(data):
    marks = Models.IntegerField()
    def __str__(self):
        return self.name
```

Step7 :- Go to admin.py file

```
from django.contrib import admin
from .models import Emp, Customer, Student
class AdminEmp(admin.ModelAdmin):
    list_display = ['name', 'loc', 'sal']
class AdminCustomer(admin.ModelAdmin):
    list_display = ['name', 'loc', 'sales']
class AdminStudent(admin.ModelAdmin):
    list_display = ['name', 'loc', 'marks']
admmin.site.register(Emp, AdminEmp)
admin.site.register(Custorm, AdminCustomer)
admin.site.register(Student, AdminStudent)
```

Step 8 : run makemigrations, migrate, createsuperuser, runserver

User : django
Password : 1234abcd

Step 9 : Go to login page or admin site

Step10 : Click on add form Customer and enter data

Add Customer

name : sai

loc : hyd

sales : 10000

Save

Scanned by CamScanner

Step11 : click on add form of emp model

add emp

name : nani

loc : banglr

sal : 25000

[Save]

Step12 : click on add form of student

add student

name : satya

loc : Chennai

marks: 99

[Save]

Step13 : Go to database and check the data

mysql > Select * from abapp_customer;

id	name	loc	sales
1	sai	hyd	10000

mysql > Select * from abapp_emp;

id	name	loc	sal
1	nani	banglr	25000

mysql > Select * from abapp_student;

id	name	loc	marks
1	satya	Chennai	99

1. Proxy model Inheritance

Proxy model is model to manage which is used to manage or control non-abstract model.

Scanned by CamScanner

* We can perform all CURD operations on abstract model by using Proxy model, that means if we insert the data in proxy model then the same data will insert into non-abstract model also, like this remaining all CURD operations can be performed.

* we have to create Meta model and set proxy=True to make any model as Proxy model.

* Django will create database table for proxy model. Generally one proxy model can manage only one abstract model

Step1 :- Foldername : febdj4pm

Step2:- Project Name : proxypro

Step3:- App Name : proxyapp

Step4:- Database Name: proxy4pmdb

Step5:- Configure database in the setting.py file

Step6:- Create models in model.py file

```
from django.db import models
class Emp(models.Model):
    ename = models.CharField(max_length=50)
    salary = models.IntegerField()
    location = models.CharField(max_length=50)
    def __str__(self):
        return self.ename

class EmpProxy(Emp):
```

Scanned by CamScanner

```
class Meta:
    Proxy = True
```

Step7 :- Go to admin.py file

```
from django.contrib import admin
from .models import Emp, EmpProxy

class AdminEmp(admin.ModelAdmin):
    list_display = ['ename', 'salary', 'location']

class AdminEmpProxy(admin.ModelAdmin):
    list_display = ['ename', 'salary', 'location']

admin.site.register(Emp, AdminEmp)
admin.site.register(EmpProxy, AdminEmpProxy)
```

Step8 :- Run makemigrations, migrate, createsuperuser and runserver commands.

Step9 :- Login to admin site

Inserting data

Step10 :- click on addform of emproxy model and enter data like below

Add EmpProxy

ename :	sravya	Satya
salary :	10000	20000
location :	mumbai	Banglr
	SEnA	SEnA

Scanned by CamScanner

Now we can see the same data in emp model also

Updating data

Step11: Click on change list of empproxy and change salary of sravya.

* Click on satya name then it will open a form like below

Change empproxy	
Ename :	satya
salary :	30000
location :	Banglr
X Delete	Save

* If we click on save button then the data will update in the empproxy model and the same data will update in emp model also

Deleting data :

Step12: Click on change list of emp proxy model and select satya record and delete it

* The same satya record will also delete from emp model

Difference between abstract and proxy models

Abstract	Proxy
1. Abstract is a model which has common fields of all non-abstract models.	1. Proxy is a model which is used to control or manage the non-abstract model.

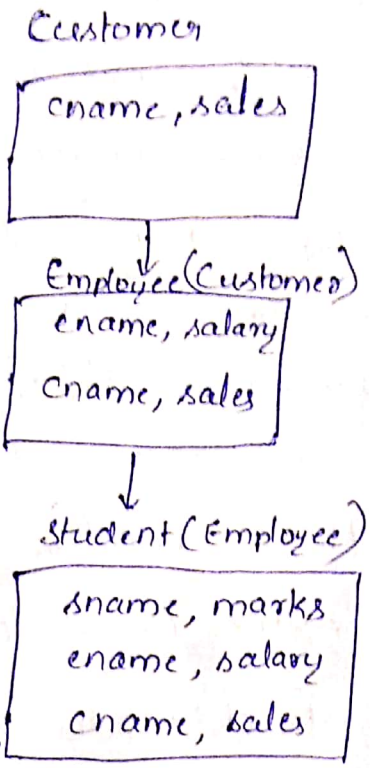
Scanned by CamScanner

2. Database will not create a table for Abstract model.	2. Database will create table for Proxy model.
3. One abstract model can take common fields of multiple non abstract models, that means it can communicate with multiple non-abstract models.	3. One proxy model can manage only one non-abstract model
4. Number of fields in both abstract and non-abstract models may be same or different	4. Number of fields in bon abstract and non-abstract models must be same.
5. Abstract model should be created before created non-abstract model	5. Proxy model should be created after creating non-abstract model
6. We set abstract = True in the meta class of abstract model	6. We set proxy = True in the meta class of proxy model
7. We cannot modify or update the data of non-abstract model by using abstract model.	7. We can update or modify the data of non-abstract model by using proxy model.

Scanned by CamScanner

Multi level model Inheritance

In this inheritance, we can derive the fields from one model to another model and we can derive the fields from this model into another model like "one after the other" process



Step1: Folder Name : 'febdj4pmd'

Step2: Project Name: multilevelpro

Step3:- App Name : multilevelapp

Step4:- Database Name: multilevel4pmdb

Step5: Configure database in settings.py file.

Step6:- Create models.py file

```
from django.db import models

class Customer(model.Model)
        cname = models.charfield(max_length=50)
        sales = models. ~~charfield~~ IntegerField()
        def __str__(self):
                return self.cname

class Employee(~~model.Model~~ Customer)
        ename = models. charfield(max_length=50)
        salary = models. IntegerField()
        def __str__(self):
                return self.cname
```

Scanned by CamScanner

```
class Student (models.Model):
        sname = models.CharField(max_length=50)
        marks = models.IntegerField()
        def __str__(self):
                return self.sname
```

Step 7 :- Go to admin.py file

```
from django.contrib import admin
from .models import Customer, Employee, Student

class AdminCustomer (model.ModelAdmin)
        list_display=['cname', 'sales']

Class AdminEmployee (model.ModelAdmin)
        list_display = ['ename', 'salary']

class AdminStudent (model.ModelAdmin)
        list_display = ['sname', 'marks']

admin.site.register (Customer, AdminCustomer)
admin.site.register (Employee, AdminEmployee)
admin.site.register (Student, AdminStudent)
```

Step 8 :- Run makemigrations, migrate, createsuperuser and runserver commands.

Step 9 :- Login to admin.site

Step 10 :- click on addform of student model and enter data like below.

Scanned by CamScanner

Add Student

ename : sai
sales : 10000

ename : nani
salary : 30000

sname : satya

marks : 99

Save

Step11 :- Go to database and check the data

mysql > select * from multilevelapp_customer;

id	ename	sales
1	sai	10000

mysql > select * from multilevelapp_employee;

customer_ptr_id	ename	salary
1	nani	30000

mysql > select * from multilevelapp_student;

employee_ptr_id	sname	marks
1	satya	99

Multiple model Inheritance

In multiple model Inheritance, one child model will take the fields from more than one parent model here, the default id fields from all parent models will derive into child model like

Scanned by CamScanner

normal fields

* After deriving, the child model will have id fields more than one which becomes clashes between id fields because the model (table) should have one field with name that means the model should not allow duplicate fields
* To solve the problem, we change the name of default id fields in all parent models
* At runtime the child model will have all fields from all parents so that if we go to child model and we can give the data.

Step1 :- FOLDER NAME : febdjupm

Step2 :- Project Name : multiplepro

Step3 :- App Name : multipleapp

Step4 :- Database Name : multipleupmdb

Step5 :- Go to settings file and configure the database

Step6 :- Go to models file and create models.

```
from django.db import models
class Customer(models.Model):
    cname = models.CharField(max_length=50)
    sales = models.IntegerField()
    def __str__(self):
        return self.cname
class Employee(models.Model):
    ename = models.CharField(max_length=50)
```

Scanned by CamScanner

```
    salary = models.IntegerField()
        def __str__(self):
            return self.ename

class Student (Employee, Customer):
    sname = models.CharField(max_length=50)
    marks = models.IntegerField()
        def __str__(self):
            return self.sname
```

Step 7 :- Go to admin.py file

```
from django.contrib import admin
from .models import Customer, Employee, Student

class AdminCustomer(admin.ModelAdmin):
 list_display = ['cname', 'sales']
class AdminEmployee(admin.ModelEmployee):
 list_display = ['ename', 'salary']

class AdminStudent(admin.ModelAdmin):
list_display = ['cid', 'eid', 'cname', 'sales',
                'ename', 'salary', 'sname', 'marks']

admin.site.register(customer, AdminCustomer)
admin.site.register(Employee, AdminEmployee)
admin.site.register(Student, AdminStudent)
```

Scanned by CamScanner

Step 8 :- Run makemigration, migrate, Createsuperuser and runserver commands.

Step 9 :- Login to admin.site

Step 10 :- Click on addform of student model and enter the data.

Add.student

cname :	sai
sales :	10000
ename :	nani
salary :	40000
sname :	satya
marks :	99

* If we click on save button then cname and sales fields will go to customer model and save the data
* ename and salary fields go to employee model and save the data
* sname and marks fields will save the data in student model.

Fieldname restrictions

* django will not allow us to write the fieldname in all the ways because it follows some restrictions like below

1. We should not use python Keywords as fieldnames, but we can use database Keywords like below

Scanned by CamScanner

ename = models.CharField(max_length = 50) — right
delete = models.CharField(max_length = 50) — right
del = models.CharField(max_length = 50) — wrong

2.) we can use only underscore special character, it will not allow any other special characters, like below
emp_name = models.CharField(max_length) - right
emp&name = models.CharField(max_length) - wrong

3.) we cannot use multiple underscores in a sequence but we can use multiple underscores individually, like below
emp___name = models.CharField(max_length = 50) - wrong
emp_old_name = models.CharField(max_length = 50) - right

4.) we cannot start fieldname with digit but we can use digits in the fieldname, like below
ename123 = models.CharField(max_length = 50) = right
e123name = models.CharField(max_length = 50) = right
123ename = models.CharField(max_length = 50) = wrong

5.) we cannot use space inside the fieldname, like below
emp name = models.CharField(max_length = 50) = wrong
empname = models.CharField(max_length = 50) = right

List of fieldtypes

1. CharField()
2. TextField()
3. IntegerField()
4. BigIntegerField()
5. PositiveIntegerField()

Scanned by CamScanner

6. ImageField( )
7. FileField( )
8. FilePathField( )
9. OneToOneField( )
10. ManyToManyField( )
11. AutoField( )
12. URLField( )
13. DateField( )
14. DateTimeField( )
15. TimeField( )
16. FloatField( )

Model Manager

A manager is an interface between the model and corresponding methods

* Every model has its own manager
* The default manager name is objects
* We can change the manager name by using ManagerFields

```
class Product(models.Model):
    pid = models.AutoField(primary_key = True)
    pname = models.CharField(max_length = 50)
    pcost = models.IntegerField( )
    pclass = models.CharField (max_length = 50)
    obj = models.Manager ( )
    def __str__(self)
        return self.pname
```

python shell

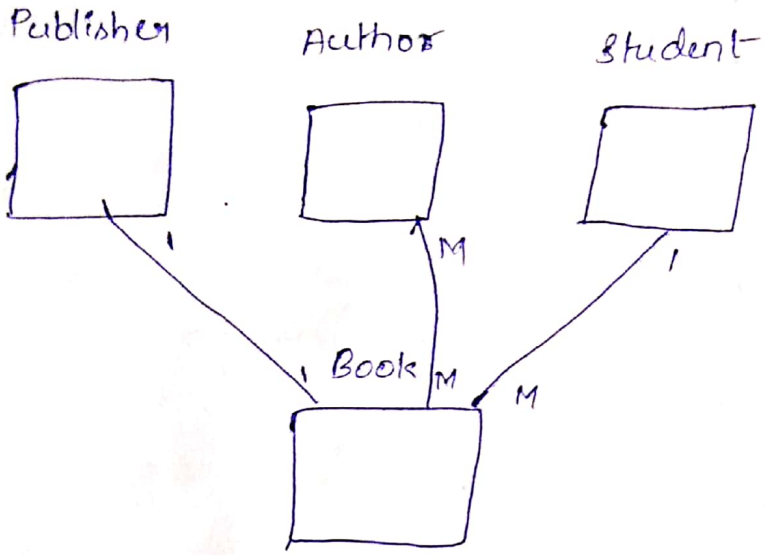
```
a = Product.obj.create(pname = 'Lap', pclass = 'High', pcost = 10000)
```

Scanned by CamScanner

x = Product.obj.get (pname = 'Lap')

x = Product.obj.filter (pname = 'Lap')

Creating a project by using all relationships



Step1 : ~~Foldername~~ Projectname : publisherpro

Step2 : App Name : publisherapp

Step3 : Database Name : publisherupmdb

Step4 : configure database in settings.py file.

Step5 : create models.py file

```
from django.db import models
class Publisher(model.Model):
    publisher_name = models.CharField(max_length=50)
    publisher_email = models.EmailField(max_length=50)
    def __str__(self):
        return self.publisher_name
class Author(model.Model):
    Author_name = models.CharField(max_length=50)
    Author_Email = models.EmailField(max_length=50)
```

Scanned by CamScanner

```
    def __str__(self):
        return self.author_name
class Student(models.Model):
    sname = models.CharField(max_length=50)
    mobileno = models.IntegerField()
    def __str__(self):
        return self.sname
class Book(publisher, Author, Student):
    Book_name = models.CharField(max_length=50)
    Book_cost = models.IntegerField()
    def __str__(self):
        return self.book_name
```

Step 6 :- Go to admin.py

```
from django.contrib import admin
from .models import Publisher, Author, Student, Book
class AdminPublisher(admin.ModelAdmin):
    list_display=['publisher_name', 'publisher_email']
class AdminAuthor(admin.ModelAdmin):
    list_display = ['author_name', 'author_email']
class AdminStudent[admin.ModelAdmin]:
    list_display = ['sname', 'mobileno']
class AdminBook[admin.ModelAdmin]:
    list_display = ['book_name', 'book_cost']
admin.site.register(publisher, adminPublisher)
admin.site.register(Author, adminAuthor)
admin.site.register(Student, adminStudent)
admin.site.register(Book, adminBook)
```

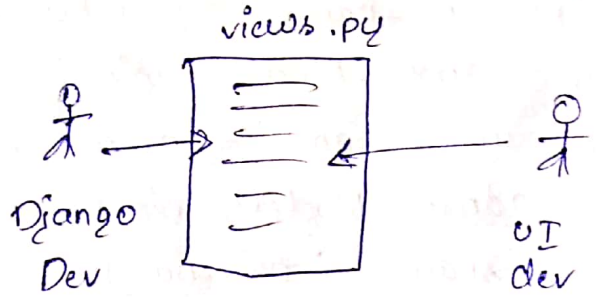
Step 7 : Run the makemigrations, migrate and createsuperuser, runserver commands.

Scanned by CamScanner

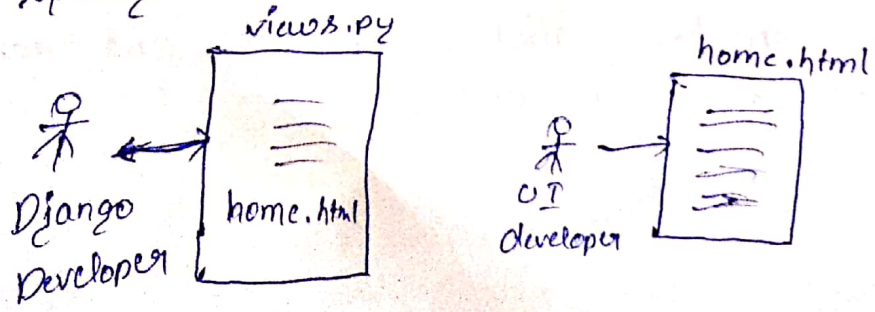
Templates :

1. Even though django follows MVT architecture, T (template) is not a default file because views.py file allows business coding and also presentation coding by default

2. ~~Eve~~ If we use views.py file for both business operations and presentation logics then there may be a chances to raise the errors because the django developer may or may not know UI coding so there may be a chances that django developer may disturb UI coding and also UI developer may or may not know django coding so there may be a chances that ~~django~~ UI developer may disturb django coding



So we go for templates to avoid the above problems that means the UI developer (front-end developer) will write the entire UI coding in a separate template (HTML file) and then the template name will specify in our views file



Scanned by CamScanner

What is a template :-

* A template is a raw string which contains collection of variables and tags
* A variable is a any text which is surrounded by pair of [{{ }}]
  * {{ name }}
  * {{ location }}
  * {{ salary }}
* A variable says "insert something"
* A tag is a any text which is surrounded by %{{ }}%.

  {% if %}

  {% for %}
* A tag says "do something"
* Generally we can create a template in different ways 1. By using python shell

  2. By using html file

(By using python shell)
* It is static way not useful in realtime

(By using html file)
* It is dynamic way, most useful in real-time

By using python shell

* Here we have to import Template and Context from django.template
* Template is a raw string which contains variables and tags.

Scanned by CamScanner

* Context is a dictionary which contains key-value pairs.
* so we assign the values to the keys in the dictionary and finally we map the keys with template variables
* so the keys names must be same as variables names
* we use render( ) to map the context keys with template variables, like below

Step1 :- Go to python shell

Step2 :- import template and Context like below

```
from django.template import Template, context
t = Template("He is {{name}} and he got job
        in {{company}} and getting {{sal}} salary")
c = Context({'name': 'sai', 'company':'TCS', 'sal':10000})
p = t.render(c)
print(p)
```

He is sai and he got job in TCS and getting 10000 salary

* We can create multiple context for a single template

By creating a templates by using html file

Step1 : Foldername : tebdj4pm

Step2 : Projectname : Templatebasicpro

Scanned by CamScanner

Step3 : App Name : Tempbasicapp

Step4 : Go to views.py file

```
from django.shortcuts import render

class
def home(request):
    return render(request, 'tempbasic.html')
```

Step5 : ~~Create~~ Create html file with name tempbasic.html

* we can create html file in different ways

Way-1

1. write the html file name in views.py file directly
2. Keep the mouse on html file name and click Alt+enter
3. Save the file

Way-2

* Right click on templates folder
* Go to new option, click on html file option
* Give any name for the with .html extension
* Save the file and use the file in the views.

Step6 :- Open tempbasic.html file and write the following code

```
<html>
<head>
<title> Templates </title>
</head>
<style>
    h1 {
        text-align : center;
```

Scanned by CamScanner

```
    color : white ;
    background - color : green;
    margin - left : 330px
    margin - right : 330px
    border - radius : 10px
}
table {
    margin - left : 330px;
    width : 420px
    text - align : center;
    height : 130px;
    color : white ;
    border - radius : 20px;
}
body {
    background - color : darkcyan;
}
tr : nth - of - type (even) {
        background - color : green;
}
tr : nth - of - type (odd) {
        background - color : orchid;
}
th {
    background - color : deepink;
}
</style>
```

Scanned by CamScanner

```
<body>
<h1> welcome to Durgasoft </h1>
<table border="2">
    <thead>
    <tr>
        <th> sno </th>
        <th> sname </th>
        <th> slocation </th>
    </tr>
    </thead>
    <tbody>
    <tr>
        <td> 100 </td>
        <td> vani </td>
        <td> hyd </td>
    </tr>
    <tr>
        <td> 101 </td>
        <td> sai </td>
        <td> banglor </td>
    </tr>
    <tr>
        <td> 102 </td>
        <td> sathya </td>
        <td> Chennai </td>
    </tr>
    <tr>
        <td> 103 </td>
```

Scanned by CamScanner

```
        <td> nani</td>
        <td> tpt </td>
    </tr>
    <tr>
        <td> 104 </td>
        <td> ramesh </td>
        <td> pune </td>
    </tr>
    <tr>
        <td>105 </td>
        <td> sirish</td>
        <td> mumbai </td>
    </tr>
    </tbody>
  </table>
  </body>
  </html>
```

Step7:- urls.py

```
from django.contrib import admin
from tempbasicapp import views
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', views.home)

]
```

Step8 :- run the server

Scanned by CamScanner

welcome to Durgasoft

Sno	Sname	Slocation
100	vani	hyd
101	sai	banglo
102	sathya	chennai
103	nani	tpt
104	ramesh	pune
105	sirish	mumbai

Styles

CSS provides three types of styles

1. inline styles → these styles we apply in the beginning html tag
2. Internal styles → these styles we take in separate <style> tag and in <style> and </style> tags we apply all required styles.
3. External styles → these styles we take in separate CSS file, in that CSS file we apply all required styles and finally we have to that CSS file in the html file.

margins and paddings

* Margin styles are used to maintain the gap between two tags

1. margin-top
2. margin-right
3. margin-bottom
4. margin-left

Scanned by CamScanner

Ex:- margin : 100px 200px 150px 300px ;
↓ top, ↓ right, ↓ bottom, ↓ left

margin : 100px 200px 150px ;
↓ top, ↓ right, left, ↓ bottom

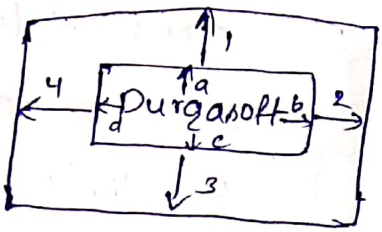
margin : 100px 200px ;
↓ top, bottom, ↓ right, left

margin : 100px ;
↓ top, right, bottom, left

Paddings :

* Padding styles are used to maintain the space between the content and its tag

a. padding - top
b. padding - right
c. padding - bottom
d. padding - left



Ex:- padding : 100px 200px 150px 300px ;
↓ top, ↓ right, ↓ bottom, ↓ left

padding : 100px 200px 150px ;
↓ top, ↓ right, left, ↓ bottom

padding : 100px 200px ;
↓ top, bottom, ↓ right, left

padding : 100px ;
top, right, bottom, left

Scanned by CamScanner

Managing static files

Generally we have to make more styles and layouts to the existing webpage, we can apply styles in different ways., but inline, internal styles and external styles.

* In real time we use external styles that means we take .css file in separate folder and we apply all required styles in .css and finally we give the link between html and css file by using <link> tag.
* here, this .css is called static file
* we can manage different files in our web applications, these files are static files like css files, javascript files, images, text file, pdf files, excel files ---
* we create a folder with name "static" and in this "static" folder we will create different sub folders like "css", "images" and "javascript" folders.
* "css" folder is used to take which css files, "images" folder is used to take images and "javascript" folder is used to take .js files.
* if we have any other files then we can create another folder with name "others" and in this "others" folder we can take any extra files.
* Generally we follow the following changes or steps to manage static files

Scanned by CamScanner

1. settings.py file

STATICFILES_DIRS = [

os.path.join(BASE_DIR, 'static')

]

2. in project folder

static

---- css

---- home.css

---- images

---- veg.jpg

3.) in html file

{% load staticfiles %} # at the top of html file

for css file --> <link rel="stylesheet" href="{% static 'css/home.css' %}">

before title tag

for images --> <img src="{% static 'images/veg1.jpg' %}">

at required place

* About bootstrap

we have to install or we have to use bootstrap lines ~~iou~~ in our html file after title tag.

1. go to www.w3schools.com
2. select "bootstrap"
3. select any one concept like tables concept
4. click on any one option "Try it yourself"

Scanned by CamScanner

5. Copy the bootstrap like which are available after <title> tag

6. paste the same bootstrap line in our html lines after <title> tag

Step1 : Foldername : febdjupm

Step2 : ProjectName : staticfilespro

Step3 : AppName : staticfilesapp

Step4 : go to settings.py file and write the following code at the end of the file

```
STATICFILES_DIRS = [
        os.path.join(BASE_DIR, 'static')

]
```

Step5 : create a folder with name "static", and create "css", "images" subfolders in "static" folder.

Step6 : create home.css file in "css" folder.

Step7 : copy one images from the computer and paste in the images folder.

Step8 : go to views.py file and write the following code.

```
from django.shortcuts import render

def home(request):
    return render(request, 'staticfiles.html')
```

Step9 : create "staticfiles.html" file in the Template folder and write the following code.

Scanned by CamScanner

```
{% load staticfiles %}
<html>
<head>
    <link rel = "stylesheet" href = " {% static 'css/home.
                                        css' %} ">
  <title> welcome to Durgasoft </title>
  # copy and paste bootlines here
</head>
<style>
    h1 {
        font-size : 50px ;
        margin-bottom : 10px;
    }
    .jumbotron {
            height : 80px;
            margin : 10px 70px 10px 70px;

    }
    img {
            margin : 10px 150px 10px 70px;
            width : 87% ;
    }
</style>
<body>
  <div class = "jumbotron bg-success">
      <h1 class = "text-white"> welcome to Django </h1>
  </div>
```

Scanned by CamScanner

```
<img src = "{% static 'images/veg1.jpg' %}">

</body>
</html>
```

Step10: Go to home.css file

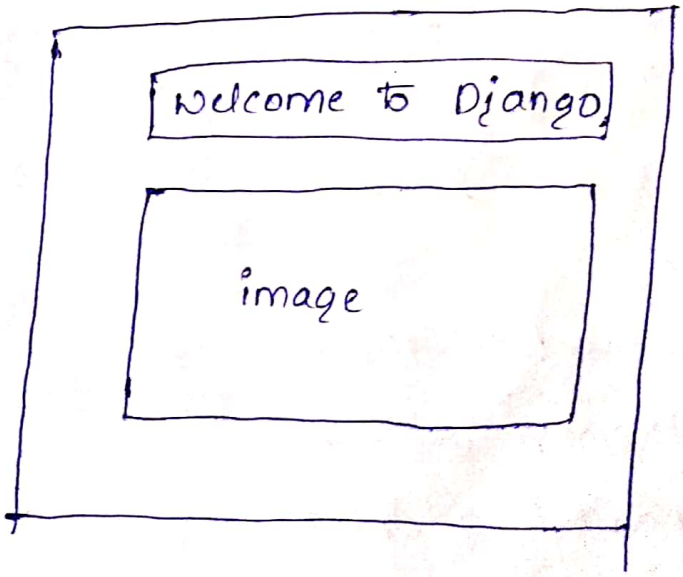
```
.jumbotron {
    margin-top : 30px ;
    text-align : center;
}
```

Step11 : Go to urls.py file

```
from django.conf.urls import include, url
from django.contrib import admin
from staticfilesapp import views
urlpatterns = [
    url [r'^admin/', include (admin.site.urls)],
    url (r'^$', views.home)
```

Step12: ~~Run~~ runserver command

Output:

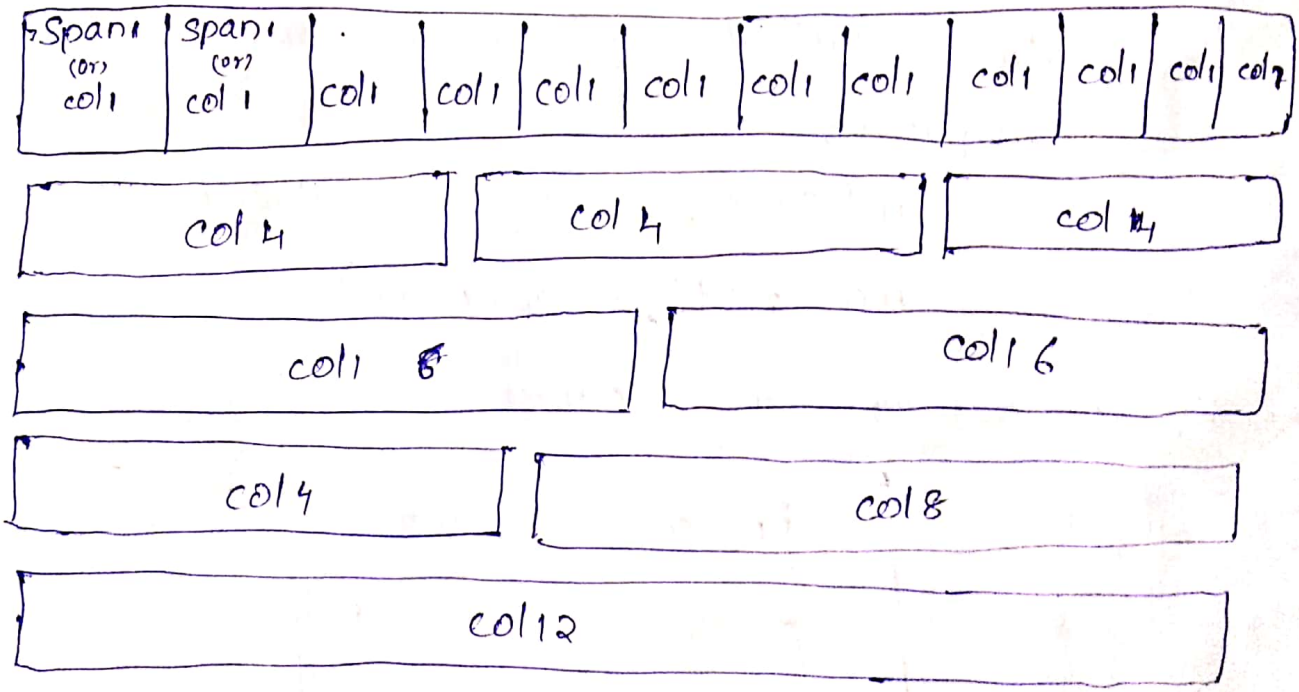


Scanned by CamScanner

Gridsystem

Gridsystem is responsive and the columns will rearrange automatically based on the screen size

* Generally bootstrap provides lot of default classes to create responsive websites
* Some of the classes which are used in gridsystem are .container, .row, .col-md-, .col-sm-, .col-lg-, .col-xl-
* A container can have one or n number of rows and each rows contains 12 columns.
* If we use gridsystem in our application, then we can use any corner of the web application



Step1 : foldername : febdjupm

Step2 : ProjectName : gridsystem

Step3 : App Name : gridsystemapp

Step4 : Go to views.py file

Scanned by CamScanner

```
from django.shortcuts import render

def home(request):
    return render(request, 'gridsystem.html')
```

Step5: Create gridsystem.html file in templates folder

```
<html>
<head>
<title> Grid system </title>
# copy and paste bootstrap lines
</head>
<style>
    .row {
        border : 2px blue solid;
    }
    # cd1 {
        border : 2px red solid;
    }
    # cd2 {
        border : 2px green solid;
    }
    # cd3 {
        border : 2px deeppink solid;
    }
    # row2 {
        background-color : orchid;
    }
    # cd4 {
        background-color : gray;
    }
```

Scanned by CamScanner

```
# row3 {
    background-color : cyan;
  }
</style>
<body>
<center><h1> Grid system </h1> </center>
<div class = "container">
    <div class = "row" id = "row1" >
        <div class = "col-md-4" id = "cd1" >
            <h1 class = "bg-success text-center text-white" >
                                    Python </h1>
                1. Simple and Easy <br>
                2. Powerful language <br>
                3. Open and Free source <br>
                4. Extendable and Portable <br>
                5. Brings tons of Packages <br>
        </div>
        <div class = "col-md-4" id = "cd2" >
            <h1 class = "bg-success text-center text-white" >
                                    Django </h1>
                1. Its Python Framework <br>
                2. Creates Dynamic website <br>
                3. Follows MVT Architecture <br>
                4. Brings default Database <br>
                5. versatile and Scalable <br>
        </div>
        <div class = "col-md-4" id = "cd3" >
            <h1 class = "bg-success text-center text-white" >
```

Scanned by CamScanner

```
UI Technologies <h1>
1. creates Frontend of website <br>
2. Applies styles and layouts <br>
3. HTML, CSS, Bootstrap, javascript .. <br>
4. HTML creates decides structure of page <br>
5. Javascripts add dynamic interactivity <br>
</div>
</div>
<div class = "row" id = "row2">
<div class = "offset-md-2 col-md-8" id = "cd-4">
<p> python is simple language and Django
is framework of python language ; python is
simple language and Django is framework
of python language ...</p>
</div>
</div>
<div class = "row" id = "row3">
<div class = "offset-md-3 col-md-6">
<h1> Please enter your details </h1>
<center>
<form method = "POST">
{% csrf_token %}
<label>Name :     </label>
<input type = "text" name = "name" <br>
```

Scanned by CamScanner

```
<label> Number : </label>
<input type = " number " name = "mob" <br>
<label> Email :     </label>
<input type = " email " name = "email" <br>
<center >
<input type = "submit" value = "submit" class
                = "btn btn-success" >
<input type = "reset" value = "clear" class
                = "btn btn-danger" >
</center>
</form>
</center>
</div>
</div>
</body>
</html>
```

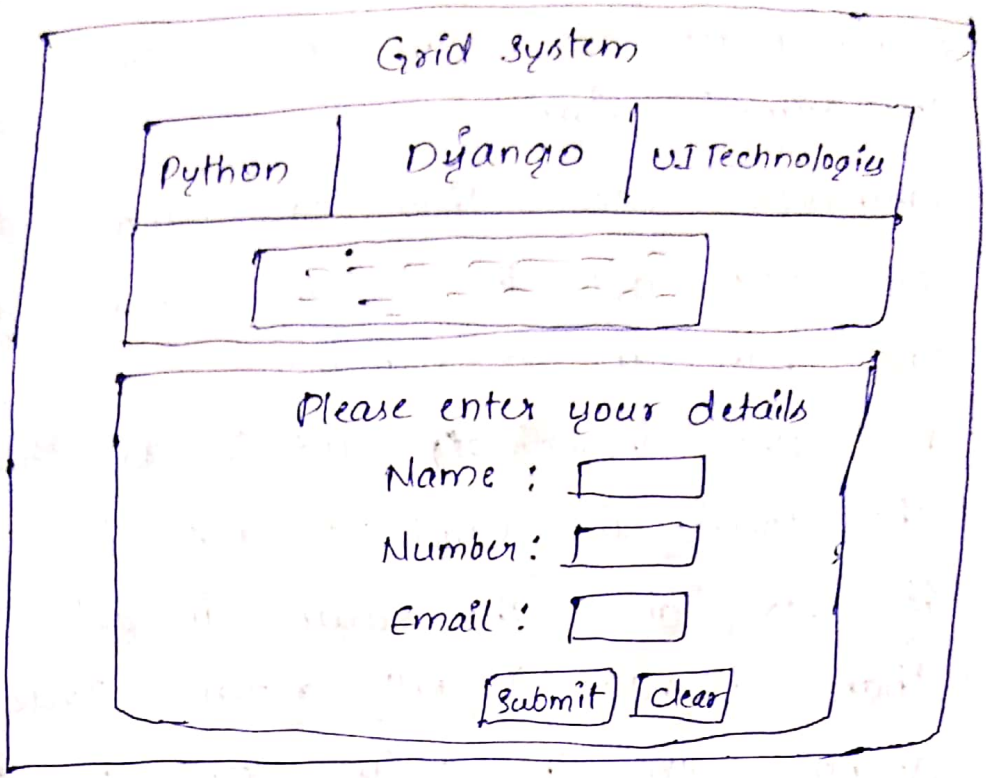
Step6 : Go to urls.py file

```
from django.conf.urls import include, url
from gridsystemapp import views
urlpatterns = (
    url( r'^$', views.home)
)
```

Scanned by CamScanner

Step7 : Runserver command

O/P :



Forms

* A form is a collection of API's and built-in classes
* A form provides the interaction between user and admin of website.
* A form will take user data and saves the data in the database.
* creating a form is like creating a model.
* When we run the models then ORM code of models file will converted into sql code and sql code will execute in the database, so database table will create.
* When we run the forms then ORM code of forms file will convert into html code and

Scanned by CamScanner

html code will reflect on the browser by creating a form.

* forms.Form is the base class for every userdefined form.

* Whenever user clicks on submit button then we take all the values along with form and save in a variable.

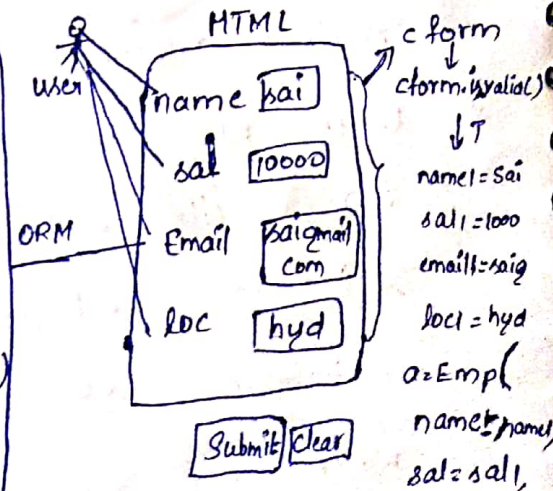
* We use ~~is_validate~~ valid() to check whether the form is valid or not

* if user gives the values in all the fields than is_valid() will return True or if user didnot give the value for any specific field then is_valid() will return false.

* If the variable is true then we take all fields values into separate variables and then we map the variables with table columns and then we save the data by using save().

Form

```
class EmpForm(f.F):
    name  = f.CF(ml=20)
    sal   = f.IF()
    email = f.EF(max_length=20)
    loc   = f.CF(ml=20)
```



~~ORM~~

model

```
class Emp(m.M):
name = m.CF(ml=20)
sal = m.IF()
email = m.EF(ml=20)
loc = m.CF(ml=20)
```

ORM

sql

name	sal	emai	loc
sai	1000	saigmail.com	hyd

admin

Note: models.py file is the default file where we write our models but forms.py file is not default to write our forms so we have to create a python file with name forms.py file.

* Now we will create a project to display a form to take user data to save in the database
* we will create this project in three parts

1. converting forms ORM into html code
2. Displaying the form on the browser.
3. Taking the user data in the database.

Step1: foldername : febdjupm

Step2: ProjectName : contactformpro

Step3: AppName : contactformapp

Step4: Create forms.py in the application
→ Right click on contactformapp → New
→ Pythonfile → forms.py

Step5: Create a contact form like below (in forms.py)

Scanned by CamScanner

```
from django import forms
class contactForm(forms.Form):
    name = forms.CharField(max_length=50)
    salary = forms.IntegerField()
    email = forms.EmailField(max_length=50)
    location = forms.CharField(max_length=50)
```

Converting forms ORM code into html

Step 6: Go to python shell and run the following commands

```
>>> from contactformapp.forms import ContactForm
>>> c = ContactForm()
>>> print(c)
```

[from EFormApp.forms import EForm.]

```
<tr>
<th> label for ="id_name"> Name: </label></th>

<td> input id = "id_name" maxlength = "50" name =
"name" type = "text"/> </td>

</tr>
<tr>
<th> label for = "id_salary" >Salary: </label></th>

<td> input id = "id_salary" name = "salary" type
= "number"/> </td>

</tr>
```

Scanned by CamScanner

```
</tr>

<tr>
  <th><label for="id_email">Email:</label></th>
  <td><input id="id_email" maxlength="50" name=
                "email" type="email"/></td>

</tr>

<tr>
  <th>label for="id_location">location:</label>
                                  </th>

  <td><input id="id_location" maxlength="50" name
                "location" type="text"/></td>

</tr>
```

Displaying the form on the browser

Step 7 :. Go to views.py file and write the following code

```
from django.http.response import HttpResponse
from django.shortcuts import render
from .forms import ContactForm
from .models import ContactData
def contact_view(request):
    if request.method == "POST":
        cform = ContactForm(request.POST)
        if cform.is_valid():
```

Scanned by CamScanner

```
name1 = request.POST.get('name', '')
salary1 = request.POST.get('salary', '')
email1 = request.POST.get('email', '')
location1 = request.POST.get('location', '')
data = ContactData(
        name = name1,
        salary = salary1,
        email = email1,
        location = location1,
    )

    data.save()
    cform = ContactForm()
    return render(request, 'contact.html', {'abc':
                                            cform})

else:
    return HttpResponse("Invalid form")

else:
    cform = ContactForm()
    return render(request, 'contact.html', {'abc':
                                            cform})
```

Step 8 : Create contact.html file in templates folder

```
<html>
<head>
<title> Contact Form </title>
```

Scanned by CamScanner

copy and paste bootstrap lines

```
</head>
<body>
<div class = "container" >
    <div class = "row">
        <div class = "offset-md-4 col-md-4">
            <form method = "POST" >
                {% csrf_token %}
                {{ abc.as-p}} <br>
                <input type = "submit" value = "submit">
                <input type = "reset" value = "clear">
            </form>
        </div>
    </div>
</div>
</body>
</html>
```

[cross site request forgery] (csrf)

Step9 : Go to urls.py file and write the following urls

```
from django.cont.urls import include, url
from django.contrib import admin
from contactformapp import views
urlpatterns = [
```

Scanned by CamScanner

```
url(r'^admin/', include(admin.site.urls)),
url(r'^$', views.contact_view)

]
```

Step 10 : Run the server by using runserver command, click on IPaddress, a browser will open like below

Name : [ ]

Salary : [ ]

Email : [ ]

Location : [ ]

[Submit] [Clear]

Here, if we give data in all the fields then that data should go to database table but upto now there is no table behind the form to store the user data

Taking userdata into database :

Step 11 : Create a database in mysql and with name contactupmdb and configure in settings.py file like below

```
DATABASES : {
```

Scanned by CamScanner

```
'default' : {
    'ENGINE' : 'django.db.backends.mysql',
    'NAME' : 'contactmpmdb',
    'USER' : 'root',
    'PASSWORD' : 'root',
  }
}
```

Step12 :- Go to models file and create a model like below

```
from django.db import models

class contactData (models.Model):
    name = models.CharField (max_length = 50)
    salary = models.IntegerField ( )
    email = models.EmailField (max_length = 50)
    location = models.Charfield (max_length = 50)
```

Step13 : run makemigrations and migrate commands.

Step14 : run the server by using runserver command and click on the IPaddress and a browser opens like below

Name : [Sai]

Salary : [10000]

Email : [sai@gmail.com]

Scanned by CamScanner

location : [hyd]

[submit] [clear]

* After giving data in all the fields, if we click on submit button then the data will save in the database and it opens a new form

* Go to database and check the data

mysql > select * from contactformapp - contactdata;

id	name	salary	email	location
1	sai	10000	sai@gmail.com	hyd
2	Kesar	20000	Kesar@gmail.com	banglr

* Every form has two request methods like GET and POST

* GET request is the first request to the form to get and display an empty form

* POST request is the second request to the form to take user data and post in the database table

* GET request has only one response from the server that is displaying empty form on the form

* POST request has two responses from the server those are

Scanned by CamScanner

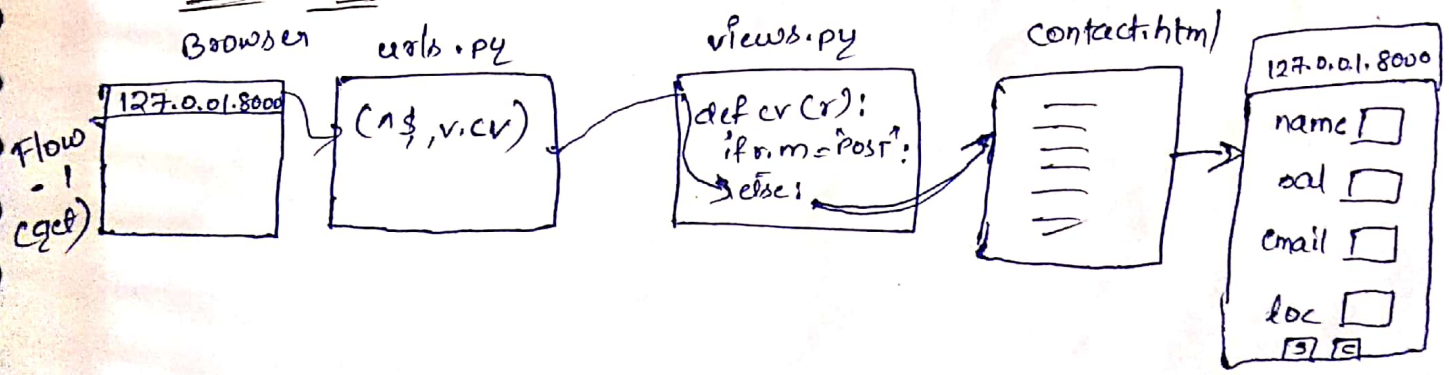
1. Posting data in the database
2. Displaying an empty form on the browser after posting (saving data in the database)

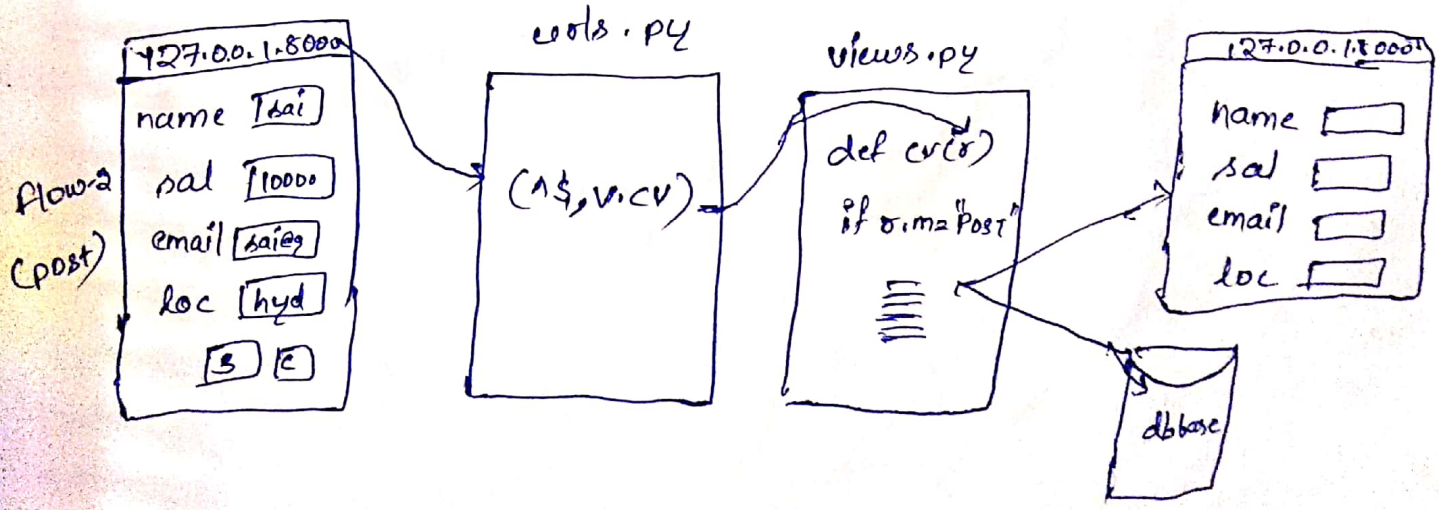
Assign :

1. Create a project to display an enquiry form
2. The form should take firstname, lastname, emailid, mobilenumber, courses, fee, location values from the user
3. user data must save in the database

model - contactdata (EnquiryData)
form - contactform (EnquiryForm)
view - contact_view (Enquiry_view)

Project flow





Scanned by CamScanner

Enquiryformapp

Step1 : foldername : febdj4pm

Step2 : ProjectName : enquiryformpro

Step3 : AppName : enquiryformapp

Step4 : create forms.py in the application
→ Right click on enquiryformapp → New
→ pythonfile → forms.py

Step5 : create a enquiryform like below in forms.py)

```
from django import forms.
class enquiryform(forms.Form):
    first_name = forms.CharField(max_length=50)
    last_name = forms.CharField(max_length=50)
    email_id = forms.EmailField(max_length=50)
    mobile_no = forms.BigIntegerField( )
    courses = forms.CharField(max_length=50)
    fee = forms.IntegerField( )
    location = forms.CharField(max_length=50)
```

Converting forms ORM code into html

Step6 : Go to python shell and run the following commands.

```
>>> from enquiryformapp.forms import EnquiryForm
>>> e = EnquiryForm( )
>>> print(e)
<tr>
<th> label for = "id_first_name"> first_name : </label>
</th>
```

Scanned by CamScanner

```
<td> input id = "id_first_name" maxlength = "50" name =
                "first name"
                "first_name" type = "text"/></td>

</tr>
<tr>
<th> label for = "id_last_name" > last_name :</label>
                                                  </th>
                                maxlength = "50" name =
<td > input id = "id_last_name" last_name = "last_name"
                "last name" type = "text"/ > </td>

<tr >
  <th> <label for = "id_email_id " > Email_id :</label></th>
  <td> input id = "id_email_id" maxlength = "50" name =
                "email" type = "email_id" /></td>

</tr>
<tr>
  <th> label for = "id_mobile_no" > Mobile_no : </label></th>
  <td > input id = "id_mobile_no" name = "mobile_no" type
                = "number"/> </td>

</tr>
<tr>
  <th> label for = "id_courses"> courses : </label></th>
  <td> input id = "id_courses" maxlength = "50" name =
                "courses" type = "text"/></td>

</tr>

<tr>
  <th> label for = "id_fee"> fee : </label></th>
  <td> input id = "id_fee" name = "fee" type
                = "number "/> </td>
```

Scanned by CamScanner

```
</tr>
<tr>
    <th><label for="id_location">location :</label>
                                        </th>
    <td><input id="id_location" maxlength="50" name
                    = "location" type = "text"/></td>
</tr>
```

Displaying the form on the browser

Step 7 : Go to views.py file and write the following code

```
from django.http.response import HttpResponse
from django.shortcuts import render
from forms import EnquiryForm
from .models import EnquiryData
def Enquiry_view(request):
    if request.method == "POST":
        eform = EnquiryForm(request.POST)
        if eform.is_valid():
            first_name1=request.POST.get('first_name','')
            last_name1=request_POST.get('last_name','')
            email_id1=request_POST_get('email_id','')
            mobile_no1=request_POST_get('mobile_no','')
            courses1 = request_POST_get('courses','')
```

Scanned by CamScanner

```
fee1 = request.POST.get('fee', '')
location1 = request.POST.get('location', '')
data1 = EnquiryData(
        first_name = first_name1,
        last_name = last_name1,
        email_id1 = email_id1,
        mobile_no = mobile_no1,
        courses = courses1,
        fee = fee1,
        location = location1,
    )
    data.save()
    eform = EnquiryForm()
    return render(request, 'Enquiry.html', {'abc': eform})
else:
    return HttpResponse("Invalid form")
else:
    eform = EnquiryForm()
    return render(request, 'Enquiry.html', {'abc': eform})
```

Step8 : Create Enquiry.html file in templates folder

```
<html>
<head>
<title>Enquiry Form</title>
```

Scanned by CamScanner

```
# copy and paste bootstrap lines
</head>
< body>
<div class = " container " >
    <div class = "row">
    <div class = " offset-md-4 col-md -4" >
        <form method = " Post " >
            { % csrf_token % " }
            { { abc.as_p } } <br>
            <input type = " submit " value = "submit">
            <input type = " reset " value = "clear">
        </form>
    </div>
    </div>
</div>
</body>
</html>
```

Step9 :- Go to urls.py file and write the following urls

```
from django.conf.urls import include, url
from django.contrib import admin
from Enquiryformapp import views
urlpatterns = [
    url (r'^admin/', include (admin.site.urls)),
    url (r'^$', views.enquiry_view)
]
```

Scanned by CamScanner

Step10 : Run the server by using runserver command, click on IPaddress, a browser will open like below

first_name : [ ]

last_name : [ ]

email_id : [ ]

mobile_no : [ ]

courses : [ ]

fee : [ ]

location : [ ]

[submit] [clear]

Here, if we give data in all the fields then that data should go to database table but upto now there is no table behind the form to store the user data.

Taking usrdata into database

Step11 : create a database in mysql and with name enquiry4pmdb and configure in settings.py file like below

```
DATABASES : {
    'default' : {
        'ENGINE' : 'django.db.backends.mysql',
        'NAME' : 'enquiry4pmdb',
        'USER' : 'root',
```

Scanned by CamScanner

'PASSWORD' : 'root',

}

}

Step12 : Go to model file and create a model like below

```
from django.db import models
class EnquiryData(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email_id = models.EmailField(max_length=50)
    mobile_no = models.BigIntegerField( )
    courses = models.CharField(max_length=50)
    fee = models.IntegerField( )
    location = models.CharField(max_length=50)
```

Step13 : run makemigrations and migrate commands

Step14 : run the server by using runserver command and click on the IPaddress and a browser opens like below

first_name : [vanitha]

last_name : [vani]

email_id : [vani@gmail.com]

mobile_no : [960252382]

courses : [Django]

fee : [2000]

location : [hyd]

[Submit] [clear]

Scanned by CamScanner

* After giving data in all the fields, if we click on submit button then the data will save in the database and it opens a new form

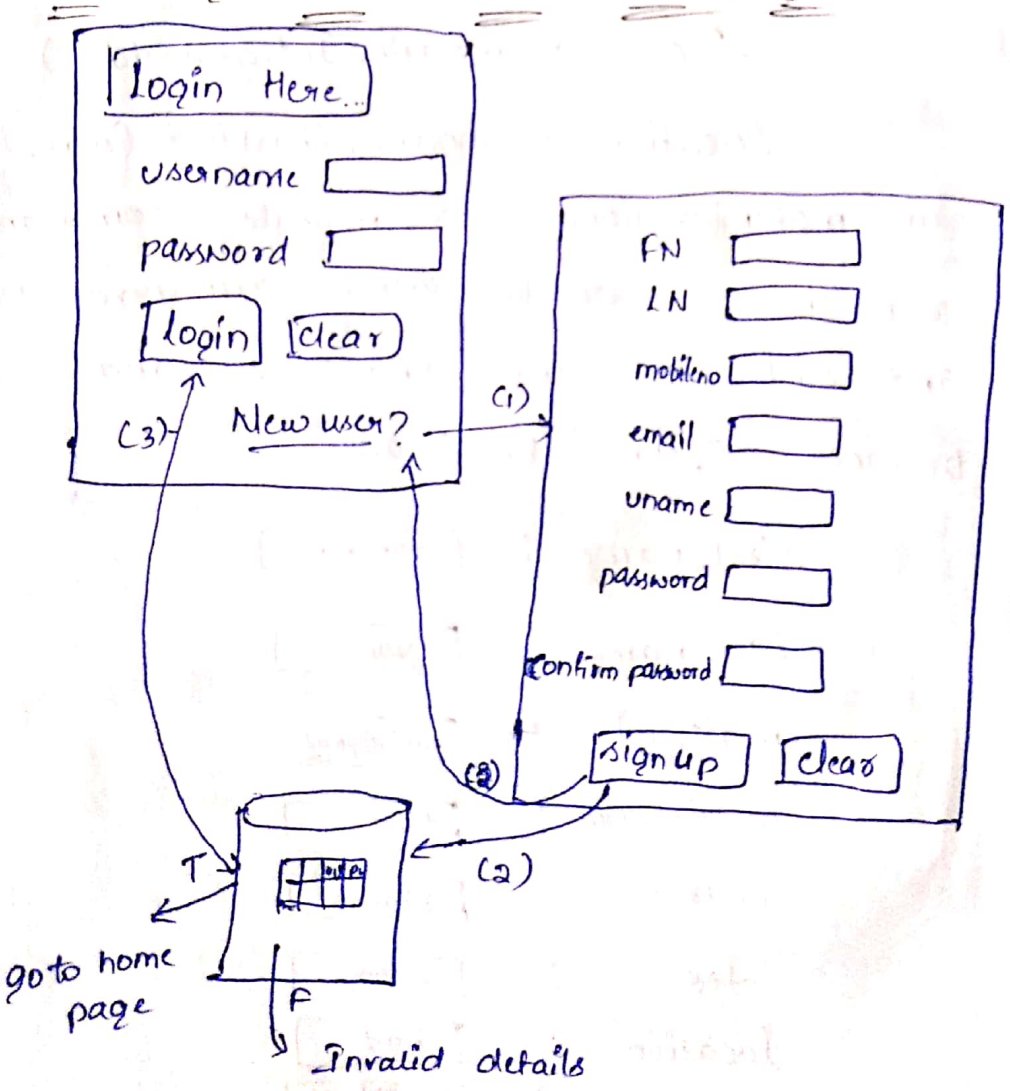
* Go to database and check the data

mysql > select * from enquiryformapp_enyinydata;

id	first_name	last_name	email_id	mobile_no	courses	fee	location
1	vanitha	vani	vani@gmail.com	902825382	django	9000	hyd
2	ammu	amutha	ammu@gmail.com	963823821	python	2000	tpt

Project Flow

Creating registration and login project



Scanned by CamScanner

Step1 : Foldername : febdj4pm

Step2 : ProjectName : registrationlogandloginpro

Step3 : AppName : registrationandloginapp

Step4 : Database Name : regandlogin4pmdb

Step5 : Go to settings.py file and configure the database static folder

```
Databases = {
    'default' : {
        ENGINE : 'django.db.backends.mysql',
        NAME : 'regandlogin4pmdb',
        USER : 'root',
        PASSWORD : 'root',
    }
}

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]
```

Step6 : Go to models files

Scanned by CamScanner

```
from  django.db import  models

class  RegistrationData (models.Model):

    first_name = models.CharField(max_length = 50)
    last_name = models.CharField(max_length = 50)
    mobile  =  models.BigIntegerField( )
    email  =  models.EmailField(max_length=50)
    username =  models.CharField(max_length=50)
    password1 =  models.CharField (max_length=100)
    password2 =  models.CharField (max_length=100)
```

Step7 :- Create forms.py file and create registration form and login form

```
from  django  import  forms

class  RegistrationForm( forms.Form):
    first_name  =  forms.CharField (
        label = "Enter your first name",
        widget = forms.TextInput (
            attrs = {
                'class' : 'form-control',
                'placeholder' : 'Your First Name'
            }
        )
    )
```

Scanned by CamScanner

```
last_name = forms.CharField(
        label = "Enter your Last Name",
        widget = forms.TextInput(
            attrs = {
                'class': 'form-control',
                'placeholder': 'Your Last name'
            }
        )
    )

mobile = forms.IntegerField(
        label = "Enter your mobile Number",
        widget = forms.NumberInput(
            attrs = {
                'class': 'form-control',
                'placeholder': 'Your mobile number'
            }
        )
    )

email = forms.EmailField(
        label = "Enter your email id",
        widget = forms.EmailInput(
            attrs = {
                'class': 'form-control',
```

Scanned by CamScanner

```
        'placeholder' : 'Enter your Email ID'
        }
    )
)

username = forms.CharField(
    label = "Enter username",
    widget = forms.TextInput(
        attrs = {
            'class' : 'form-control',
        'placeholder' : 'Enter Username'
        }
    )
)

password1 = forms.CharField(
    label = "Enter your password",
    widget = forms.PasswordInput(
        attrs = {
            'class' : 'form-control',
        'placeholder' : 'Enter your password'
        }
    )
)
```

Scanned by CamScanner

```
password2 = forms.CharField(
    label = "Enter Confirmation Password",
    widget = forms.PasswordInput(
        attrs = {
            'class' : 'form-control',
            'placeholder' : 'Enter Confirmation password'
        }
    )
)


class LoginForm(forms.Form):
    username = forms.CharField(
        label = "Enter Username",
        widget = forms.TextInput(
            attrs = {
                'class' : 'form-control',
                'placeholder' : 'Your Username'
            }
        )
)
```

Scanned by CamScanner

```
password1 = forms.CharField(
    label = "Enter Your Password",
    widget = forms.PasswordInput(
        attrs = {
            'class' : 'form-control',
            'placeholder' : 'Your Password'
        }
    )
)
```

Step 8 : Go to views.py file

```
from django.shortcuts import render
from .forms import RegistrationForm, LoginForm
from .models import RegistrationData
from django.http.respons import HttpResponse

def registration_view(request):
    if request.method == "POST":
        form = RegistrationForm(request, POST)
        if form.is_valid():
            first_name = request.POST.get('firstname', '')
            last_name = request.POST.get('lastname', '')
            mobile = request.POST.get('mobile', '')
```

Scanned by CamScanner

```
email = request.POST.get('email', '')
username = request.POST.get('username', '')
password1 = request.POST.get('password1', '')
password2 = request.POST.get('password2', '')
data = RegistrationData(
        first_name = first_name,
        last_name = last_name,
        mobile = mobile,
        email = email,
        username = username,
        password1 = password1,
        password2 = password2,
    )
    data.save()
    rform = RegistrationForm()
    lform = LoginForm()
    return render(request, 'login.html', {'lform':
                                          lform})
else:
    rform = RegistrationForm()
    return render(request, 'registration.html', {'rform':
                                                 rform})

def login_view(request):
    if request.method == 'POST':
        lform = LoginForm(request.POST)
```

Scanned by CamScanner

```
if lform.is_valid():
        uname = request.POST.get('username','')
        pwd = request.POST.get('pwd','')

        uname1 = RegistrationData.objects.filter
                                (pas username=uname)
        pwd1 = RegistrationData.objects.filter
                                (password1 = pwd)
        if uname1 and pwd1:
            return HttpResponse("valid Details")
        else:
            return HttpResponse("Invalid Details")

else:
    lform = LoginForm()
    return render(request, 'login.html', {'lform':lform})
```

Step9: Create registration.html file

```
{% load staticfiles %}
<html>
<head>
    <title>Registration Form</title>
    <link rel="stylesheet" href="{% static 'css/reg.css' %}">

# copy and paste bootstrap lines

</head>
<body>
```

Scanned by CamScanner

```
<div class = "container">
    <div class = "row">
        <div class = "offset-md-4 col-md-4 ">
            <form method = "POST">
                <h4 class = " text-center bg-dark
                    text-white "> Please Register Your
                        Details.. </h4>
                {% csrf_token %}
                {{ rform }} <br>
                <center>
                <input type = "submit" value = "Signup"
                        class = "btn btn-success ">
                <input type = "reset" value = "clear"
                        class = "btn btn-danger">
                </center>
            </form>
        </div>
    </div>
</div>
</body>
</html>
```

Scanned by CamScanner

Step10: Create login.html in template folder

```
{% load staticfiles %}
<html>
<head>
<title> LoginForm </title>
<link rel = "stylesheet" href = "{% static 'css/login.css' %}">
# Copy and paste bootstrap lines
</head>
<body>
<div class = "container">
    <div class = "row">
        <div class = "offset-md-4 col-md-4">
            <form method = "POST">
                <h2 class = "text-center bg-dark
                        text-white"> Login here ..</h2>
                {% csrf_token %}
                {{ form }} <br>
                <center>
                <input type = "submit" value = "Login"
                        class = "btn btn-success">
                <input type = "reset" value = "clear"
                        class = "btn btn-danger">&lt;br&gt;
                <a href = "/registration/" > New User? </a>
```

Scanned by CamScanner

```
</center>
</form>
</div>
</div>
</div>
</body>
</html>
```

Step 11 :- Create static folder in the project and create css folder in static folder create reg.css file and login.css file in css folder.

Open reg.css file

```
form {
    width : 400px;
    margin-top : 30px;
}
```

Open login.css file

```
form {
    margin-top : 50px;
    width : 500 px;
    border-radius : 20px;
}
```

Scanned by CamScanner

Step12 : Go to urls.py file and write the following urls.

```
from django.conf.urls import include, url
from django.contrib import admin
from registrationandloginapp import views
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^registration/', views.registration_view),
    url(r'^$', views.login_view),
]
```

Step13 : Run makemigrations, migrate and runserver commands. Click on the IPaddress a browser will open like below

Login Here....

Enter Username:

Your username

Enter Your Password:

Your Password

Login clear

New User?

* If we click on New User link then it will open registration Form

Scanned by CamScanner

127.0.0.1:8000/registration/

Please Register Your Details.

Enter your First Name :

Your First Name

Enter Your Last Name :

Your last Name

Enter Your Mobile Number :

Your Mobile Number

Enter Your Email ID :

Your Email ID

Enter your User Name :

Your username

Enter Your Password :

Your password

Enter Confirmation Password :

confirmation Password

SignUp | Clear

* After giving the data in all the fields if we click on signup button then the data will save in the database it will open login form

* The login form we give proper username and password to go to homepage

Scanned by CamScanner

Creating a feedback form

Step1 : FolderName : febdj4pm

Step2 : ProjectName : feedbackpro

Step3 : App Name : feedbackapp

Step4 : Database Name : feedback4pmdb

Step5 : Configure the database in the settings.py file

```
DATABASES = {
    'default' : {
    'ENGINE' : django.db.backends.mysql',
    'NAME' : 'feedback4pmdb',
    'USER' : 'root',
    'PASSWORD' : 'root',
    }
}
```

Step6 : Go to model file and create the model

```
from django.db import models

class FeedbackData(models.Model):
    name = models.CharField(max_length=100)
    rating = models.IntegerField()
    date = models.DateField()
    location = models.CharField(max_length=100)
    feedback = models.TextField(max_length=100)
```

Step 7 : Go to forms.py file

```
from django import forms

class FeedbackForm(forms.Form):
```

Scanned by CamScanner

```
name = forms.CharField(
    label = "Enter Your Name",
    widget = forms.TextInput (
        attrs = {
            'class': 'form-control',
            'placholder': 'Your Name'
        }
    )
)

rating = forms.IntegerField (
    label = "Enter your Rating",
    widget = forms.NumberInput (
        attrs = {
            'class': 'form-control',
            'placeholder': 'Your Rating'
        }
    )
)

location = forms.CharField (
    label = "Enter your location",
    widget = forms.TextInput (
        attrs = {
            'class': 'form-control',
            'placeholder': 'Your Location'
```

Scanned by CamScanner

```
        }
    )
)
                                        your valuable feedback")
feedback : forms.CharField(help_text = "Thanks for
    label = "Enter your Feedback",
    widget = forms.TextArea(
        attrs = {
            'class' : 'form-control',
            'placeholder : ' Your Feedback'
        }
    )
)
```

Steps : Go to views.py file

```
from django.shortcuts import render
from .models import FeedbackData
from .forms import FeedbackForm

import datetime as dt
date1 = dt.datetime.now()

def feedback_view(request):
    if request.method == "POST":
        fform = FeedbackForm(request.POST)
        if fform.is_valid():
```

Scanned by CamScanner

```
name = request.POST.get('name', '')
rating = request.POST.get('rating', '')
location = request.POST.get('location', '')
feedback = request.POST.get('feedback', '')

data = FeedbackData(
        name = name.capitalize(),
        rating = rating,
        date = date1,
        location = location,
        feedback = feedback,
    )
data.save()
fform = FeedbackForm()
feedbacks = FeedbackData.objects.all()
return render(request, 'feedback.html', {'fform':
                fform, 'feedbacks': feedbacks})


else:
    feedbacks = FeedbackData.objects.all()
    fform = FeedbackForm()
    return render(request, 'feedback.html', {'fform': fform,
                'feedbacks': feedbacks})
```

Step 9 : Create feedback.html file

Scanned by CamScanner

```
<html>
<head>
<title>Feedback Form </title>
# copy and paste boostrap lines
</head>
<style>
    b{
        color:red;
        font-family : cursive ;
        font-size : 30px;
    }
</style>
<body>
<div class = " container " >
    <div class = " row " >
        <div class = " offset-md-2 col-md-8 " >
        <h2> Your Feedback are ... </h2>
            {% if feedbacks %}
                {% for i in feedbacks %}
                    {% if i.rating >= 4 and i.rating <= 5 %}
                        <b>Name : {{i.name}} <br></b>
                        Rating : {{i.rating}} <br>
                        Location : {{ i. location }} <br>
                        Date : {{ i.date }} <br>
```

Scanned by CamScanner

```
            Feedback : {{ i.feedback }} <br><br>
        {% endif %}
      {% endfor %}
    {% else %}
        <h3>Feedbacks are not Available</h3>
    {% endif %}
    <form method="POST">
        <h2>Please give your Feedback.</h2>
        {% csrf_token %}
        {{ fform }} <br><br>
        <input type="submit" value="Send">
        <input type="reset" value="clear">
    </form>
  </div>
 </div>
</div>
</body>
</html>
```

Step10: Go to urls.py file

```
from django.conf.urls import include, url
from feedbackformapp import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
```

Scanned by CamScanner

```
    url(r'^$', views.feedback_view)
]
```

Step 11 :- Run makemigrations, migrate and runserver commands

Your Feedback are ...

Feedbacks are not Available

Please give your Feedback...

Enter your Name :

Kesav

Enter your Rating :

4

Enter your Location :

Hyderabad

Enter your Feedback :

Hello sir, good morning..

Thanks for your valuable Feedback

Send Clear

* If we click on send button then the feedback will go and send in the database and coming back in the browser and

Scanned by CamScanner

displays on the top of the form

Your Feedbacks are ...

Name : Kesar

Location: Hyderabad

Date : March 21, 2019

Feedback : Hello Sir, Good Morning

Please give your feedback ..

Enter Your Name :

Enter your Rating :

Enter your Location :

Enter your Feedback :

Thanks for your valuable feedback

Send Clear

* If anyone gives feedbacks with rating less than 4 and above 5 then that feedback will go and save in the database like all feedbacks but they will not display on

Scanned by CamScanner

the browser that means we can give the feedbacks with rating 4 and 5 only on the browser

Assignment:- 1. Create a project on Employee

2. The main page should have two links (Employee Entry and Employee Details

3. If we click on Employee Entry then a form should open with ~~First~~ Name Employee Number, FirstName, LastName, Salary, Location, Company, Position, Mobile Number, Emailid fields

4. if we click on Employee Details then all details of all employees should display on the browser in the table format

Create student registration form

ORM operations

Faker module

* Faker module is used to generate some fake or dummy automatically at runtime.
* Python provides this faker module.
* In realtime, if we want to perform testing operations then initially we perform testing operations, on some dummy data if all test cases are working properly then

Scanned by CamScanner

only we can apply on main data

* If we apply directly on main data, then there may be chances to damage our main data if the test cases are failed.

Step1 : foldername : feldjrpm

Step2 : Projectname : fakerproject

Step3 : Appname : fakerapp

Step4 : Database Name : faker12pmdb

Step5 : configure database in settings.py file

Step6 : Go to models.py and create emp model

```
from django.db import models

class EmpFakeData(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(max_length=50)
    location = models.CharField(max_length=100)
    salary = models.IntegerField()
    job = models.CharField(max_length=100)
```

Step7 :- Go to views.py file

```
from django.shortcuts import render, redirect
from .models import EmpFakeData
from faker import Faker
fake = Faker()
```

Scanned by CamScanner

```
def emp_inserting_view(request):
    for i in range(1000):
        first_name = fake.first_name()
        last_name = fake.last_name()
        email = fake.email()
        salary = fake.random_element(elements =
                    (1000, 12000, 14000, 15000, 21000))
        location = fake.random_element(elements =
                    ('Hyderabad', 'Banglore', 'Chennai', 'Mumbai'))
        job = fake.random_element(elements =
                    ('TL', 'PM', 'HR', 'GPM'))
        data = EmpFakeDate(
            first_name = first_name,
            last_name = last_name,
            email = email,
            salary = salary,
            location = location,
            job = job,
        )
        data.save()
return return redirect('/fetch/')
def emp_fetching_view(request):
    edata = EmpFakeData.objects.all()
    return render(request, 'emp_fake_data.html', {'edata':
                edata})
```

Scanned by CamScanner

Steps :- Create emp_fake_data.html file

```
<html>
<head>
<title>Emp Fake Data</title>
</head>
<style>
    th, td {
        height : 40px;
        width : 100px;
        font-size : 20px;
        text-align : center;
    }
</style>
<body>
{% if data %}
    <center>
    <h3> EmployeeData</h3>
    <table border="2px">
    <thead>
    <tr>
      <th>FirstName</th>
      <th>LastName</th>
      <th>Email</th>
      <th>Salary</th>
      <th>Location</th>
```

Scanned by CamScanner

```
<th> Job <lth>
<ltr>
<lthead>
{ % for i in data % }
<tbody>
<tr>
<td>{{ i. first_name }}<ltd>
<td>{{ i. last_name }} <ltd>
<td> {{ i. email }} <ltd>
<td> {{ i. salary }} <ltd>
<td>{{ i. location}} <ltd>
<td> {{ i. job } } <ltd>
<ltr>
<ltbody>
{ % endfor % }
<ltable>
<lcenter>
{ % else % }
<h2>Emp Fake Data is not Available <lh2>
{ % endif % }
<lbody>
<lhtml>
```

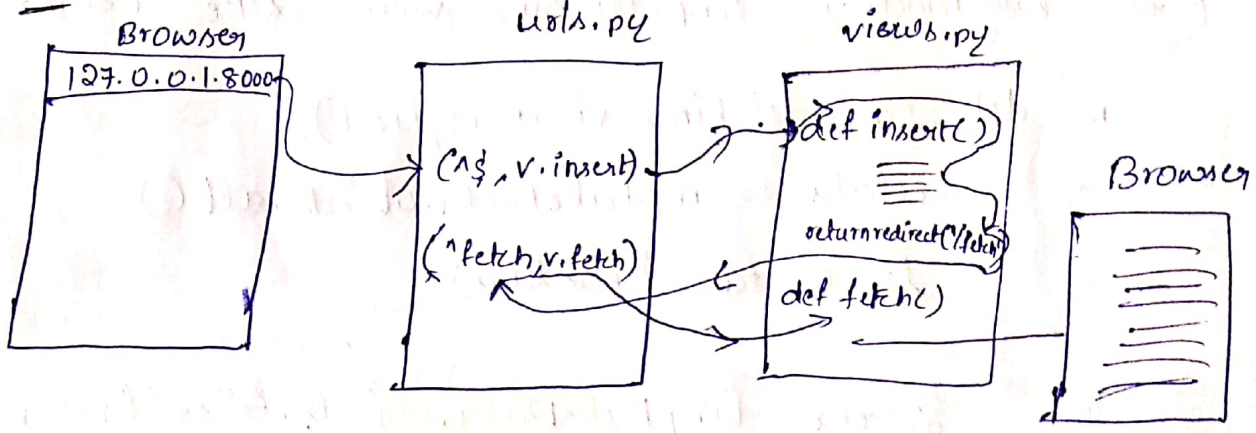
Scanned by CamScanner

Step9 : Go to urls.py file

```
from django.conf.urls import url
from django.contrib import admin
from fakerapp import views
urlpatterns = [
    url(r'^$', views.emp_inserting_view),
    url(r'^fetch/', views.emp_fetching_view),
]
```

Step10 : run makemigrations, migrate and runserver commands.

Flow :



ORM operations.

* Object Relational Mapping

Django model support ORM language to write the queries, to interact with database table so that we can perform any kind of operation on the database table through models.

Scanned by CamScanner

* We can perform so many operations (filters) on a table like

--contains =
--endswith =
--startswith =
--search =
--exact =
--in =
--icontains =
--iendswith =
--iexact =

* We can also perform operations based on single field or based on multiple fields

Note: We modify emp-fetching-view like below

```
def emp-fetching-view(request):
    edata = EmpFakeData.objects.all()
    len_edata = len(edata)

    fdata = EmpFakeData.objects.filter(first_name__
                                startswith = 'A')
    len_fdata = len(fdata)
    return render(request, 'emp_fake_data.html',
                  {'fdata': fdata,
                   'len_edata': len_edata,
                   'len_fdata': len_fdata})
```

Scanned by CamScanner

Note: To perform ORM operation by modifying 4th line.

* Go to html file and add the following 2 lines after h1 tag or after title tag.

```
<h1> Employee Data </h1>
<h2> Total No of Rows: {{ len_edata }} </h2>
<h2> Total No of FilteredRows {{ len_fdata }} </h2>
```

Write a query (or) ORM code to display all employees data whose name starts with 'A' character.

ORM:

```
fdata = EmpFakeData.objects.filter(first_name__startswith='A')
```

ORM-SQL:

we go to python shell to get sql code query of above ORM

```
>>> python manage.py shell
>>> from fakerapp.model import EmpFakeData
>>> fdata = EmpFakeData.Objects.filter(first_name__startswith='A')
>>> str(fdata.query)
'SELECT
    'fakerapp_empfakedata'.'id',
    'fakerapp_empfakedata'.'first_name',
    'fakerapp_empfakedata'.'last_name',
```

Scanned by CamScanner

'fakerapp_emplakedata'.'email',

'fakerapp.emplake_data'.'location',

'fakerapp_emplake_data'.'salary',

'fakerapp_emplakedata'.'job',

FROM 'fakerapp_emplake data'

WHERE 'fakerapp_emplakedata'.'first_name' LIKE BINARY 'A%'

SQL

Go to mysql and run the following query

```
mysql > select * from fakerapp_emplakedata where
        first_name like 'A%';
```

Write ORM code to display all employees data whose name ends with 'e' character

ORM

```
fdata = EmpFakeData.objects.filter(first_name__endswith='e')
```

ORM_SQL

```
'SELECT 'fakerapp_emplakedata'.'id',
        'fakerapp_emplakedata'.'first_name',
        "      "            .'last_name',
        "      "            .'email',
        "      "            .'location',
        "      "            .'salarly',
        "      "            .'job'.

FROM 'fakerapp_emplakedata'
```

Scanned by CamScanner

WHERE 'fakerapp_empfakedata'. 'first_name'
LIKE BINARY '%e'

SQL

mysql > select * from fakerapp_empfakedata where first_name like '%e';

* Write ORM code to display all the employees details whose name contains 'A' character.

Note: if we use first_name__contains = 'a' then it will take all the names which are having lower case 'a' only.

* if we use first_name__contains = 'A' then it will take all the names which are having uppercase 'A' only.
* if we want to get all employee names whose name contains 'a' or 'A' then we use first_name__icontains = 'a', here icontains filter will ignore the case.

ORM

fdata = EmpFakeData.objects.filter(first_name__icontains = 'a')

Write ORM code to display all the employees who are getting above 12000 Rs. salary.

fdata = EmpFakeData.objects.filter(salary__gt = 12000)

Write ORM code to display all the employees who is getting 15000 or above 15000 salary

fdata = EmpFakeData.objects.filter(salary__gte = 15000)

Scanned by CamScanner

Display all the employees who is getting 21000 salary

fdata = EmpFakeData.Objects.filter(salary__exact = 21000)

Display all the employees who are getting either 10000 or 15000 salary

fdata = EmpFakeData.objects.filter(salary__in = (10000,15000))

Write ORM code to display all the employees whose salary is between 12000 and 17000

fdata

Note: Here both start value and endvalue are within the range only

fdata = EmpFakeData.objects.filter(salary__range = (12000, 17000))

Filtering or displaying required fields or fieldnames

We use values method to specify the required field names

* if we don't want to display any required fields on the browser, those names we don't specify in the values method and also those names and in thead tags and variables in the tbody tags should be commented in the html file.

Write ORM code to display first_name, last_name and location of the employees who are working in hyderabad location

Note: Here we don't take email, salary and job fields in values method and these three column names and variables should be commented

Scanned by CamScanner

in the html file

fdata = EmpFakeData.objects.filter(location__iexact='Hyderabad').values('first_name','last_name','location')

Display employee first_name. whose salary is 21000

fdata = EmpFakeData.objects.filter(~~first~~ salary__exact=21000).values('first_name')

OR operations

We can implement 'or' operations in different ways.

1. <Queryset1> | <Queryset2>
2. (Q(filter1) | Q(filter2))

Note: Here Q represents query we have to import Q

from django.db.models import Q

Display all the employees whose salary is 21000 or whose location is bangalore

fdata = EmpFakeData.objects.filter(salary__exact=21000) | EmpFakeData.objects.filter(location__iexact='Bangalore')

(or)

fdata = EmpFakeData.objects.filter(Q(salary__exact=21000) | Q(location__iexact='Bangalore'))

Display all the employees whose name starts with 'K' character or whose job is 'PM' are working in Pune location

Scanned by CamScanner

```
fdata = EmpFakeData.objects.filter(Q(first_name__
                startswith='K')|Q(~~location~~job__
                iexact='Pune'))|
                Q(location__iexact='Pune'))
```

Write ORM to display the first_name and last_name of the employees whose ~~nam~~ last_name contains 'K' character or who is working in PUNE or whose salary is 15000

```
fdata = EmpFakeData.objects.filter(Q(~~first_na~~
                last_name__contains='K')|Q(location
                __iexact='Pune')|Q(salary__iexact
                = 15000).
                values('first_name','last_
                name')
```

And operations

We can perform 'and' operations in different ways.

1.) <Queryset1> & <Queryset2>

2.) (Q(filter1) & Q(filter2))

3.) (filter1, filter2)

Write ORM code to display all the employees who are working in Pune as PM

```
fdata = EmpFakeData.objects.filter(job__iexact='pm') &
        EmpFakeData.objects.filter(location__iexact='pune')
```

(or)

Scanned by CamScanner

fdata = EmpFakeData.objects.filter(Q(job__iexact='pm') & Q(location__iexact='pune'))

(or)

fdata = EmpFakeData.objects.filter(job__iexact='pm', location__iexact='pune')

1. Display all the employees whose salary is above 15000 and working in 'hyderabad'

fdata = EmpFakeData.objects.filter(salary__gt=15000, location__iexact='Hyderabad')

2. Display the first_name and last_name of the employees whose name endswith with 'm' character and getting 10000 and working in 'Chennai' location

fdata = EmpFakeData.objects.filter(last_name__endswith='m', salary__iexact=10000, location__iexact='Chennai').values(first_name, last_name)

Write ORM code to display the first_name, job and salary of the employees whose salary is 10,000 and working as 'pm' in bangalore location

fdata = EmpFakeData.objects.filter(salary__exact=10000, job__iexact='pm', location__iexact='bangalore').values(first_name, job, salary)

Scanned by CamScanner

Indexing and slicing

Indexing is nothing but getting a specific row by using its index number

Slicing is nothing but fetching sequence of rows by using their start and end index numbers

Write ORM code to display first 10 employees details from the table.

```
fdata = EmpFakeData.objects.filter()[0:10]
```

Order by method

It is used to sort the data either in ascending or descending order

** [It allows one numerical field] x

* It allows one or more fields

* If we give two fields then it gives first priority to the first field then second priority for the second field.

```
order_by("salary")   ---- in asc order
order_by("-salary")  --- in desc order
```

Not operations

We can perform not operations in different ways

1. exclude(<filter>)
2. filter(~Q(<filter>))

Display all the employees details who doesnot belong to pune location

Scanned by CamScanner

location__iexact

fdata = EmpFakeData.objects.exclude('Pune')

fdata = EmpFakeData.objects.filter(~Q(location__iexact = 'Pune'))

Display ORM code to display all the employees who are not taking 10000 and 21000 salaries.

fdata = EmpFakeData.objects.exclude(salary__in = (10000, 21000))

Creating search button or search box

Go to the views file and modify emp_fetching_view like below

```
def emp_fetching_view(request):
    if request.method == "POST":
        loc = request.POST.get('loc', '')
        edata = EmpFakeData.objects.all()
        len_edata = len(edata)
        fdata = EmpFakeData.objects.filter(location__iexact = loc)
        len_fdata = len(fdata)
        return render(request, 'emp_fake_data.html',
                      {'fdata': fdata,
                       'len_edata': len_edata,
                       'len_fdata': len_fdata})
```

Scanned by CamScanner

```
else :
                    fakedata
    edata = Emp.objects.all()

    len_edata = len(edata)

    fdata = EmpFakeData.objects.all()

    len_fdata = len(fdata)

    return render(request, 'emp_fake_data.html',
                                    {'fdata': fdata,
                                     'len_edata': len_edata,
                                     'len_fdata': len_fdata})
```

Go to html file and create a form on top of table opening tag

```
<form method = "POST">
    {% csrf_token %}
    <input type = "text" name = "loc" placeholder
                                = "Enter location">
    <input type = "submit" value = "Search">

</form>
```

Scanned by CamScanner

Employee Data

Total Number of Rows : 4001

Total Number of Filtered Rows : 4001

Enter Location | Search

FirstName	LastName	Email	Salary	Location	Job
Angela	Williams	emilybrown@gmail.com	12000	Bangalore	PM

Template tags and filters

* Template tags and filters are used to check the data in the template files
* We can use any number of tags and filters in a single html file
* Generally if {%if%} and for tags must have ending tags
* These two are considered as template tags
* The result of template tags and filters will be effected on the browser
* Modify emp_fetching_view in the above example like below

Ex1:

```
def emp_fetching_view(request):
    lst = [10,20,30,40]
    return render(request, 'temp_filters.html', {'lst':lst})
```

Scanned by CamScanner

Create temp_filters.html file

```
<html>
<head>
<title> Template tags and filters </title>
</head>
<body>

{% if lst %}
<h1> The data is available </h1>
{% else %}
<h1> The data is not available </h1>
{% endif %}

</body>
</html>
```

output:

The data is available

Ex2: If lst = [ ] in views then the output is data is not available

Ex3: views.py file

```
def emp_fetching_view(request):
    lst = ['Durga', 'Hyderabad', 'Odisha', 'Pune',
           'Mumbai', 'Delhi']
    return render(request, 'temp_filters.html', {'lst': lst})
```

Ex. HTML file

Scanned by CamScanner

```
<html>
<head>
    <title> Template tags and filters </title>
</head>
<body>
 {% if lst %}
      {% for i in lst %}
          {{ i }} <br>
      {% endfor %}

 {% else %}
 <h1> Data is not available </h1>
 {% endfor %}
</body>
</html>
```

Ex:4 : If we take {{ i|upper }} in the above template

Output :

```
DURGA
HYDERABAD
ODISHA
PUNE
MUMBAI
DELHI
```

Ex:-5 : If we take {{ i|lower }} in the above template.

Scanned by CamScanner

Output : durga

hyderabad

odisha

pune

mumbai

delhi

Ex: 6. If we take {{ i|length }} in the above template then it finds length of each name

Output : 5

9

6

4

6

5

Ex:7 Changing the date formats

views.py file

```
import datetime as dt

def emp_fetching_view(request):

    current_date = dt.datetime.now()
    return render(request, 'temp_filters.html',
                  {'current_date': current_date})
```

html. file

```
<html>
<head>
    <title> Template tags and filters </title>
</head>
<body>
```

Scanned by CamScanner

```
{% if current_date %}
    <h1> The current date is {{ current_date|
                date:"d-M-Y "}} </h1>

{% endif %}
</body>
</html>
```

Ex :

```
<html>
<head>
    <title> Template tags and filters </title>
</head>
<body>
{% if current_date %}
    <h1>The current date is {{ current_date|
                date:"d-M-Y "}} </h1>

    <h1> Today is {{ current_date | date : 'z3'}}
                day of this year </h1>

    <h1> Today is {{ current_date|date : 'l'}} </h>

    <h1>Now it is {{ current_date|date :"h : i : sA"}}
                                                  </h1>
{% endif %}
</body>
</html>
```

Output :

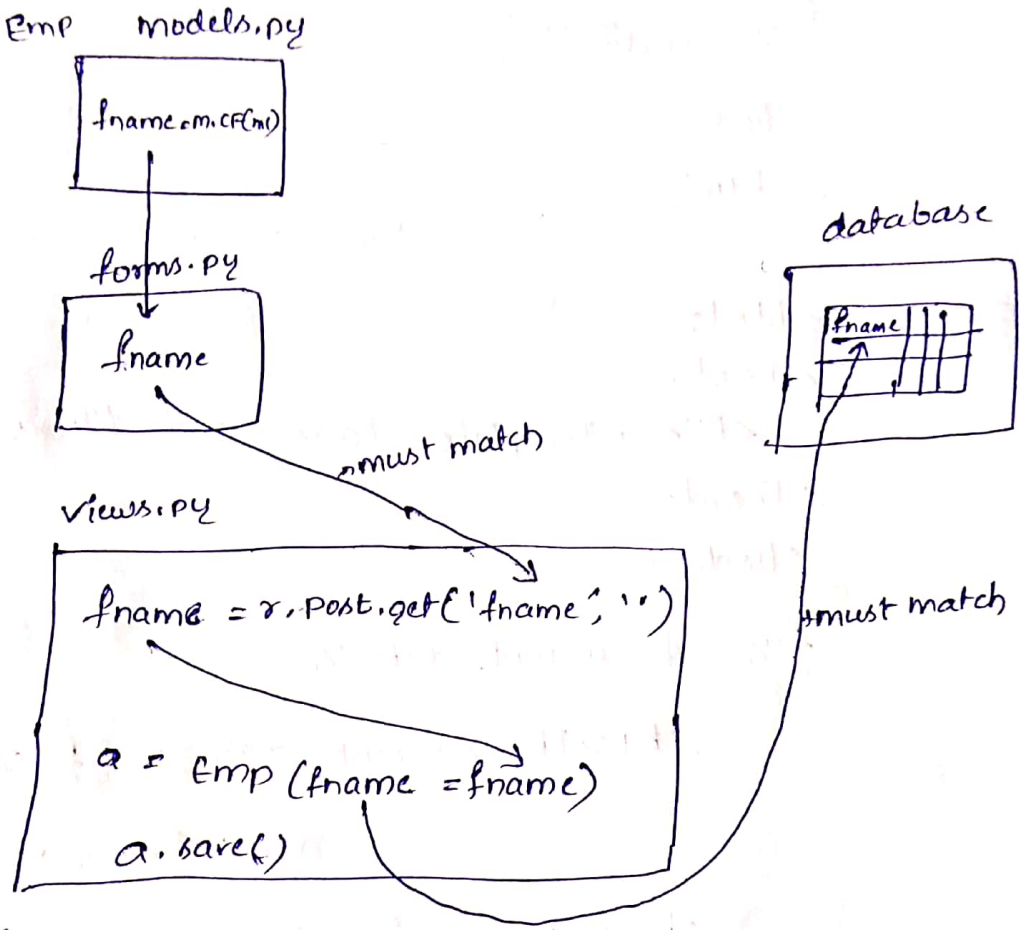
The current date is 27th - Mar - 2019

Scanned by CamScanner

Today is 86th day of this year

Today is Wednesday

Now it is 05 : 19 : 55 PM



Scanned by CamScanner

Student Registration form

We can create a form by using Django code (forms.py) file & also by using UI code (html file)

Step1 : Project Name : Studentregformpro

Step2 : App Name : Studentregformapp

Step3: Database Name : studentregform4pmdb

Step4 : Configure the Database in the settings.py file.

Step5 : Create a model in models.py file.

```
from django.db import models

class studentData(models.Model):

    roll_number = models.IntegerField(primary_key = True)

    student_name = models.CharField(max_length=50)

    mobile_number = models.BigIntegerField()

    fee = models.IntegerField

    email = models.IntegerField()

    email = models.EmailField(max_length = 50)

    dateof birth = models.DateField(max_length =100)
```

Scanned by CamScanner

```
gender = models.CharField(max_length=50)
courses = models.CharField(max_length=1000)
institute_name = models.CharField(max_length=50)
address = models.TextField(max_length=500)
```

Step6 :- Go to views.py file

```
from django.shortcuts import render
from .models import StudentData
def studentreg_view(request):
    if request.method == "POST":
        roll = request.POST.get('roll', '')
        sname = request.POST.get('sname', '')
        mobile = request.POST.get('mobile', '')
        fee = request.POST.get('fee', '')
        email = request.POST.get('email', '')
        address = request.POST.get('address', '')
        dob = request.POST.get('dob', '')
        gender = request.POST.get('gender', '')
        course = request.POST.get('course', '')
        institute = request.POST.get('institute', '')
        data = StudentData(
            roll_number = roll,
            student_name = sname,
```

Scanned by CamScanner

```
mobile_number = mobile,
fee = fee,
email = email,
address = address,
dateofbirth = dob,
gender = gender,
course = course,
institute_name = institute
)
data.save()
return render(request, 'studentregform1.html')
return render(request, 'studentregform1.html')
```

Step7: Create studentregform1.html in templates folder

```
{% load staticfiles %}
<html>
<head>
    <title> Student Registration Form </title>
    <link rel = "stylesheet" href = "{% static 'css/
                                        studentreg.css' %}">
# copy and paste bootstrap lines
</head>
<body>
<div class = "container">
```

Scanned by CamScanner

```
<div class = "row">
    <div class = "offset-md-3 col-md-6">
        <form method = "POST">
            <h3 class = "text-success">Student
                Registration Form</h3>
            {% csrf_token %}
            <label>RollNumber :</label>
            <input type = "number" name = "roll"
                autocomplete = "off" placeholder =
                "Your RollNumber" ><br>
            <label> Your Name </label>
            <input type = "text" name = "sname".
                autocomplete = "off" placeholder =
                "Your Name" ><br>
            <label> Mobile Number </label>
            <input type = "number" name = "mobilenumber"
                autocomplete = "off" placeholder =
                "mobile" ><br>
            <label> Fee Amount </label>
            <input type = "number" name = "fee".
                value = "2000" readonly <br>
```

Scanned by CamScanner

```
<label> Email ID : </label>
<input type ="email "name = "email " autocomplete=
                "off " placeholder = "Email"<br>
<label >Date of Birth </label>
<input type = "date" name = "dob" <br>
<label >Gender </label><br>
    <input type ="radio " name = "gender " value ="male"
                class = "gcls "> Male <br>
    <input type = "radio" name = "gender" value = "female"
                class ="gcls "> Female <br>
    <label> Courses </label>
    <input type = "checkbox" name = "course" value =
                "python "> Python <br>
    <input type = "checkbox" name = "course " value =
                "Django "> Django <br>
    < input type = "checkbox" name = "course" value =
                " UI "> UI <br>
    <input type = "checkbox" name = "course" value =
                " RestAPI "> RestAPI <br>
```

Scanned by CamScanner

```
<label> Institute Name: </label>
    <select name = "institute" >
        <option value = "durgasoft" > Durgasoft <br>
                                    </option >
        <option value = "djangosoft" > Djangosoft </option>
        <option value = "pythonsoft" > Pythonsoft </option>
        <option value = "UISoft " > UISoft </option >
        </select> <br>
<label> Address: </label>
    <input type = "text" name = "address " autocomplete
            = " off " placeholder = " Address " > <br><br>
    <input type = " submit " value = "submit " class =
                    " btn btn-success " >
    <input type = " reset " value = " clear " class =
                    " btn btn-danger " >
    </form>
    </div> </div > </div>
</ /body >
</ /html>
```

Steps : Create static folder, create CSS folder

Scanned by CamScanner

Create studentreg.css file

```
form {
    margin-top : 30px;
}
h3 {
    text-align : center;
}
input[name] {
    width : 200px;
}
label {
    font-family : cursive;
}
#gtr {
    border : 2px red solid;
}
.gcls {
    border : 2px red solid;
    word-spacing : -60px;
}
```

Scanned by CamScanner

Step9 : Go to urls.py file

```
from django.conf.urls import url
from django.contrib import admin
from studentregformapp import views

urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^$', views.studentreg_view),
]
```

Step10 :- Run makemigrations, migrate and runserver commands.

Scanned by CamScanner

Working with MultiSelectField module

1. multiselectfield is a python module which is used to take multiple values from the user
2. Generally multiselectfield provides two different classes
   1. MultiSelectField
   2. MultiSelectformfield

* MultiSelectField is used in models file
* If we want to create any field which is providing multiple options to user then we use MultiSelectField

Ex:- Preferred_locations, Preferred_positions, required_courses, skills ...

* MultiSelectformfield is used in forms.py file

Syntax: Locations = (

('Hyd', 'Hyderabad'),

('Bang', 'Banglore')

)

↓ Storing value in the database

↓ Displaying value on the browser

Create employeeregistrationform by using multiselectfield module

Scanned by CamScanner

Step1: foldername : febdj4pmd

Step2: Project Name : empregistrationpro

Step3: App Name : empregistrationapp

Step4: Database Name : empregistration4pmdb

Step5: Configure the database in the settings.py file.

Step6: Go to models.py file

```
from django.db import models
from multiselectfield import MultiselectField
class EmpRegistrationData(models.Model):
    emp_name = models.CharField(max_length=50)
    emp_email = models.EmailField(max_length=50)
    LOCATIONS_CHOICES = (
        ('Hyd', 'Hyderabad'),
        ('Bang', 'Bangalore'),
        ('Che', 'Chennai'),
        ('Odi', 'Odisha'),
    )
    preferred_locations = MultiselectField(max_length=150,
                                           choices=LOCATIONS_CHOICES)
    POSITION_CHOICES = (
        ('TL', 'Team Lead'),
        ('MGR', 'Manager'),
```

Scanned by CamScanner

```
        ('PM', 'Project Manager'),
    )
    preferred_positions = MultiselectField(max_length=150,
                                           choices = POSITION_CHOICES)

    emp_gender = models.CharField(max_length=100)
    emp_dob = models.DateField(max_length=100)
```

Step7 :- Create forms.py file

```
from django import forms
from multiselectfield import MultiselectFormField

class EmpRegistrationForm(forms.Form):
    emp_name = forms.CharField(
        label = "Enter your name",
        widget = forms.TextInput(
            attrs = {
                'class' : 'form-control',
                'placeholder' : 'Your Name'
            }
        )
    )

    emp_email = forms.EmailField(
        label = "Enter your Email",
        widget = forms.EmailInput(
```

Scanned by CamScanner

```
        attrs = {
            'class' : 'form-control',
            'placeholder' : 'Enter Your Email'
            }
        )
    )

LOCATIONS_CHOICES = (
        ('Hyd', 'Hyderabad'),
        ('Bang', 'Bangalore'),
        ('Che', 'Chennai'),
        ('Odi', 'Odisha'),
)

preffered_locations = MultiselectFormField (max_length=100,
                                            Choices = LOCATIONS_CHOICES)

POSITION_CHOICES = (
        ('TL', 'Team Lead'),
        ('MGR', 'Manager'),
        ('PM', 'ProjectManager'),
)

preffered_positions = MultiselectFormField (max_length = 100,
                                            choices = POSITION_CHOICES)
```

Scanned by CamScanner

```
GENDER_CHOICES = (
        ('M', 'Male'),
        ('F', 'Female'),
)
emp_gender = forms.ChoiceField(
    widget = forms.RadioSelect(), choices = GENDER_CHOICES
)
y = range(2019, 1960, -1)
emp_dob = forms.CharField(
    widget = forms.SelectDateWidget(years = y)
)
```

Steps :- Go to views.py file

```
from .models import EmpRegistrationData
from .forms import EmpRegistrationForm
def home(request):
    if request.method == "POST":
        eform = EmpRegistrationform(request.POST)
        if eform.is_valid():
            emp_name = eform.cleaned_data.get('emp_name', '')
            emp_email = eform.cleaned_dat.get('emp_email', '')
```

Scanned by CamScanner

```
preferred_locations = eform.cleaned_data.get('preferred_
                                locations','')

preferred_positions = eform.cleaned_data.get('preferred_
                                positions','')

emp_gender = eform.cleaned_data.get('emp_gender','')

emp_dob = eform.cleaned_data.get('emp_dob','')

        data = EmpRegistrationData(
                emp_name = emp_name,
                emp_email = emp_email,
                preferred_locations = preferred_locations,
                preferred_positions = preferred_positions,
                emp_gender = emp_gender,
                emp_dob = emp_dob,
        )
        data.save()
        eform = EmpRegistrationForm()
        return render(request, 'empreg.html', {'eform':
                                eform})

else:
    eform = EmpRegistrationForm()
    return render(request, 'empreg.html', {'eform':
                                eform})
```

Scanned by CamScanner

Step9 : Create empreg.html in templates folder

```
<html>
<head>
    <title> EmpRegistration Form </title>

# copy and paste bootstrap lines

</head>
<body>
<div class = "container">
    <div class = "row">
        <div class = "offset-md-3 col-md-6">
        <form method = "POST">
        <h2> Register your Details :. </h2>
        {% csrf_token %}
        {{ eform }} <br>
        <input type = "submit" value = "Register">
        <input type = "reset" value = "Clear">
        </form>
    </div>
</div>
</div>
</body>
</html>
```

Step10 : Go to urls.py file

Scanned by CamScanner

```
from django.conf.urls import url
from django.contrib import admin
from empregistrationformapp import views

urlpatterns = [
    url(r'^admin/', admin.site.url),
    url(r'^$', ~~hom~~ views.home),
]
```

Run makemigration, migrate and runserver commands.

Register Your Details ...

Enter Your Name :

[ ]

Enter Your Email :

[ ]

Preffered locations:

- ☐ Hyderabad
- ☐ Bangalore
- ☐ Chennai
- ☐ Odisha

Preferred positions:

- ☐ Team Lead
- ☐ Manager
- ☐ Project Manager

Scanned by CamScanner

Emp gender :

. O Male

. O Female

Emp dob: [January] [1] [2019]

[Register] [Clear]

* After giving the or selecting the data, if we click on register button data will save in the database it will open the empty form

ModelForm

Django provides forms to take the data from the user and to save the data in the database.

* Django has different types of form classes

1. forms. Form
2. forms. ModelForm

* forms. ModelForm provides more shortcuts to develop a form but in forms. Form class we write all coding manually which may take more time to develop the application

* we just specify model name and required or all fields under the Meta model in forms.py file if we use forms. ModelForm

* Incase of forms. Form class, we have to write all

Scanned by CamScanner

fields in forms.py file which are already defined in models.py file.

* If we want to create a form by using all the fields of a model then we go for forms. ModelForm class

* In views.py file also, we don't directly save the form variable if it is valid that means we don't take the values from a form into a variables and we don't map the variables to table columns

Step 1 :- foldername : febdj4pm

Step 2 : ProjectName : HospitalRegistrationFormPro

Step 3 : AppName : HospitalRegistrationFormapp

Step 4 : DatabaseName : Hospital4pmdb

Step 5 : Configure the database in settings.py file.

Step 6 : Go to models.py file

```
from django.db import models

class HospitalRegistrationForm(models.Model):
    patient_name = models.CharField(max_length=50)
    patient_age = models.IntegerField( )
```

Scanned by CamScanner

```
patient_problem = models.CharField(max_length=50)
patient_fee = models.IntegerField()
patient_mobile = models.BigIntegerField()
patient_location = models.CharField(max_length=50)
```

Step7 : Create forms.py file

```
from django import forms
from .models import HospitalRegistrationForm

class HospitalRegistrationForm(forms.ModelForm):
    class Meta:
        model = HospitalRegistrationform
        fields = '__all__'
```

Step8 : Go to views.py ~~from~~ file

```
from .models import HospitalRegistrationForm
from .forms import HospitalRegistrationForm

def hospitalRegistrationview(request):
    if request.method == "POST":
        hform = HospitalRegistrationForm(request.
                                         POST)
        if hform.is_valid():
```

Scanned by CamScanner

```
        hform.save()
        hform = HospitalRegistrationForm()
        return render(request, 'hospital.html',
                                {'hform': hform})
    else:
        hform = HospitalRegistrationForm()
        return render(request, 'hospital.html',
                                {'hform': hform})
```

Step9 :- Go to hospital.html in templates folder

```
<html>
<head>
<title> Hospital Registration Form </title>
  # copy and paste bootstrap lines
</head>
<body>
<div class = "container">
    <div class = "row">
        <div class = "offset-md-3 col-md-6">
            <form method = "POST">
            <h2> Enter Patient Details </h2>
            {% csrf_token %}
            {{ hform.as_p }} <br>
```

Scanned by CamScanner

```
<input type = "submit" value = "submit or Register" >
<input type = "reset" value = "clear" >
</form>
</div>
</div>
</div>
</body>
</html>
```

Step10 :- Go to urls.py file

```
from django.conf.urls import include, url
from django.contrib import admin
from HospitalRegistrationForm import views

urlpatterns = [
    url(r'^$', views.Hospitalregistrationview),
]
```

Step11 :- Run makemigrations, migrate and runserver commands

Scanned by CamScanner

Enter Patient Details

Patient_name :

Patient_age :

Patient_problem :

patient_fee :

patient_mobile :

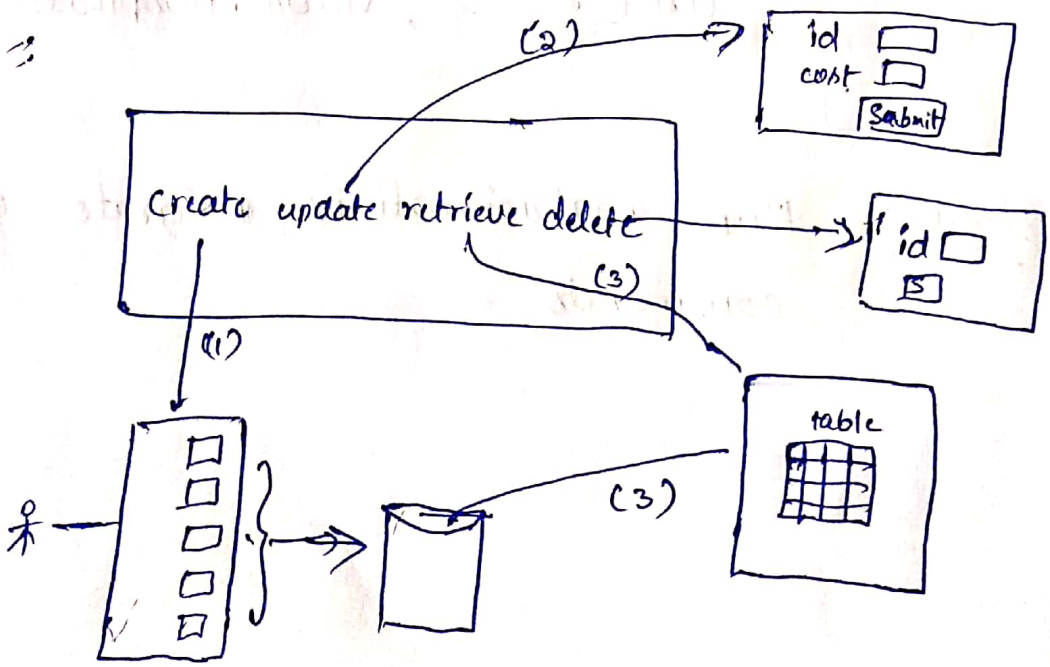
patient_location :

Register Clear

* After giving all the values if we click on register button then all the values will save in the database it will open the empty form

1. Curd Operations

Step1 :-



Scanned by CamScanner

Step1 : foldername : febdj4pm

Step2 : ProjectName : curdpro

Step3 : AppName : curdapp

Step4 : Database Name : curd4pmdb

Step5 : Configure the database in the settings.py.

Step6 : Go to models.py file

```
from django.db import models

class ProductData(models.Model):
    product_id = models.IntegerField( )
    product_name = models.CharField(max_length=50)
    product_cost = models.IntegerField( )
    product_color = models.CharField(max_length = 50)
    product_class = models.CharField(max_length=50)
    customer_name = models.CharField(max_length=100)
    customer_mobile = models.BigIntegerField ()
    customer_email = models.EmailField(max_length=50)
```

Step7 :- Go to forms.py file

```
from django import forms

class ProductInsertform(forms.Form):
    product_id = forms.IntegerField(
    label = "Enter your product Id",
    widget = forms.NumberInput(
        attrs = {
```

Scanned by CamScanner

```
            'class' : 'form-control',
            'placeholder' : 'Your product Id'
            }
        )
    )

    product_name = forms.CharField(
        label = "Enter product name",
        widget = forms.TextInput(
            attrs = {
            'class' : 'form-control',
            'placeholder' : 'Your product name'
            }
        )
    )

    product_cost = forms.IntegerField(
        label = "Enter product cost",
        widget = forms.NumberInput(
            attrs = {
            'class' : 'form-control',
            'placeholder' : 'Your product cost'
            }
        )
    )
```

Scanned by CamScanner

```
product_color = forms.CharField(
        label = "Enter your product color",
        widget = forms.TextInput(
            attrs = {
                'class': 'form-control',
                'placeholder': 'Your product color'
            }
        )
    )

product_class = forms.CharField(
        label = "Enter your product class",
        widget = forms.TextInput(
            attrs = {
                'class': 'form-control',
                'placeholder': 'Your product class'
            }
        )
    )

customer_name = forms.CharField(
        label = "Enter your name",
        widget = forms.TextInput(
            attrs = {
                'class': 'form-control',
                'placeholder': 'Your customer name'
            }
        )
    )
```

Scanned by CamScanner

```
customer_mobile = forms.IntegerField(
        label = "Enter your Mobile Number",
        widget = forms.NumberInput(
                attrs = {
                'class' : 'form-control',
                'placeholder' : 'Your Mobile Number'
                }
        )
)

customer_email = forms.EmailField(
        label = "Enter email Id",
        widget = forms.EmailInput(
                attrs = {
                'class' : 'form-control',
                'placeholder' : 'your email.Id'
                }
        )
)

class UpdateForm(forms.Form):
        product_id = forms.IntegerField(
                label = "Enter your product ID",
                widget = forms.NumberInput(
                        attrs = {
                        'class' : 'form-control',
                        'placeholder' : 'Your product Id'
```

Scanned by CamScanner

```
        }
    )
)
product_cost = forms.IntegerField(
    label = "Enter your product cost",
    widget = forms.NumberInput(
        attrs = {
        'class' : 'form-control',
        'placeholder': 'Your product cost'
        }
    )
)
class DeleteForm(forms.Form):
    product_id = forms.IntegerField(
        label = "Enter your product id",
        widget = forms.NumberInput(
            attrs = {
            'class' = 'form-control',
            'placeholder' : 'Your product id'
            }
        )
    )
```

Scanned by CamScanner

Step 8 : Go to views.py file

```
from .models import ProductData
from .forms import ProductInsertForm,
                   ProductUpdateForm, ProductDeleteForm
from django.http.response import HttpResponse

def home(request):
    return render(request, 'product_home.html')

def product_insert_view(request):
    if insert request.method == "POST":
        iform = ProductInsertForm(request.POST)
    if iform.is_valid():
        product_id = request.POST.get('product_id','')
        product_name = request.POST.get('product_name','')
        product_cost = request.POST.get('product_cost','')
        product_color = request.POST.get('product_color','')
        product_class = request.POST.get('product_class','')
        customer_name = request.POST.get('customer_name','')
        customer_mobile = request.POST.get('customer_mobile','')
        customer_email = request.POST.get('customer_email','')
        data = ProductData(
            product_id = product_id,
```

Scanned by CamScanner

```
                product_name = product_name,
                product_cost = product_cost,
                product_color = product_color,
                product_class = product_class,
                customer_name = customer_name,
                customer_mobile = customer_mobile,
                customer_email = customer_email,
            )
        data.save()
        iform = ProductInsertForm()
        return render(request, 'Product_insert.html',
                                {'iform': iform})
    else:
        iform = ProductInsertForm()
        return render(request, 'Product_insert.html',
                                {'iform': iform})

def product_retrieve_view(request):
    products_data = ProductData.objects.all()
        return render(request, 'product_retrieve.html',
                                {'pdata': products_data})

def product_update_view(request):
    if request.method == "POST":
    uform = ProductUpdateForm(request.POST)
```

Scanned by CamScanner

```
if uform.is_valid():
    product_id = request.POST.get('product_id', '')
    product_cost = request.POST.get('product_cost', '')
    pdata = ProductData.objects.filter(product_id=
                                       product_id)

    if not pdata:
        return HttpResponse('Product is not
                             Available')
    else:
        pdata.update(product_cost = product_cost)
        uform = ProductUpdateForm()
        return render(request, 'product_update.html',
                      {'uform': uform})
  else:
      return HttpResponse('Invalidform')
else:
    uform = ProductUpdateForm()
    return render(request, 'product_update.html', {'uform':
                                                   uform})

def product_delete_view(request):
    if request.method == "POST":
        dform = ProductDeleteForm(request.POST)
        if dform.is_valid():
            product_id = request.POST.get('product_id', '')
            pdata = ProductData.objects.filter(product_id=
                                               product_id)
            if not pdata:
                return HttpResponse('Product is not Available')
            else:
                pdata.delete()
                dform = ProductDeleteForm()
                return render(request, 'product_delete.
                              html', {'dform': dform})
```

Scanned by CamScanner

```
else:
        return HttpResponse('Invalid Form')

else:
    dform = ProductDeleteForm()
    return render(request, 'product_delete.html',
                                    {'dform': dform})
```

Step9 : Go to urls.py file

```
from django.conf.urls import url
from django.contrib import admin
from curdapp import views

urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^$', views.home),
        url(r'^create/', views.product_insert_view),
        url(r'^retrieve/', views.product_retrieve_view),
        url(r'^update/', views.product_update_view),
        url(r'^delete/', views.product_delete_view),
]
```

Step10: Create static folder, create css folder in static folder

Step11: create home.css file

Step12 : Create product_home.html file in templates folder

```
{% load staticfiles %}
<html>
```

Scanned by CamScanner

```
<head>
    <title>Product Home Page</title>
    <link rel="stylesheet" href="{%static 'css/
                                    home.css'%}">
# copy and paste bootstrap lines
</head>
<style>
    .jumbotron {
        background-color : deepink;
        border : 2px solid black;
        border-radius : 20px;
        box-shadow : 10px 10px 10px black, 20px 20px
                                        10px red;
    }
    body {
        background-color : cyan;
    }
    h1 {
        font-size : 100px;
        text-shadow : 3px 3px red, 7px 7px 7px blue;
    }
    .btn {
        border-radius : 10px;
        box-shadow : 2px 2px 2px yellow;
        text-shadow : 2px 2px 2px black;
    }
</style>
```

Scanned by CamScanner

```
<body>
<center><h1> CURD operations </h1></center>
<div class = "jumbotron">
<a href = "/create/" class = "btn bg-success text-white">
Create </a>

<a href = "/update/" class = "btn bg-success text-white">
update </a>

<a href = "/retrieve/" class = "btn bg-success text-white">
retrieve </a>

<a href = "/delete/" class = "btn bg-success text-white">
delete </a>

</div>
</body>
</html>
```

Step13 : Create home.css file

```
.jumbotron {
    margin : 20px 100px 0px 100px;
}
a {
    margin-left : 120px;
    font-size : 30px;
    border : 2px red solid;
}
```

Step14 : Create product_insert.html

```
<html>
<head>
    <title> Product Insert form </title>
    # copy & paste bootstrap lines
```

Scanned by CamScanner

```
</head>
<style>
    label {
        color : red;
        margin-top : -20px;
    }
    #pid {
        margin-top: 10px;
    }
    body {
        background-color : cyan;
    }
    h2 {
        text-align : center;
        font-size : 45px;
    }
    form {
        margin-top : 20px;
        border : 2px solid black;
        border-radius : 20px;
        box-shadow : 10px 10px 5px red, -10px -10px
                                                5px red;
    }
    #homeid {
        margin-top : 200px;
        font-size : 20px;
    }
    a {
        margin-left : 100px;
        color : black;
    }
    h5 {
        margin-top : 200px;
    }
```

Scanned by CamScanner

```
</style>
<body>
 <div class = "container">
    <div class = "row">
       <div class = "col-md-3">
         <h5> welcome to Insert Page </h5>
       </div>
       <div class = "col-md-6">
          <form method = "POST">
            <h2> Enter Product Details </h2>
             { % csrf_token % }
             <label id = "pid" > Enter Product ID: </label>
              {{ iform.product_id }} <br>
             <label > Enter Product Name : </label>
              {{ iform.product_name}} <br>
             <label> Enter Product cost : </ label>
              {{ iform.product_cost}} <br>
             <label> Enter Product color: </ label>
              {{ iform.product_color}} <br>
             <label > Enter Product class : </ label>
              {{ iform.product_class }} <br>
             <label > Enter Customer Name : </ label>
              {{ iform. customer_name}} <br>
            <label > Enter customer mobile : </ label>
              {{ iform . customer_mobile }} <br>
```

Scanned by CamScanner

```
<label> Enter customer email: <label>
    {{ iform.customer_email }} <br>
<center>
<input type="submit" value="Insert" class=
                    "btn btn-success">
<input type="reset" value="clear" class=
                    "btn btn-danger">  <lcenter>

<lform>
<ldiv>
<div class="col-md-3" id="home-id">
    <a href="/#/">Go to Home <la>
<ldiv>
<ldiv>
<ldiv>
<lbody>
<lhtml>
```

Step15 :- Create product_update.html file

```
<html>
<head>
    <title> Product Insert Update Form <lTitle>
    # copy and paste bootstrap lines
<lhead>
<style>
    h2 {
        text-shadow : 2px 2px 2px red, 3px 3px 3px green;
```

Scanned by CamScanner

```
font-size : 70px;
}
body {
    background-color : cyan;
}
label {
    margin-top : -5px;
    font-size : 25px;
    color : red;
    font-family : cursive;
}
form {
    margin-top : 30px;
}
# divid {
    margin-top : 30px;
    font-size : 20px;
}
a {
    color : red;
}
</style>
<body>
<div class = "container">
    <div class = "row">
        <div class = "offset-md-3 col-md-6">
            <form method = "POST">
                <h2> update Product </h2>
```

Scanned by CamScanner

```
{% csrf_token %}
<label> Enter Product ID: </label>
{{ uform.product_id }} <br>
<label> Enter Product cost: </label>
<center> {{ uform.product_cost }} <br>
<input type = "submit" value = "update" class =
                "btn btn-success">
<input type = "reset" value = "clear" class =
                "btn btn-danger">
                </center>

<center><div id = "divid"><a id = "aid" href =
                "/#/"> Go to home </a></div></center>
</form>
</div>
</div>
</div>
</body>
</html>
```

Step16 :- Create product_retrieve.html file

```
<html>
<head>
    <title> product retrieve form </title>
# copy and paste bootstrap lines
</head>
<style>
    table {
```

Scanned by CamScanner

```
    margin-top : 10px;
    width : 120px;
}
th, td {
    padding : 10px 20px 10px 20px;
    text-align : center;
}
#nopro {
    color : red;
    border : 2px red solid;
    height : 400px;
    padding-top : 150px;
    font-size : 50px;
    margin : 10px 300px 10px;
    text-align : center;
    background-image : radial-gradient(blue,
                                yellow, green deppink);
    border-radius : 20px 100px 20px 100px;
}
body {
    background-color : cyan;
}
#divid {
    margin-top : 30px;
    font-size : 20px;
}
a {
```

Scanned by CamScanner

```
        color : red;
    }
    h2 {
        font-size : 70px;
        text-shadow : 2px 2px red;
    }
    tbody : nth-of-the type (even) {
            background-color : darkorchid;
    }
    tbody : nth-of-the type (odd) {
            background-color : orchid;
    }
    thead {
        background-color : red;
    }
    th, td {
        color : white;
    }
</style>
<body>
{% if pdata %}
    <center>
        <h2> Product Details </h2>
    <table border = "2px" >
    <thead>
```

Scanned by CamScanner

```
<tr>
    <th> Product ID </th>
    <th> Product Name </th>
    <th> Product cost </th>
    <th> Product Color </th>
    <th> Product class </th>
    <th> Customer Name </th>
    <th> customer mobile </th>
    <th> customer email </th>
</tr>
</thead>
  {% for i in pdata%}
<tbody>
  <tr>
    <td> {{ i. product_id }} </td>
    <td> {{ i. product_name}} </td>
    <td> {{ i. product_ cost}} </td>
    <td> {{ i. product_color}} </td>
    <td> {{ i. product_class }} </td>
    <td> {{ i. customer_name }} </td>
    <td> {{ i. customer_mobile }} </td>
    <td> {{ i. customer_email }} </td>
  </tr>
</tbody>
```

Scanned by CamScanner

```
{% endfor %}
</table>
</center>
{% else %}
<h2 id = "nopro">Products are not available</h2>
{% endif %}
<center><div id = "divid"><a id = "aid" href =
"/#/">Go to Home</a></div></center>

</body>
</html>
```

Step17 : Create product_delete.html file

```
<html>
<head>
  <title> Product delete form</title>
  # copy and paste boostrap lines
</head>
<style>
    h2{
        font-size : 70px;
        text-shadow : 2px 2px red;
    }
    body{
        background-color : cyan;
    }
```

Scanned by CamScanner

```
label {
    font-size : 25px;
    font-family : cursive;
    color : red;
}
# divid {
    margin-top : 30px;
    font-size : 20px;
}
a {
    color : red;
}
</style>
<body>
<div class = "container">
    <div class = "row">
        <div class = "offset-md-3 col-md-6">
            <form method = "POST">
                <h2> Delete Product </h2>
                {% csrf_token %}
                <label> Enter Product ID: </label>
                {{ dform.product_id }} <br>
                <center>
                <input type = "submit" value = "Delete"
                    class = "btn btn-success">
```

Scanned by CamScanner

```
<input type = " reset " value = "clear" class =
                "btn btn- danger" > </center>
  </form >
  <center> <div id = "divid" ><a id = "aid" href =
                "/#/ " > Go to home </a> </div </center>
  </div>
 </div>
</div>
</body>
</html>
```

Step18 : Run makemigrations, migrate and runserver commands, Click on the IPAddress link then it will open home page.

Go to settings. file and configure static file

```
STATICFILES_DIRS = [
        Os. path. join(BASE_DIR, 'static')
]
```

Outputs:

CURD Operations

Create | Update | Retrieve | Delete

Scanned by CamScanner

1. * If we click on create button, it will open a form to insert the data.

* After giving data if we click on insert button then the data will store in the database and it will open the empty form again.

2. If we click on update button, it will open a form with product_id field and product_cost.

* We enter both product_id and product_cost and click on update button, it will check the product_id in the database, if the product_id is available in the database then it will update the existing product_cost with new product_cost, and it will open again update form.

3. If we click on retrieve button, if the database has the data then it will take all the data and display in table format.

* If the data is not available in the database then it will display "products are not available".

4. If we click on delete button then, it will open a form with product_id field, we enter product_id and click on delete button. it will check the product_id in the database, if the product_id is available then it will delete the product from the database.

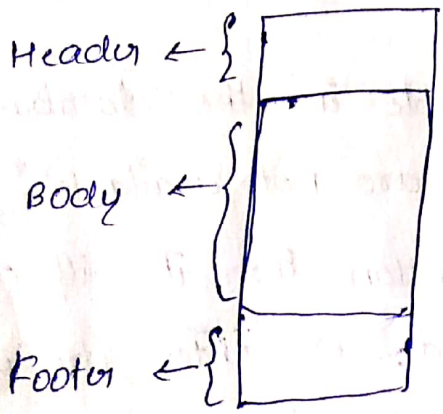
* If the product_id is not available in the database then it will display "products are not available".

Scanned by CamScanner

Template Inheritance

It is the process of deriving tags from one template to another template

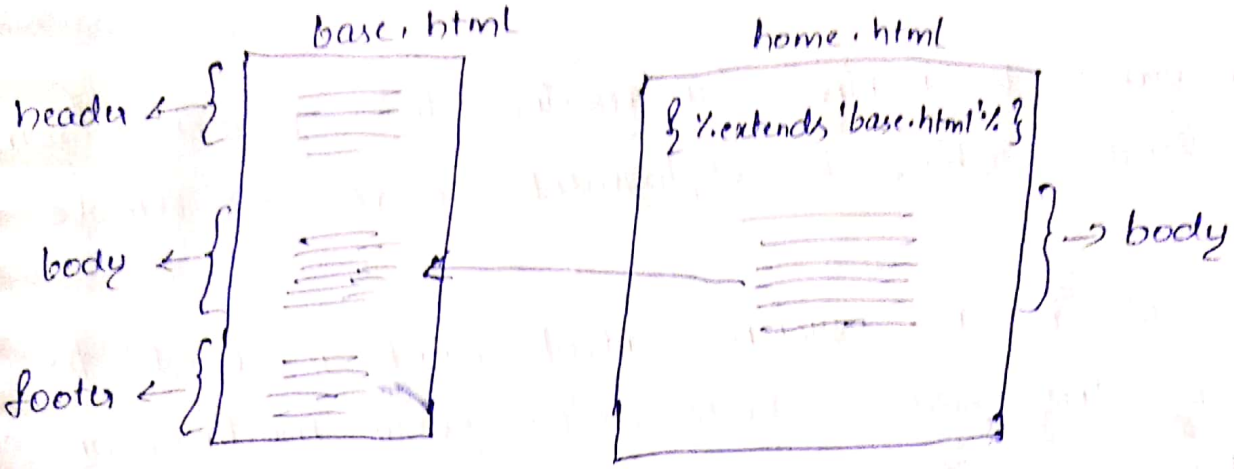
* If a website contains two or more pages and each page contains some common elements then or common data that means the html files contains the common code in each file, it is leading to duplicate code each file.
* So we can create base.html file and the common code can be taken inside the base.html
* We can use base.html in normal html files so that the tags of base.html file can be derived into normal html files.
* Generally a web page contains three parts like below



* The data of header and footer parts will be repeated for every webpage and the body part will be different from each page
* We can give the link between based on html file and normal html files in two different ways.

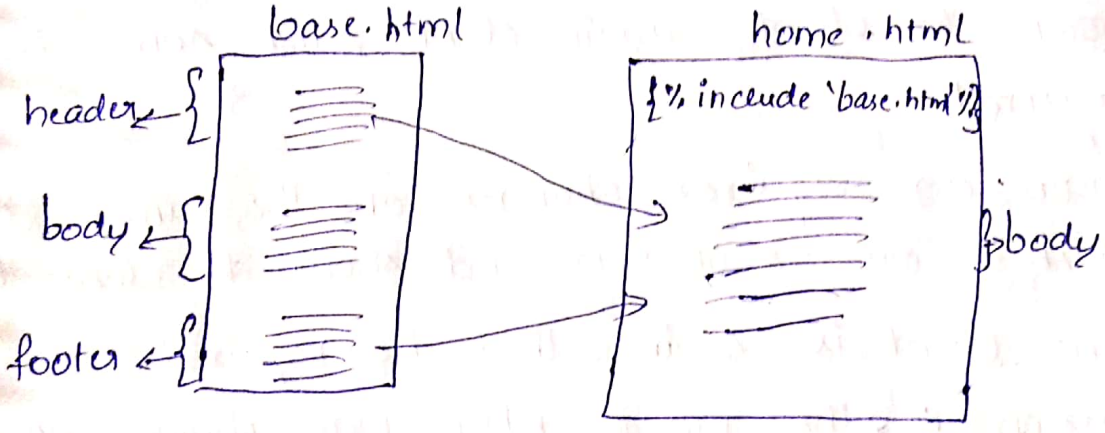
Scanned by CamScanner

1. By using extends Keyword



If we use extends keyword in normal html file then the code of normal html file will go to base.html file and then base.html file will execute

2. By using include Keyword



If we use include keyword then the base.html file will come to the normal.html file and then normal html file will execute

Scanned by CamScanner

Deployment

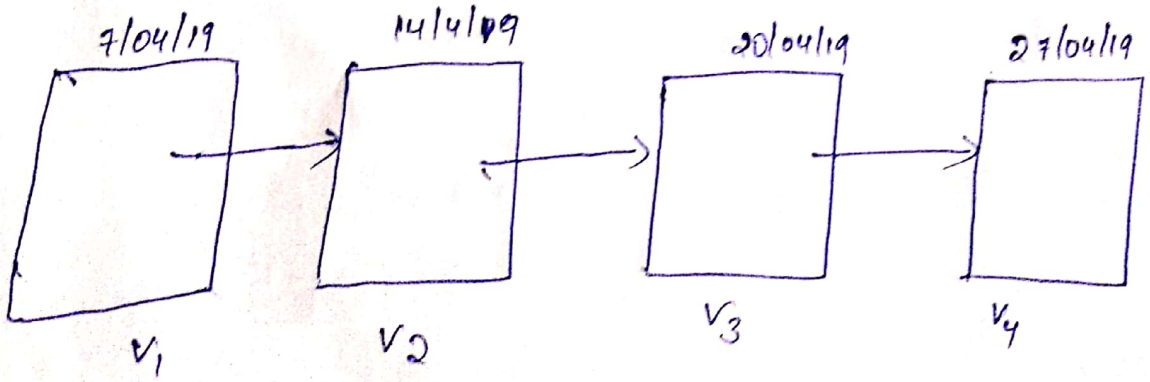
Deployment is nothing but moving the project from our local machine to deployment server or remote servers.

git: git is a version control which is used to control the source code after every modification at runtime

Ex:- * we have finished our project on 07-04-2019 and shown to client, client said some changes are required in the current project

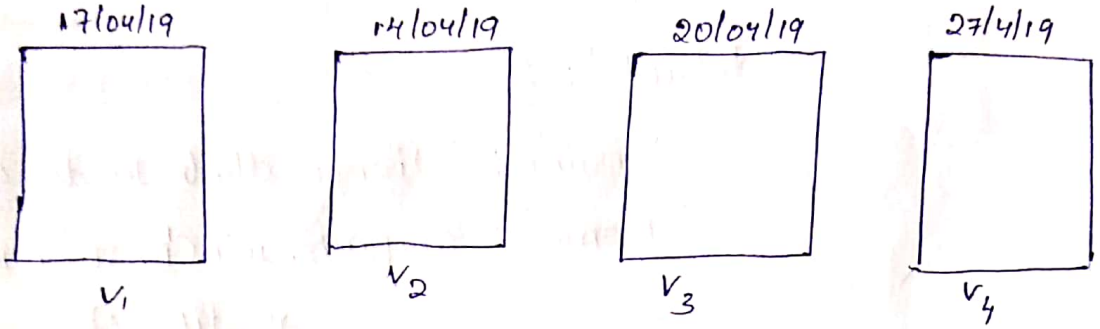
* we do required changes and finished on 14-4-2019 and shown to client, again client said some changes are required

* we again do required changes on the same code and finished on 20-04-2019 and shown to client but this time client is saying that he required the first version of the project which was shown on 07-04-2019 but we don't have that version because we are overriding the existing code each time, so it make more problem to get the first version of the project, like below



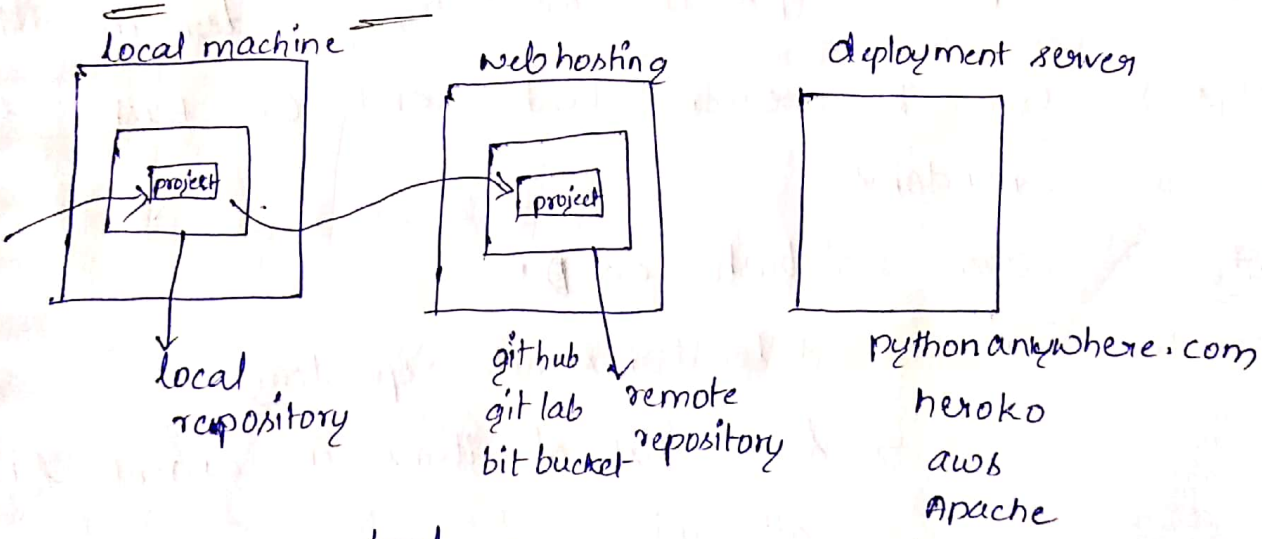
Scanned by CamScanner

Here we use git to control each version after every modification in the source code that means it will create new version after saving the new modification if we do n number of modifications on the same source code git will create n number of versions or commits



now we can give any version to the client whatever he required

The process of deployment



* we create a local repository in our machine and we paste our project in our local respository
* we send the project from local repository to remote repository in web hosting like (github).
* we send the project from web hosting to

Scanned by CamScanner

deployment server

Step1 : Google → download git bash download →
http://git-scm-com/downloads
Click 2.21.0 Download for windows
settings.py → take default server

```
Database = {
    'default': {
        'engine': 'django.db.backends.sqlite3',
        'Name': os.path.join(BASE_DIR,
                             'db.sqlite3')
```

Step2 → Migrate

Step3 → Drive D → foldername : local_institute_responsitory create

Step4 → Copy the project into the folder in drive E.

Step5 → Go to search bash and Git Bash in windows

Step6 → Open gitbash cd D:
→ cd local_institute_repository
→ ls → list of items in current folder
→ git init

~~Step7~~:
→ git add
→ git status
→ git commit -m 'firstcommit'
→ git status

if any changes again
→ git commit -m 'Second commit'.
→ git status

Scanned by CamScanner

git config --global user.name Vanitha

[Create pythonanywhere steps → create remote repository

Create an account

username :

password :

email :

passwordCagain :

→ Register

* Go to Consoles → click on Bash)

* Create Git hubaccount with

* Create remote respository then it will display 5 git links

→ first 3 links are already we run in the git bash

→ copy the 4th line and paste in the git bash and click enter.

→ copy the 5th line and paste in the git bash and click enter

→ Now it may return error 403

→ if error comes then type Credential Manager in the windows search and click on the credential manager

→ Click on windows Credentials

→ Go to Generic Credentials and delete git link

Scanned by CamScanner

again copy the 5th line from the github and paste in the git bash then it will open git hub login page

→ We enter git hub username and password and click on login button

→ Go to gmail account and verify and the github link

→ Go to the github and refresh then we can see our project folder

username remote respository

pythondjango001/remote_institute_repository

Code

clone or download

institute project

first commit

→ Now our project is moved from local repository to remote repository

→ Now create pythonanywhere account in www.pythonanywhere.com

→ ~~More~~ Go to the gmail account and click on the python anywhere link

→ pythonanywhere displays dash board with consoles, files, web, task, databases

→ click on consoles option to install required softwares

Scanned by CamScanner

→ ~~ok~~ create If we click on consoles option then it displays bash and mysql at the left side

→ click on bash option then it will open a console

→ Now we create a virtual environment with name institutefolder and we install python, django, multiselectfield and other required modules like below

```
mkvirtualenv --python=python3.7 institutefolder
(institutefolder)06:53 ~ $ pip install -U django==1.9
(institutefolder)06:56 ~ $ pip install django-multiselectfield
```

→ Go to github click on clone or download button then it displays a link, copy the link

→ Go to pythonanywhere and paste the link like below

```
(institutefolder) 06:56 ~ $ git clone https://github.com/python-
                                  django0001/remote_institute_responsitory.git
```

→ we use list command to see the available versions or software like below

```
(institutefolder) 06:56 : ~ $ pip list
```

→ type ls and change the directories to the our project like below

```
(institutefolder) ~ $ ls
                  $ cd remote_institute_responsity
                  $ ls
                  $ cd instituteproject
                  $ python manage.py makemigrations
```

Scanned by CamScanner

so there are no errors in the console

→ Click on the ~~web~~ browse option in the top right corner select web

→ Click on add new web app

→ Click on next

→ Click on the manual configuration

→ Click on python 3.7

→ Click on next then it opens configuration page

Note we have to run pwd in the console to open the present working directory

→ Go to Code title in the configurationpage and paste the pwd link in the source code option

→ Change the pwd link like below

pwdlink: /home/institute/remote_institute_repository/institutepoject

virtuallink: /home/institute/.virtualenvs/institutefolder

→ Copy the virtual link and paste in Virtualenv title virtualenv in the configuration page & click on reload option

→ Click on institute.pythonanywhere.com link in the same page then it will open a new webpage by displaying hello world message

→ Now we delete the code of helloworld message and then we configure our project

Scanned by CamScanner

→ Go to configuration page, go to code title and click on the link of WSGI configuration file then it opens a new web page

→ Remove all the lines which belongs to helloworld message between 12 to 35 lines

→ go to django option and uncomment

```
import os
import sys

path and modified the path like below
path = '/home/institute/remote_institute_repository/instituteproject'
if path not in sys.path:
        sys.path.append(path)
        os.chdir(path)

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
                      'instituteproject.settings')


from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

→ save the file go to configuration file or page and click on reload option and click on institute.pythonanywhere.com link & displays on the browser.

→ Go to files in the dash board and we can find our project, application and other folders

→ open any required file and make the changes and save the file so that the changes reflect on the web page

Scanned by CamScanner

Our website url is http://insititute.pythonanywhere.com/

Class based views

A view is a python function or python class which takes http request and gives HttpResponse.

* Views are two types
    1. Function based views
    2. Class based views

* Function based are defined by using def keyword but class based is defined by using class keyword
* Class based views have more advantages when compare to function based views because we can use or perform inheritance the class based views
* In function based views we use condition statements to handle the incoming request but we use handler methods in class based views to handle the incoming request

```
def feedback()
    if request.method == "POST":
        - - - -
        - - - -
    elif request.method == "GET":
        - - - -
        - - - -
    elif request.method == "PUT":
        - - - -
        - - - -
    else:
        - - - -
        - - - -
```

```
class feedback()
    def get():
        - - - -
        - - - -
    def post():
        - - - -
        - - - -
    def put():
        - - - -
        - - - -
    def delete():
        - - -
        - - -
```

Scanned by CamScanner

Step1 : foldername : febdjrpm

Step2 : ProjectName : classbasedpro

Step3 : AppName : class genericviews

Step4 : Database Name : generic_db

Step5 : Configure the database in settings.py file

Step6 :- Go to models.py file and create the model

```
from django.db import models
class Person(models.Model):
    name = models.CharField(max_length=50)
    desc = models.CharField(max_length=50)
    def __str__(self):
        return self.name
```

Step7 : Create forms.py file

```
from django import forms
class PersonForm(forms.Form):
    name = forms.CharField(
        widget = forms.TextInput(
            attrs = {
                'class' : 'form_control',
                'placeholder' : 'Enter name',
            }
        )
    )
```

Scanned by CamScanner

```
desc = forms.CharField(help_text = "write a brief
                        about a person",
        widget = forms.Textarea(
            attrs = {
                'class': 'form-control',
                'placeholder': 'Enter description'
            }
        )
    )
```

Steps :- Go to views.py file

```
from django.views import generic
from .models import Person
from .forms import PersonForm

def makeentry(request):
    if request.method == "POST":
        form = PersonForm(request.POST)
        if form.is_valid():
            name = request.POST.get('name', '')
            desc = request.POST.get('desc', '')
        person = Person(name=name, desc=desc)
        person.save()
        form = PersonForm()
        return render(request, 'genericviews/
```

Scanned by CamScanner

```
                    makeentry.html', {'form' : form})
else :
    form = PersonForm()
    return render(request, 'genericviews/makeentry.
                    html', {'form' : form})

class IndexView(generic.ListView):
    context_object_name = 'list'
    template_name = 'genericviews/index.html'
    def get_queryset(self):
        return Person.objects.all()

class DetailsView(generic.DetailView):
    model = Person
    template_name = 'genericviews/detail.html'
```

Step9 : Create genericviews folder in templates folder

* Create makeentry.html file in genericviews folder

```
<html>
<head>
<title> Person Makeentry </title>
 # copy and paste bootstrap lines
<style>
body {
    background-color : blue;
}
.container {
```

Scanned by CamScanner

```
background-color : green;
    border-radius : 10px;
    margin-top : 40px;
}

form {
    border : 5px;
    border-color : red;
}
</style>
</head>
<body>
<div class = "container">
    <h1 class = "text-center">Please make entries here</h1>
    <div class = "row">
        <div class = "col-md-5">
            <form action = "{% url 'genericviews :
                                makeentry' %}" method = "post">
                {% csrf_token %}
                {{ form }} <br>
            <center> <input type = "submit" value = "submit">
                                                    </center>
            </form>
        </div>
    </div>
</div>
</body>
</html>
```

Scanned by CamScanner

Step10: Create Index.html file in genericviews folder

```
{% if list %}
<h1>person Details</h1>
<ul>
  {% for person in list %}
<li>
<a href="{% url 'genericviewsdetails' person
                                id %}">
{{ person.name }}</a>
</li>
  {% endfor %}
</ul>
{% else %}
<h3> No Person found </h3>
{% endif %}
```

Step11: Create details.html file in genericviews folder

```
<h1> {{ person.name }} </h1>
<h3> {{ person.desc }} </h3>
```

Step12: Create urls.py file in the application

```
from django.conf.urls import url
from genericviews import views
app_name = 'genericviews'
urlpatterns = [
        url(r'^$', views.IndexView.as_view(),
                        name = 'index'),
        url(r'^(?P<pk>[0-9]+)/$', views.DetailsView.
                        as_view(), name = 'detail'),
```

Scanned by CamScanner

```
    url(r'^makeentry$', views.makeentry, name='makeentry'),
]
```

Step13 : go to urls file in the project

```
from django.conf.urls import url
from django.contrib import admin
from django.conf.urls import include
urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^genericviews/', include('genericviews.urls')),
]
```

Step14 : Do makemigrations, migrate and runserver commands.

* Go to the browser and add generic views to the existing url

127.0.0.1:2245/genericview

Page not found

Add makeentry to the above url... so that it displays a form to enter the data...

Please make entries here

Name :

Enter name

Desc :

Enter description

write a brief about a person

submit

Scanned by CamScanner

Note: Give some data in the form.

Now remove makeentry from the above url enter then it displays a list of persons.

127.0.0.1:2245/genericviews/

Person Details

- ramya
- sai
- Nani

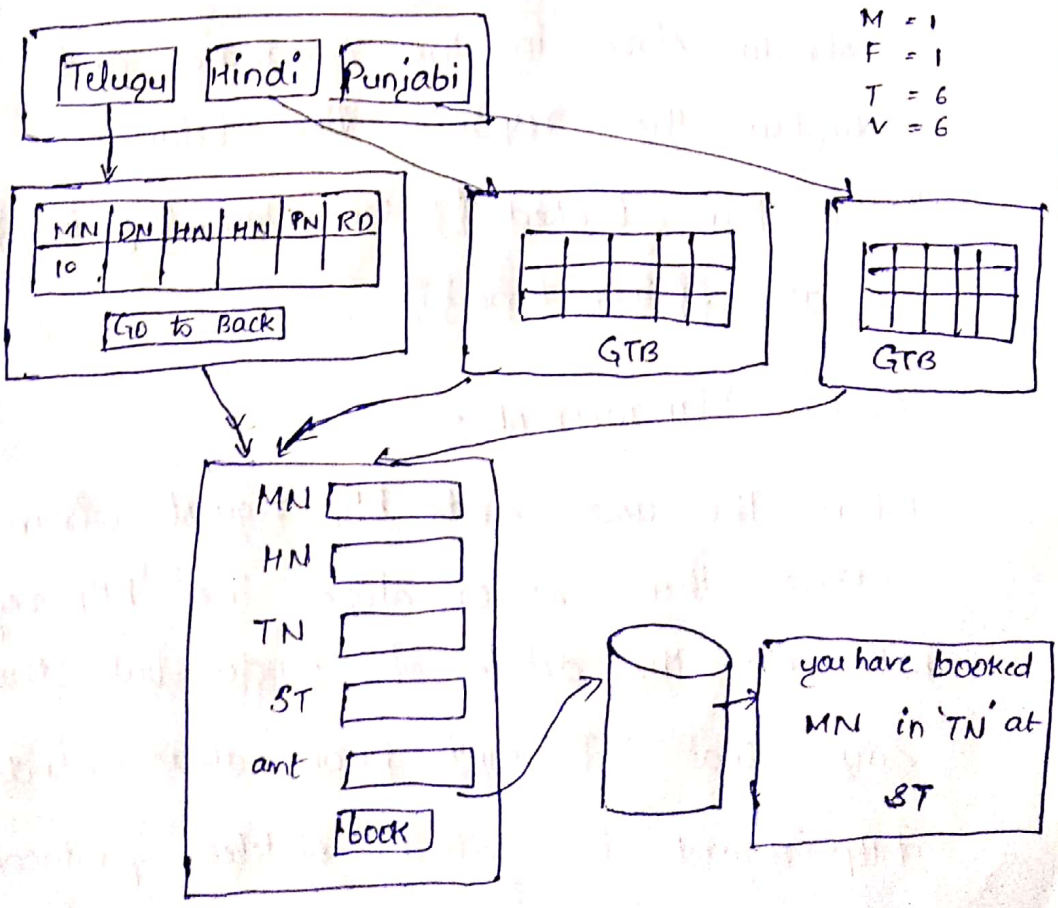
if we click on 'sai' person name then, the id of 'sai' will add in the url of it displays sai name and description like below.

127.0.0.1:2245/genericviews/2/

sai

Django Developer

Movie Project:



Scanned by CamScanner

* Display three buttons like Telugu, Hindi, Punjabi in the jumbotron in the homepage.
* If we click on Telugu button then we need to display atleast to telugu movie details like Director Name, Hero Name, Heroine Name, Producer Name, Released-date columns names. same fields for hindi and Punjabi also
* we have to create a button "want to Book" after the table of movies in each language.
* If we click "want to Book" button then we need display a form with "Movie Name", "Theatre Name", "show Time" and "Amount"
* After the form we have to display a button "Book"
* If we click on "Book" button then the form data should store in the database and we need to display the response like below.

"You have booked {{Movie Name}} in {{Theater Name}} at {{show time}}

Cookie Management:

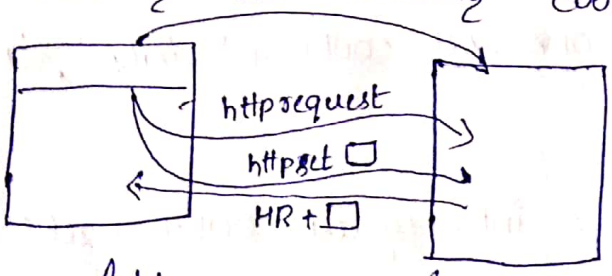
When the user sends http request from browser to server then server stops the http request and it will ask the details of sender but the http request says that "I don't know about sender" that's why http request is called 'stateless' protocol.

* The server allows the http request of platform

Scanned by CamScanner

the operation.

* While http response going to browser, server will add a cookie to the http response.
* The browser takes http response and cookie the browser stores cookies in it
* When the sender makes the same second request, then the browser sends the http request cookie to the server.
* The server takes the sender data from the cookie of stores the sender data and performs operation as per the request.
* Now onwards the browser sends the cookie along with same request 'n' no of times.
* If the sender changes the request then the server creates a new cookie.
* All cookies are stored in the browser.
* If we clear the history of the browser then automatically all existing cookie will expire.



Step1 : foldername : febdjrpm
Step2 : projectName : cookiepro
Step3 : App Name : cookieapp

Scanned by CamScanner

Step 4 : go to views.py file

```
from django.shortcuts import render
from django.http import HttpResponse
def create_cookie(request):
    if not request.COOKIES.get('color'):
        response = HttpResponse("cookie is created")
        response.set_cookie('color', 'blue')
        return response
    else:
        return HttpResponse("Your favourite"
                            "color is {0}".format(request.
                                                  COOKIES['color']))
def count_cookie(request):
    if not request.COOKIES.get('visits'):
        response = HttpResponse("This is your first
                                visit to the site."
                                "from now on I will track"
                                "your visits to this site")
        response.set_cookie('visits', '1')
    else:
        visits = int(request.COOKIES.get('visits')) + 1
        response = HttpResponse("This is your {0} visit".
                                format(visits))
```

Scanned by CamScanner

```
        response.set_cookie('visits', str(visits))
    return response
def delete_cookie(request):
    if request.COOKIES.get('visits'):
        response = HttpResponse("cookies cleared")
        response.delete_cookie("visits")
    else:
        response = HttpResponse("we are not tracking you")
    return response
```

Step 5 : Go to urls.py file

```
from django.urls cont.urls import url
from django.contrib import admin
from cookieApp import views
urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^cookie/', views.create_cookie),
        url(r'^count/', views.count_cookie),
        url(r'^delete/', views.delete_cookie),
    ]
```

Step6 : runserver

O/P1: 127.0.0.1:4411/cookie/
cookie is created

Scanned by CamScanner

O/P 2 : Your favourite color is blue

O/P3 : 127.0.0.1:4411/count/

This is your first visit to the site from now on I will track your visits to this site

O/P4 : 127.0.0.1:4411/count/

This is your 2 visit

O/P5 : 127.0.0.1:4411/delete/

cookies cleared

O/P 6 : 127.0.0.1:4411/delete/

we are not tracking you.

Even though cookies are most useful, but there are some problems with cookies

* Hackers can easily hack the cookies because cookies are storing on the browser, the hackers may break our application
* we cannot store sensitive data like userid, passwords, pin numbers ...
* Generally the browsers will allows the cookies to store only 4KB of data, if more data is available then the data will be broken into multiple cookies Ex:- if the data size is 20KB then it will break into 5 cookies
* Managing multiple cookies is also very complex, because each request and response will carry these multiple cookies.

Scanned by CamScanner

* The cookies will expire when we delete the history of the browser

* We go for sessions to avoid the above problems

Session Management :

When sender sends HttpRequest then the HttpRequest will go to server and server will ask multiple questions about sender but HttpRequest will not give any information about sender, so server allows the HttpRequest and create a memory and django allocates an ID (also called SID or SessionID) to the memory location. At HttpResponse time, server sends HttpResponse and also SID to the browser.

→ So here, the cookies are not created, so the data is more secured because the data is storing on the server

→ SID is 32 characters long random string which is generated by django.

* to create cookie — set_cookie()
* to create session — set_test_cookie()
* to test the session — test_cookie_worked()
* to delete cookie — delete_cookie()
* to delete session — delete_test_cookie()

Scanned by CamScanner

Step1 : foldername : febdj4pm

Step2 : Project Name : Sessionpro

Step3 : App Name : Sessionapp

Step4 : Go to views.py file

```
from django.shortcuts import render
from django.http import HttpResponse
def test_session(request):
    request.session.set_test_cookie()
    return HttpResponse("cookie is set in the
                                server")
def test_delete(request):
    if request.session.test_cookie_worked():
        request.session.delete_test_cookie()
        response = HttpResponse("Session deleted")
    else:
        response = HttpResponse("Session is not
                                available")
    return response
def save_session_data(request):
    request.session['Eno'] = 1001
    request.session['Ename'] = 'Narayana'
    request.session['Language'] = 'Python'
    request.session['Framework'] = 'Django'
```

Scanned by CamScanner

```
return HttpResponse("Session Data Saved")

def access_session_data(request):
    response = ""
    if request.session.get('Eno'):
        response += "Eno : {0} <br>".format(request.
                                    session.get('Eno'))
    if request.session.get('Ename'):
        response += "Ename : {0} <br>".format(request.
                                    session.get('Ename'))
    if request.session.get('Language'):
        response += "Language : {0} <br>".format(request.
                                    session.get('Language'))
    if request.session.get('Framework'):
        response += "Framework : {0} <br>".format(request.
                                    session.get('Framework'))
    if not response:
        return HttpResponse("No session data")
    else:
        return HttpResponse(response)

def delete_session_data(request):
    try:
        del request.session['Eno']
        del request.session['Ename']
```

Scanned by CamScanner

```
        del request.session['Language']
        del request.session['Framework']
    except KeyError:
        pass
    return HttpResponse("Session Data cleared")
```

Step5 : Go to urls.py file

```
from django.conf.urls import url
from django.contrib import admin
from Sessionapp import views
urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^test-session/', views.~~set~~test_session),
        url(r'^test-delete/', views.test_delete),
        url(r'^save/', views.save_session_data),
        url(r'^access/', views.access_session_data),
        url(r'^delete/', views.delete_session_data),
]
```

Step6 : run the server

O/p1 : 127.0.0.1:3320/test-session

cookie is set in the server

Note: Here session is created and displaying the above message.

Scanned by CamScanner

O/P 2 : 127.0.0.1:3320/ test-delete

session deleted

Note: It will verify for any existing session in the server by using test_cookie_worked (), if any session is available then it will delete the session and displayed the above message.

O/P 3 : 127.0.0.1.3320 / save

Session Data saved

Note: It will save the data in the session and displayed the above message.

O/P 4 : 127.0.0.1.3320/ access

Eno : 1001

Ename : Narayana

Language: Python

Framework: Django

O/P 5 : 127.0.0.1.3320/delete

Session Data cleared

Virtual Environment

Virtual Environment is a place where we can have the copy of the python folder (base folder or physical environment)

* when we create multiple projects on same physical environment then it will become more burden on the physical environment when we run all projects because all projects will hit the

Scanned by CamScanner

the same physical environment and get required packages to create and run the current project.

* To avoid this problem, we can create one or more virtual environments for all the projects so that whenever we run the projects then go to the corresponding virtual environment and they will get the required packages from the corresponding virtual environments.

* we can create multiple projects for a single virtual environment

* we have to use virtualenv tool to create virtual environment folder

* virtualenv tool should be downloaded from commant prompt by using pip command

```
C:\Users\Narayana > pip install virtualenv
```

Step1 : Create a folder with name "Django_VirtualEnv"

Step2 : Open Django_VirtualEnv folder and type cmd to open command prompt with current location.

Step3 : Create virtual environment folder "myenv" by using virtualenv tool like below

```
E:\Django_VirtualEnv > virtualenv myenv
```

Step4 : we have to activate the virtual environment like below

Scanned by CamScanner

E:\Django_VirtualEnv > cd myenv

E:\Django_VirtualEnv\myenv\cd Scripts

E:\Django_VirtualEnv\myenv\Scripts > activate

(myenv) E:\Django_VirtualEnv\myenv\Scripts >

Step5 : Now we can create a project in the main folder

(myenv) E:\Django_VirtualEnv\myenv\Scripts > cd..

(myenv) E:\Django_VirtualEnv\myenv > cd..

(myenv) E:\Django_VirtualEnv > django-admin startproject mywebsite

Step6 : After doing all operations we can deactivate the virtual environment.

(myenv) E:\Django_VirtualEnv > cd myenv

(myenv) E:\Django_VirtualEnv\myenv > cd Scripts

(myenv) E:\Django_VirtualEnv\myenv\Scripts > deactivate

E:\Django_VirtualEnv\myenv\Scripts >

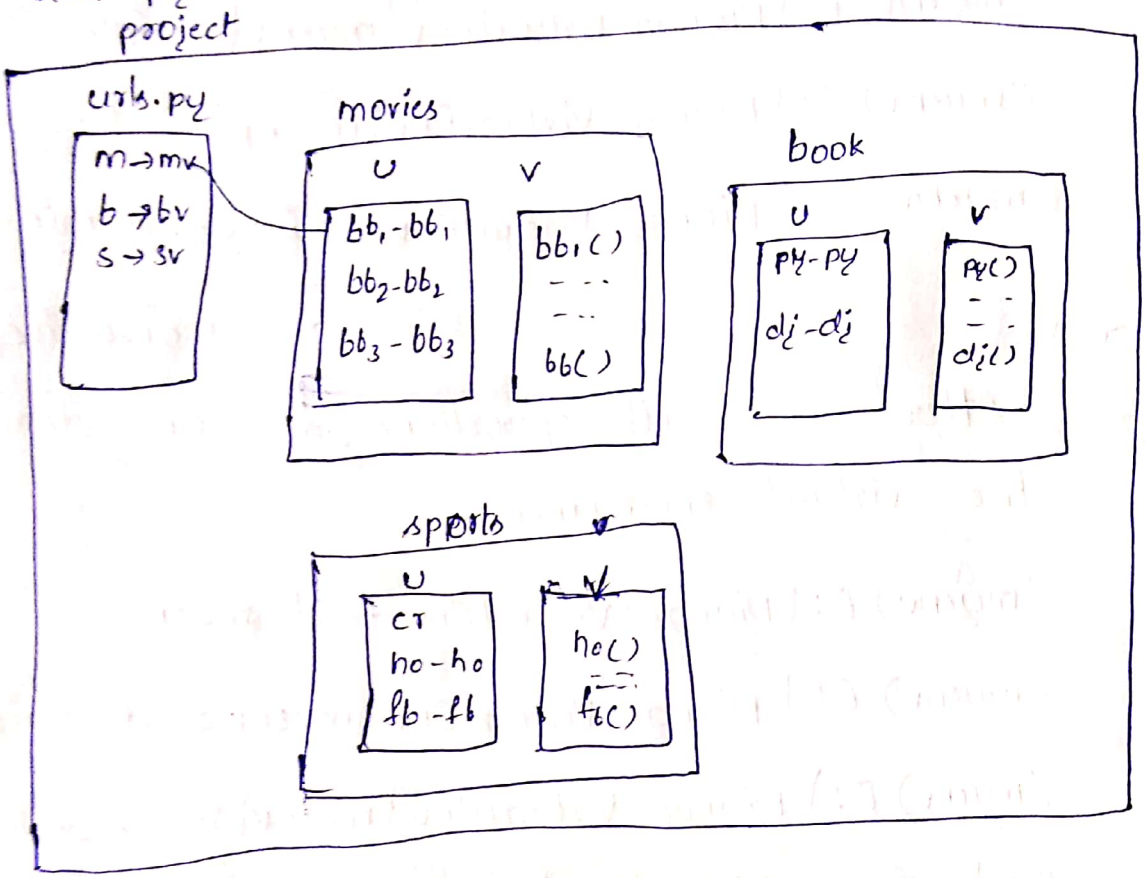
Creating urls in the application

* Generally urls.py file is the default file in the project
* We can also create urls.py files in the application
* If our project contains multiple applications and each application contains multiple views then

Scanned by CamScanner

we have to write no of url patterns in the project level urls.py file.

* It is more difficult to write more no of urls in single urls.py file in the project level.
* To avoid these problem we can create urls.py file in each application and we can configure the views of each application in its corresponding urls.py



Step1 : foldername : febdj4pm

Step2 : project Name : appurlspro

Step3 : app name : movies

Step4 : Go to terminal and create books and sports apps.

E:\febdj4pm\appurlspro > python manage.py startapp books

Scanned by CamScanner

E:\ febdjpm\appurlspro >python manage.py startapp sports

Step5: Go to moviesviews.py and write following views

```
from django http.response import HttpResponse

def bb1(request):
    bb1 = "It is a Nice Movie"
    return HttpResponse (bb1)
def bb2 (request):
    bb2 = "It is Animation Movie"
    return HttpResponse (bb2)
```

Step6: Go to booksviews.py file and write the following views

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

def python(request):
    p = "<h1>Python is more powerful and simple
                language</h1>"

    return HttpResponse(p)

def django (request):
    d = "<h1>Django is more powerful framework</h1>"
    return HttpResponse (d)
```

Step7: Go to sports/views.py file and write the following code

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

Scanned by CamScanner

```
def cricket(request):
    c = "It is a nice game"
    return HttpResponse(c)
def hocky(request):
    h= "It is a national game"
    return HttpResponse(h)
```

Step8: Create urls.py file in movies application

```
from movies import views
urlpatterns = [
        url(r'^bb1/', views.bb1),
        url(r'^bb2/', views.bb2)
]
```

Step9: create urls.py file in books application

```
from books import views
urlpatterns = [
        url(r'^python/', views.python),
        url(r'^dj/', views.dj)
]
```

Step10: Create urls.py in sports application

```
from sports import views
urlpatterns = [
        url(r'^cricket/', views.cricket),
        url(r'^hocky/', views.hocky)
]
```

Step11: Go to project level urls.py file and write

Scanned by CamScanner

the following code

```
from django.conf.urls import url
urlpatterns = [
    url (r'^admin/', include (admin.site.urls)),
    url (r'^movies/', include ('movies.urls)),
    url (r'^$, include ('books.urls')),
    url (r'^sports/', include ('sports.urls'))
]
```

Step12 Run the server and click on IP address it displays like below

127.0.0.1:8000/
python is morepowerful and simple language

Step13 :

127.0.0.1:8000/sports/cricket
It is a nice game

Scanned by CamScanner