General Request Flow:

Once user requests for something it will go to DNS (domain naming system). DNS will check for the corresponding Domain name or IP address. DNS will pass the request to web server.

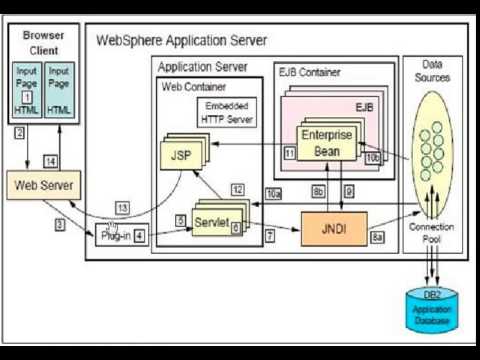
The web server will route to the app server from there it will go to JNDI from there it will go to the connection pool where objects will be allocated to the requests. Each request will get assigned one object. From there, objects will assist to take the data from the corresponding database.

Web Server: web server will return the response if the request is static content like HTML.

App Server: App server will consist 1. Web Container 2. EJB container.

1.Web Container: It consists of servlets and JSP pages.

2.EJBContainer: It consists of enterprise java beans



**Django**

Django is a frame work that is used to build apps. It uses to follow MVT model.

M stand for Model

V stands for View

T stand for Template.

**Django Advantages and Disadvantages:**

1. **Implemented in Python**
2. **Better CDN connectivity and Content Management**
3. **Batteries Included Framework**
4. **Fast Processing**
5. **Offers Rapid-development**
6. **Scalable**
7. **Security**

**Limitations:**

1. **It is not suitable smaller projects**

**Request flow of Django:**

1.Once user submits request it will go to Django project it will check URLS in the project folder if it is project level configuration.

2.If it is app level configuration of URLS it will go to respective apps URLS and from there it will go to Views and from there if it is database request it will go to Model.

3.Model is connected with DB.

4.Anything related static like HTML and CSS it will go templates and render the data in templates and then it will route to views and from there it will go to URL’s and URLS will send back the data to the respective user by using HTTP response.

**Request and Response:**

All backend technologies and web APIs are based on one system that is The Request-Response Cycle. The system is responsible for data interchange between client and servers. Even the new technologies based on micro-services use the Request/Response cycle.

Request / Response objects are transmitted over the web. Content like images, HTML, CSS, JavaScript is all Response Objects.

HTTP: Hyper Text Transfer Protocol is nothing but having set of rules and standards followed by devices to transfer information via internet.

Both the client and server are using HTTP. If both of them don’t have the same protocol, the connection cannot be done.

Just like humans need to have a common language to understand each other. The same way, various computers and machines over the internet need to have the same protocol. At least the one through which they communicate.

Response

Request

The requests are smaller in size than response. Basically, a Request is a “request made by the client” to the server. Any URL that gets searched is a request.

Below is the sample request.

<http://localhost:8000>

here http is protocol and localhost will act as domain name and 8000 will port number.

Requests are smaller compare to Responses as response consist HTML and CSS, JavaScipt , image, video files.

Usually, responses will be provided by the servers and the response consists different parts.

HTTP headers use to contain various information regarding the response.

We can consider that header info as Meta Data. Meta data use to tell the browser from where the request came and what type of data it is contains.

Django is an application which resides in server and its main task is to process the request received by the server and then it calls the functions and provide a response.

Generally, requests and response are handled by middleware’s in the Django.

When ever request comes it will call security middleware first if the middleware is unhealthy it won’t allow to go further.

If it allows the request to go further and later time Authentication middleware will come to the picture.

Authentication middleware won’t know how to handle the unhealthy request.

Once our request passes all the middleware’s it will go to URL router, URL router simply extracts url from the request and it will try to match with the request in the url’s, which we use to provide in URL.py file.

Once, we get a matching URL, the corresponding view function is called. The view function will get various attributes and other URL parameters. Also, the view function will be able to access files from the requests. These Requests are considered to be HttpRequest class objects.

Once, the view function has been executed, it’s time to give a response. The response is given in the form of HttpResponse. The response is not limited to that. The Response can be PDF, JSON, CSV. That is one of the features of Django. It provides built-in support to provide responses of various types.

If the response is render it will look for the HTML and then the HTML file will be processed by the Django Templating Engine and response will be sent, which consist HTML and other static content which is requested by the requester.

**Cookies:**

Every time when we request something it use to take as every request as new so that it causes issues like user login and authentication, these problems will be solved by cookies.

Cookies are small text files which are created and maintained by your browser on the particular request of Web-Server. They are stored locally by browser, and most browser will also show the cookies generated in the Privacy and Security settings of the browser.

Suppose, you are logging in any website, that website will respond the browser with some cookies which will have some unique identification of user generated by the server and some more details according to the context of the website.

Cookies made these implementations possible with ease which were previously not possible over HTTP implementation.

When initial request come’s the server use to send the response with some cookies along with requested content and from next time onwards when ever user makes new request along with the request browser will send cookies.

This process will be repeat until cookies expire or Session is closed. Post session closes cookies will auto delete by the browser itself.

Different websites are using different cookies depends on their requirements. In generally cookies get generated when ever we use to login any sites are online shopping apps like Amazon,flipcart.

In Django we can create the cookies by using below method.

**set\_cookie()**

Set method have attributes like name and values.

**name:** It specifies the name of cookie.

**value:** It specifies the text or variable you want to store in the cookie.

**max\_age:** It is the time period of cookie in seconds. After the mentioned time period over cookie will expire. It an optional parameter, if this parameter we didn’t gave that time cookie will be active until we close the browser.

There few drawbacks with cookies, Cookies are not using HTTPS protocol and cookies are plain text in nature, it is not safe to keep the important info which can be easily hacked by the hackers.

As cookies are storing in local browser users will get the prompt to store the data or not. Many websites are sending cookies with the accept or decline prompt.

Sessions:

To over come drawbacks of cookies the web developers come up with solution called sessions.

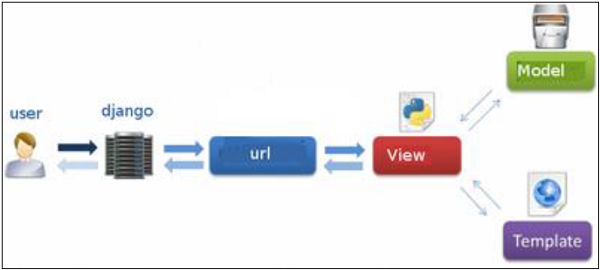
Which more secure.

Session uses two-way communication between browser and the server. When ever the client request something session will gets created with session ID which is unique and it will not change until session gets finish.

In Django we use to middleware and inbuilt app which will help you generate these session IDs.

Caching:

Caching is the process of storing the data which is recently generated, so that we can use that data in near future when requested again. We can remove cache when ever we want.



**Architecture of Django**

**Heigh level steps we are performing in Django:**

**1.Create project**

**2.Create App**

**3.Start App**

**4.Create template folder**

**5.Create subfolder inside template folder**

**6.Create HTML file**

**7.Create a database**

**8.Write a class in model which fields you want in database table.**

**9.Edit HTML file with required fields of database.**

**10.Import model in views by model class name.**

**11.Create static folder inside create another folder called image and then place images required for website here.**

**12.Edit settings file with required parameters like Database and Static and Templates Installed Apps.**

Commands to create Django project:

>mkdir folder name

Go inside the folder

>Django-admin startproject projectname

Start the Django Project

>python manage.py run server

Create App:

>python manage.py startapp appname

1.init file 🡪by seeing this we can understand the folder is Django project.

2.Settings file🡪 It contains config settings to connect DB and Middleware

1.Apps details

2.Middleware details

3.Templates

4.WSGI APP config

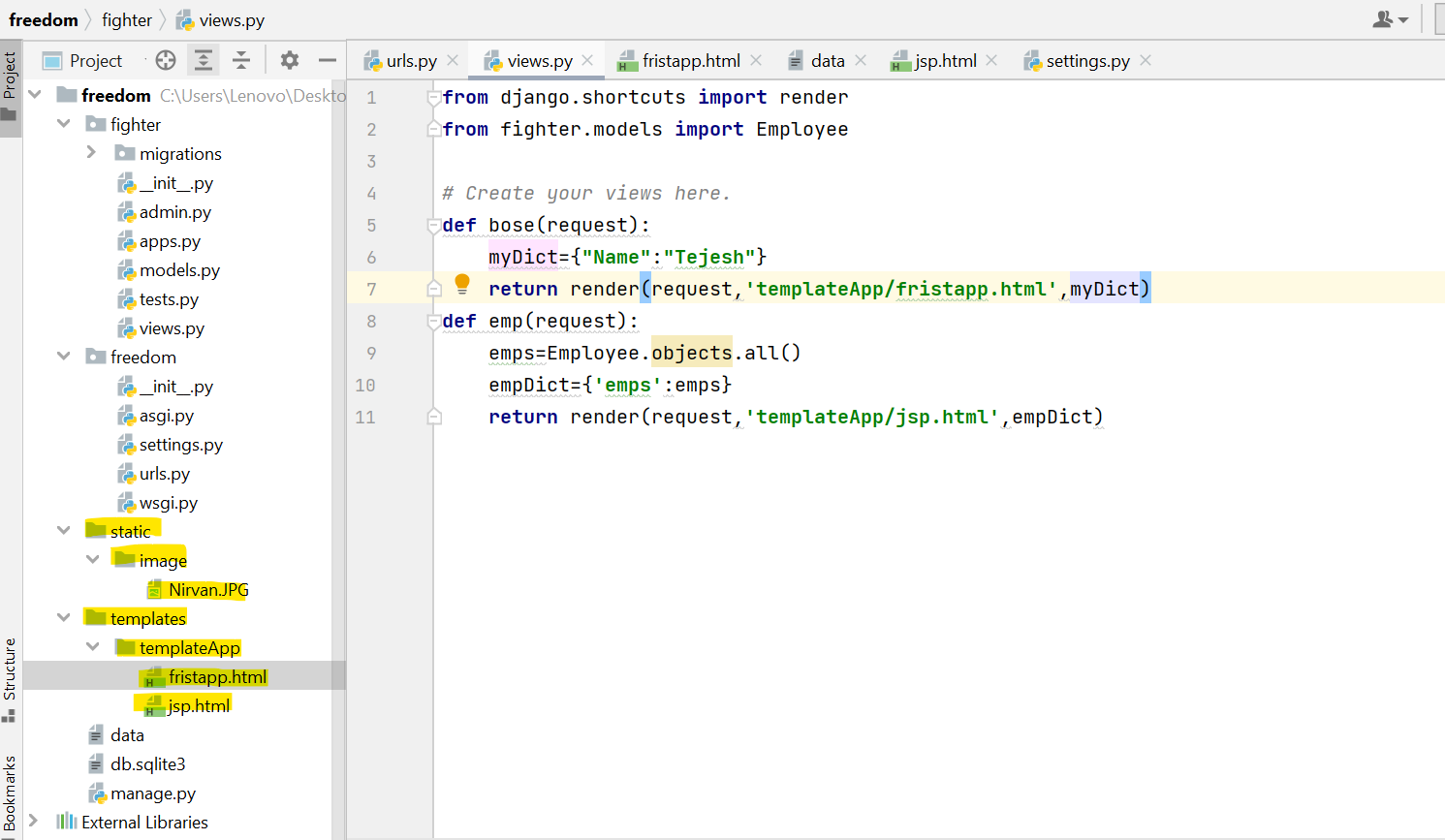
5.Data bases

3.urls🡪 It carries url pattern. Web pages or views urls we will configure here.

4.wsgi🡪Web Server gateway interface 🡪it is used to deploy our applications to online servers or cloud servers (We use this file on prod deployments)

5.manage.py🡪it is useful while doing migrations

Default Django port number is 8000



**Configurations in Settings file:** from pathlib import Path

import os

import operating system is used to communicate from python to System.

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

In the above field BASE\_DIR will take the absolute path of the project.

DEBUG = True

We should not use this option in production as

INSTALLED\_APPS = [

"django.contrib.admin",

"django.contrib.auth",

"django.contrib.contenttypes",

"django.contrib.sessions",

"django.contrib.messages",

"django.contrib.staticfiles",

"app1",

]

We have to add created app-name in the settings->Installed Apps.

In the above example we have added “app1” which we have created earlier.

MIDDLEWARE:

It is a frame or a plugin that takes an incoming request and manipulate (make some required changes) and send it to the view.

It can also work on templates like once the view process everything and middleware can manipulate and process and send it back to views.

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

**Security Middleware(django.middleware.security.SecurityMiddleware)** :

It is responsible for HTTPS or SSL support and other security features.

**Session Middleware (django.contrib.sessions.middleware.SessionMiddleware)** :

It helps us to manage the sessions, it is responsible for taking the incoming request and make sure if session active and response goes to web browser that time cookies will be created. **Common Middleware(django.middleware.common.CommonMiddleware):**

It works on URL’s; it checks any forward slash at the end of the URLS if any issues with that it will resolves.

**CSRF Middleware(django.middleware.csrf.CsrfViewMiddleware):**

If the form is submitted it checks for CSRF token if that is not there it will throws an error.

**Authentication Middleware(django.contrib.auth.middleware.AuthenticationMiddleware) :**

Authentication middleware is responsible for authenticating the end user request. It will also make sure there is object available for the views and for the template once the authentication happens.

Above are the default middleware’s which are provided by Django. We won’t change these existing middleware’s. We will create our own custom middleware’s to interrupt the incoming request and do some changes depends on the requirements like encrypt or zip.

How to create custom middleware’s:

1. **\_\_init\_\_** , **\_\_call\_**\_ are mandatory inbuilt methods in every middleware which we are creating.
2. **process\_view** , **process\_exception** and **process\_template\_response** methods are inbuilt optional while creating our own custom middleware’s. In below example we are explaining how we can declare these methods in a class.

Example:

MyMiddleware

\_\_init\_\_(self,get\_response)

\_\_call\_\_(self,request)

Process\_view(self,request,view\_func,view\_args,view\_kwargs)

Process\_exception(self,request,exception)

Process\_template\_response(self,request,response)

Once view sends the request,

First it will call \_\_init\_\_ method and it will go to” Security Middleware”, Once it is done it will point to next middleware “Session Middleware”. Once Session Middleware done it will point to Common Middleware,

After Common Middleware it will points to CSRF Middleware finally it will go to Authentication Middleware.

Once all above middleware’s gets complete Django will hand over request to view itself.

­­­­­­\_\_call\_\_ method is invoked before it calling next method or after the response processed and just before response goes back to the client.

Process\_view: This Method has direct access to the view function and its arguments can change and manipulate if required usually we won’t do that. Within this method we will add extra logic like unzip, zip, Encrypt, Decrypt

Process\_Exception: We can handle the exceptions that are raised with in our application. It has access to the request and exception that has been raised.

Process\_template\_response: This method will be invoked just before the template response is sent to the client it has access to request and response.

Creation of Middlleware:

create a file with middleware.py and write a class with required methods like \_\_init\_\_.

Class Middlewarelife:

def \_\_init\_\_(self,get\_response):

self.get\_response=get\_response

def \_\_cell\_\_(self,request):

print(‘Before the view is executed’)

response=self.get\_response(request)

print(‘after the view is executed’)

return response

Custom middleware which we have created will be configured at the end of the middleware section in the security.py file. After this middle if there any other middleware, the request will go to that (get response points to the next middleware). if the created customized middleware is the last middleware in the settings file, it will go to the view itself.

Exception handling exception:

TEMPLATES = [

{

"BACKEND": "django.template.backends.django.DjangoTemplates",

"DIRS": [os.path.join[Base-Dir,’templates’]],

"APP\_DIRS": True,

"OPTIONS": {

"context\_processors": [

"django.template.context\_processors.debug",

"django.template.context\_processors.request",

"django.contrib.auth.context\_processors.auth",

"django.contrib.messages.context\_processors.messages",

],

},

},

]

DATABASES = {

"default": {

"ENGINE": "django.db.backends.sqlite3",

"NAME": BASE\_DIR / "db.sqlite3",

}

}

If you want to use default database as your database then we no need to perform any modifications. If not, we can specify our own database and we can mention the DB name as below.

Ex: ‘ENGINE’: ‘django.db.backends.Mysql’,

‘NAME’: ‘tejesh’,

‘USERNAME’: ’root’,

'PASSWORD':'Admin@123'

Static:

STATIC\_URL = 'static/'

STATICFILES\_DIRS=[os.path.join(BASE\_DIR,'static')]

Here we use to provide Base directory path and static as parameter to identify where the static file exists.

**CRUD Operations: An admin user can perform below operations by logging in to**

**C-create: The ability of the application to store data in the database.**

**R-read: The ability of the application to read data from the database.**

**U-update:** **The ability of the application to edit the stored value in the database.**

**D-delete: The ability of the application to delete the value in the database.**

**We are going to develop the operations in the same order.**

**Majority of applications on the internet are CRUD applications**

CRUD operations on function-based views.

Creating new project to perform CRUD operations.

1.creation of virtual environment.

2.Install Django

3.Create the Django project

4. Installing Application and Other Important Settings

5.Create Models

6.Create Views

7.CRUD operations on tables using ORMS

Example program:

Settings.py:

from pathlib import Path

import os

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-=m-32%\_g@agdwr=ji2u!!ls+90d#vdg@tcur7i0#++j^zvf6%c'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'fighter'

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'freedom.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR,'templates')],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'freedom.wsgi.application'

# Database

# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'empdb',

'USER':'root',

'PASSWORD':'Admin@123'

}

}

# Password validation

# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/4.1/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/4.1/howto/static-files/

STATIC\_URL = 'static/'

STATICFILES\_DIRS=[os.path.join(BASE\_DIR,'static')]

# Default primary key field type

# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

Views:

from django.shortcuts import render

from fighter.models import Employee

# Create your views here.

def bose(request):

myDict={"Name":"Tejesh"}

return render(request,'templateApp/fristapp.html',myDict)

def emp(request):

emps=Employee.objects.all()

empDict={'emps':emps}

return render(request,'templateApp/jsp.html',empDict)

def createEmp(request):

form = EmployeeForm()

if request.method =='POST':

form = EmployeeForm(request.POST)

if form.is\_valid():

form.save()

return redirect('/')

return render(request,'templateApp/create.html',{'form':form})

def deleteEmp(request,id):

emp=Employee.objects.get(id=id)

emp.delete()

return redirect('/')

HTML file1:jsp.html

HTML in Templates

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Employee</title>

</head>

<body>

<h1>Emp List:</h1>

{% if emps %}

<table>

<thead>

<th>firstName</th>

<th>lastName</th>

<th>sal</th>

<th>email</th>

</thead>

{% for emp in emps %}

<tr>

<td>{{emp.firstName}}</td>

<td>{{emp.lastName}}</td>

<td>{{emp.sal}}</td>

<td>{{emp.email}}</td>

</tr>

{% endfor %}

</table>

{%else%}

<p>No records found</p>

{% endif %}

</body>

</html>

HTML file 2: create.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Create</title>

</head>

<body>

<h1>Create Employee</h1>

<form method="post">

<table>

{{form.as\_table}}

{%csrf\_token%}

</table>

<br/>

<input type="submit" name="" value="save">

</form>

</body>

</html>

Form:

from django import forms

from fighter.models import Employee

class EmployeeForm(forms.ModelForm):

class Meta:

model=Employee

fields='\_\_all\_\_'

Admin.py

from django.contrib import admin

from fighter.models import Employee

# Register your models here.

class EmployeeAdmin(admin.ModelAdmin):

list\_display = ['firstName','lastName','sal']

admin.site.register(Employee,EmployeeAdmin)

URLS:

from django.contrib import admin

from django.urls import path

from fighter import views

urlpatterns = [

path('admin/', admin.site.urls),

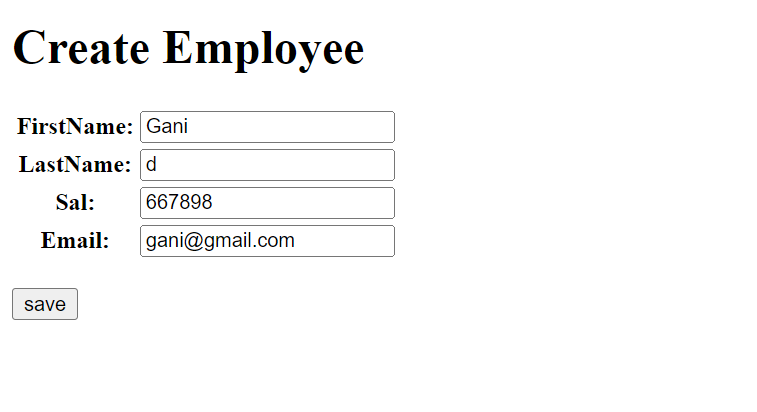
path('p/', views.bose),

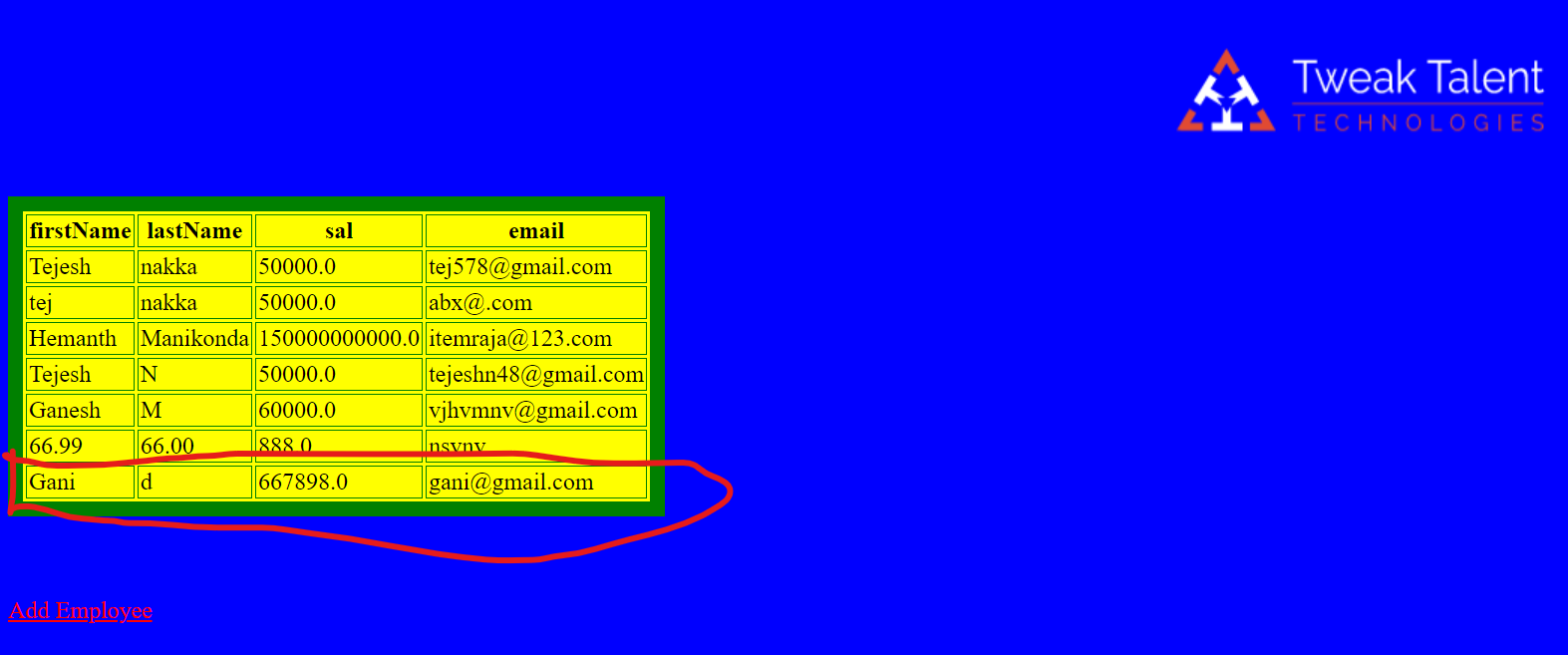
path('q/',views.emp)

]









1.In the above example we have created a database called empdb, and the required fields of database like firstname and lastname, sal, email , we have placed in the class of models.py file.

2.We have written a function in views to get all objects in the model and render with empdb.

3.In html file called jsp we have created under Templates\templateApp\jsp.html. In jsp.html we have created table and mentioned database fields names.

4.After above steps we have done make migrations and performed all migrate steps for the fields we have mentioned in models.

5.Post this fields will create in table.

6.For insertion of the data we have created form and to redirect to the form from the actual tables page we have given hyper link by using anchor tag in html.

7.To create a form we have created an empty file in the form file we have imported forms from Django.

8. Import models of the app with class name.

9.Create class and take the meta data from the model with all fields.

10.Provide context root and corresponding view name (along with function name).

Delete:

We have added delete in existing HTML like below.

<img src="{%static "image/Nirvan.JPG" width="104" height="142"%}"/>

After HTML now its time to create views for the delete operation.

**ORM:**

Django lets us interact with its database models, i.e. add, delete, modify and query objects, using a database-abstraction API called ORM (Object Relational Mapper). This article discusses all the useful operations we can perform using Django ORM.

For demonstration purposes, we will use the following Django models.

Example1:

from django.db import models

# Create your models here.

class Employee(models.Model):

firstName=models.CharField(max\_length=30)

lastName=models.CharField(max\_length=30)

sal=models.FloatField()

email=models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.title

Example2:

from django.contrib.auth.models import User

from django.db import models

payment\_choices = (

('UPI', 'UPI'),

('CC', 'Credit Card'),

('DC', 'Debit Card'),

('COD', 'Cash On Delivery'),

('APB', 'Amazon Pay Balance'),

)

class Product(models.Model):

name = models.CharField(max\_length=15)

price = models.FloatField()

description = models.CharField(max\_length=100)

variants = models.CharField(max\_length=5)

items\_available = models.IntegerField()

def \_\_str\_\_(self):

return self.name

class Cart(models.Model):

user\_id = models.ForeignKey(User, on\_delete=models.CASCADE)

product\_id = models.ForeignKey(Product, on\_delete=models.CASCADE)

class Order(models.Model):

user\_id = models.ForeignKey(User, on\_delete=models.CASCADE)

billing\_address = models.CharField(max\_length=50)

delivery\_address = models.CharField(max\_length=50)

payment\_type = models.CharField(max\_length=10, choices=payment\_choices)

class OrderProduct(models.Model):

order\_id = models.ForeignKey(Order, on\_delete=models.CASCADE)

product\_id = models.ForeignKey(Product, on\_delete=models.CASCADE)

no\_of\_products = models.IntegerField(default=1)

class Wishlist(models.Model):

user\_id = models.ForeignKey(User, on\_delete=models.CASCADE)

product\_id = models.ForeignKey(Product, on\_delete=models.CASCADE)

We can access the Django ORM by running the following command inside our project directory.

python manage.py shell

This brings us to an interactive Python console.