Django: Free and Open source Web application framework

Sunday, July 31, 2022 8:53 AM

Framework:
A combination of certain components and packages.
When we want to build something complex, we need something readymade. The framework will consists of this readymade things that can be used to achieve those tasks.

For a web application to work, we need HTML, CSS and JS for the frontend to make the website to have a structure, to maintain design and to make that interactive.
For the backend part where the data is stored and processed, we need a language to work on that.
Examples like Servlet, PHP, JavaScript, ASP, Python

Django is a python based tool that helps to build the backend part using Python.

MVC: Model View Controller

No matter what is the tool we are using, this is must to know
This is done in order to separate the concerns.

Model will talk about the data.
View decides how we see the data.
Controller will control this whole operation.

In Django we have MVT(Model View Template)
Model(M) --> Model(M)
View(V) --> Template(T)
Controller(C) --> View(V)

More about this to be added....

Installing Django
- Python latest version(3.7+), pip should be installed by default.
- Virtual environment(Optional, but useful), Installation in Linux will be different
  - pip install virtualenvwrapper-win
  - mkvirtualenv myvenv

- Django
  - pip install django
  - django-admin --version #To verify installation

Getting started with Django

To start a project
django-admin startproject myproject
In a directory myproject, we'll have some files which are designed for a specific purpose,
Whatever we need to do, we'll do it with the help of a manage.py file.
Example is to run the server: python manage.py runserver

First app in Django

As seen in the installation steps, we'll have a server running that does NOTHING.
Now we'll see how we can create something useful...

python manage.py startapp someapp

Now there will be a folder inside the myproject named someapp which also contains its own admin, models, apps, tests, views, urls(we need to create) files. Each has its own working.

URLs will be used to direct the request coming
Views will be sed to control/process the data before sending to template(HTMLpage with data that is displayed to us)
Models file will be used to serve the data, ORM(Object Relational Mapping) will be used here(More will be updated soon)
Tests file will be used to test the application

Main urls.py

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

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

App urls.py

```
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      path('',views.home, name='home')
7
8  ]
```

This will direct the requests coming to / or " "(home) to home method in views module(file).
Home method will handle the request , get the data, process the data and send it to user via HTML
We'll use render method to mix the data with the HTML so that it can be displayed to user as a single unit.

Example with HTTPResponse

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def home(request):
    return HttpResponse("Hello World")
```

Example with render method:

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3  # Create your views here.
4
5  def home(request):
6      return render(request, 'home.html',{'name':'Navin'})
```

Dictionary

Refer the Template and Theme section for more info

File structure

seetings.py

- BASE_DIR
  - Stores the base directory of the project
- SECRET_KEY
  - Used for deployment, should be removed while pushing to production
- DEBUG
  - A Boolean for debug
- Allowed hosts
  - By default, it is available only via localhost, if we want to make it available for any machine, this need to be updated.
  - We can give list of hosts, or we cany say everything so that any host can reach the server.
- INSTALLED_APPS
  - Installed apps in the project
- MIDDLEWARE
  - For security related concerns
- TEMPLATES
  - Will store the templates directory and its related information
- DATABASES
  - By default, Django will provide SQL lite Database, we can use whatever we want

Urls.py

We'll have multiple urls.py, each application will have its own urls that define how to route among the page.
There will be one common url file that's where the request hits first and redirected accordingly

By default, django will give one url for admin stuff, later we'll add based on the apps we create

Templates

Templates is where we'll have all static data like the bg color, preconfigured texts and etc.
And some place for Dynamic content.
Once everything is in place, we can send the data to that html using render method and the dynamic data will be replaced and will be see at the end.

DTL (Django Template Language)
Language that is used in the templates
Kind of programming language with variables, loops, conditional statements, etc.
Variables: {{ name }}
Loops: {{ for element in list }} .... {{ endloop }}
Conditional statements: {{ if condition }} .... {{ endif }}

Create the HTML page and keep it in a directory(Generally in the root directory of the project)
Once the HTML page is created, we need to update the TEMPLATES in settings.py file by providing the path of the directory with all HTML pages

To this template, we can send data from views as a dictionary and refer here.

Themes

Generally pages will have a standard theme and will be used entirely across the project.
If that theme has a lot of CSS built in and requires many lines in every HTML page, putting them in every page will be difficult.
Instead of that, we can define everything at one place and refer that in the HTML so that all reusable thing are kept at a place and code looks clean.
Since everything is at once place, if we want to make some changes to the theme, we can do it in single place and it'll be changed across the project since the theme is shared among all.

Static/Standard part of all HTML

```
1   <!DOCTYPE html>
2   <html lang="en">
3
4   <head>
5       <meta charset="UTF-8">
6       <meta name="viewport" content="width=device-width, initial-scale=1.0">
7       <meta http-equiv="X-UA-Compatible" content="ie=edge">
8       <title>Telusko</title>
9   </head>
10
11  <body bgcolor="cyan">
12
13      {% block content %}
14
15      {% endblock %}
16
17
18  </body>
```

Here inside the block, all the HTML pages will come and sit.
For example,

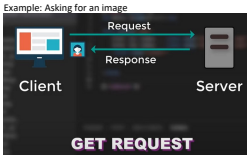
```
1  {% extends 'base.html' %}
2
3  {% block content %}
4  <h1>Hello {{name}}!!!!!</h1>
5
6  {% endblock %}
```

This says include this block content in the block defined in base.html.

GET and POST

GET and POST are HTTP request methods

GET is basically when we are requesting the server any data.
Example: Asking for an image



POST is basically when we are sending any data to Server

```
5   def home(request):
6       return render(request, 'home.html',{'name':'Navin'})
```

Refer the Template and Theme section for more info

Addition of two number in Django

Addition of two number in Django

```
<form action="add">
    Enter 1st number : <input type="text" name="num1"><br>
    Enter 2nd number : <input type="text" name="num2"><br>
    <input type="submit">


</form>
```

When this from is submitted, it will route to the name specified in the action field.
We need to map it in the urls.py of the app and define a method in views.py to process the request

In urls.py of the app

```
urlpatterns = [
    path('',views.home, name='home'),
    path('add',views.add, name='add')
]
```

In viwes.py of the app

```
def add(request):
    return render(request, "result.html")
```

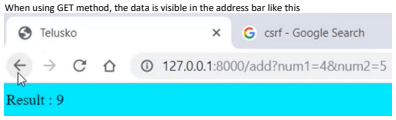
Above one is just an example one, will need some more stuff to get the data and parse and process the data.

Using GET method(Unsecure way of doing)

```
def add(request):

    val1 = int(request.GET["num1"])
    val2 = int(request.GET["num2"])
    res = val1 + val2

    return render(request, "result.html", {"result": res})
```

This is with GET method, but in practice we need to use POST method.
Refer GET and POST section for more info.

When using GET method, the data is visible in the address bar like this



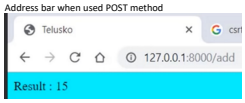
Data like this can be intercepted easily and for critical information like username, password, forms and all the data should not be seen like this. This can be achieved by using POST method.

The type of method can be specified in the form itself as a method attribute, by default it is GET.

```
<form action="add" method='post'>

    {% csrf_token %}

    Enter 1st number : <input type="text" name="num1"><br>
    Enter 2nd number : <input type="text" name="num2"><br>
    <input type="submit">


</form>
```

Refer CSRF for more info....

Address bar when used POST method



Modify the views accordingly to get data from POST instead of GET

```
def add(request):

    val1 = int(request.POST["num1"])
    val2 = int(request.POST["num2"])
    res = val1 + val2

    return render(request, "result.html",
```

Static files

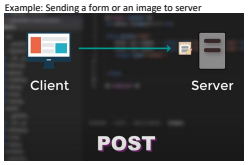
Static files refer to images, CSS and JS files that are needed for the website to work.

```
STATIC_URL = '/static/' #This is how we'll refer them
#List of directories that contains static file
STATICFILES_DIRS  = [
        os.path.join(BASE_DIR,'static'),
]
#Directory for django keep all the static files
STATIC_ROOT = os.path.join(BASE_DIR,'assets')
```

After placing all in static directory, we need to inform the Django to collect the files and keep it in its folder.

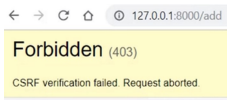
python manage.py collectstatic
This command will create the assets directory and keeps all the static files in its directory

POST is basically when we are sending any data to Server
Example: Sending a form or an image to server



CSRF Token

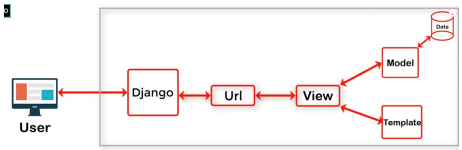
In the internet, there's a lot of user data.
That data might be at risk by hacker with multiple attacking techniques.
More info: https://portswigger.net/web-security/csrf



To avoid this, by default django gives a Middleware(see setting.py)

MVT: Model View Template

Model will be linked to database to fetch the data or update the data
Template is what the page design(static) with some dynamic data to update before displaying(Usually with Django Template Language(DTL))
Views will process the data and send the response to user with all his requested data along with the static content.



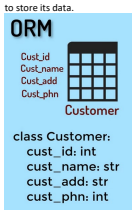
ORM: Object Relational Mapping/Mapper

In short, by looking at the class definition, it will create the columns in the database

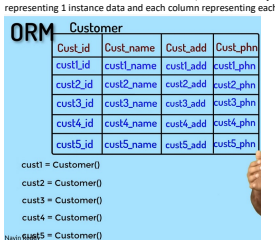
In general, data will be stored in the database, and if we want to access the data via an application written in any language with OOPs, then we'll have a class.

A database will have columns and a class will have properties.

In this example, a class has 4 properties and therefore the table should have 4 columns.
Any number of instances can be created for a same class. Similarly for each instance, a row will be created in the database to store its data.

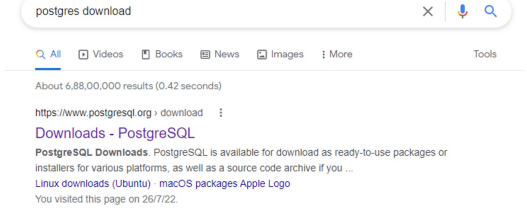


If we create 5 instances of the customer class, then those 5 objects data will be stored in the database as 5 rows. Each row representing 1 instance data and each column representing each property value.



Postgres and PgAdmin setup

Postgres will be the database we'll be working with and PgAdmin is the interactive GUI to work with the Postgres database.



```
STATIC_ROOT = os.path.join(BASE_DIR,'assets')
```

After placing all in static directory, we need to inform the Django to collect the files and keep it in its folder.

python manage.py collectstatic
This command will create the assets directory and keeps all the static files in its directory

Using them in the templates(using DTL)

First we need to load static
{% load static%}
Then we need to add {% static 'previous/path/to/file' %} wherever the path is there....

Passing Dynamic data Static files

Previously we just took the existing template and made some changes, created assets folder with collectstatic command.
Now we'll try to send the dynamic data to that.

Since we are working with Python, we can fetch the data from anywhere.
Once the data is in the Python, we can pass that to HTML and use it in the DTL format.

Example:
Passing a sample data from views.py as **{'price':700}** via render method to a HTML page,
In the HTML page, we can use it as **{{ price }}**
Now that can be a simple dictionary with one key, value pair or can be multiple key, value pairs or even it can be dictionary of dictionaries also.

For this definition purpose, we'll use the models.py file to define a class with required fields and use that class and create objects in the views.py

Models.py

```python
# Create your models here.
class Destination():

    id: int
    name: str
    description: str
    image: str
    price: int
```

Once after defining models, we'll now create the data in views(later it'll be fetched from database, for now let's keep it simple)

Views.py

```python
# Create your views here.
def index(request):


    dest1 = Destination()

    dest1.id = 1
    dest1.description = 'The Spiritual Capital'
    dest1.name = 'Tirupati'
    dest1.price = 500
    dest1.image = 'tirupati.webp'


    dest2 = Destination()

    dest2.id = 2
    dest2.description = 'The city never sleeps'
    dest2.name = 'Mumbai'
    dest2.price = 500
    dest2.image = 'mumbai.webp'

    dest3 = Destination()

    dest3.id = 3
    dest3.description = 'Biryani city'
    dest3.name = 'Hyderabad'
    dest3.price = 600
    dest3.image = 'hyderabad.jpg'

    dests = [dest1, dest2 , dest3]

    return render(request,'index.html',{'dests':dests})
```

Accessing that data in index.html
Since we are passing a list, we can iterate through that and access the data using Jinja format

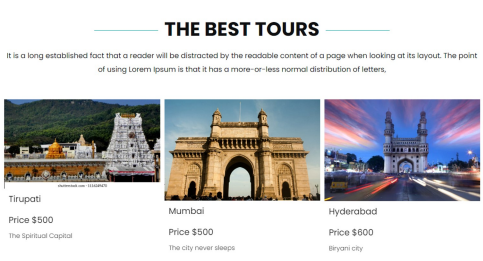
```
{% for dest in dests %}
<div class="item">
    <img class="img-responsive" src="{{baseUrl}}/{{dest.image}}" alt="#" />
    <h3>{{dest.name}}</h3>
    <h3>Price ${{dest.price}} </h3>
    <p>{{dest.description}}</p>
</div>
{% endfor %}
```

Additional setup...as jinja inside jinja won't work at images part

index.html | urls.py | urls.py | setti

```
{% load static %}
{% static 'images' as baseUrl %}
```

Final output....

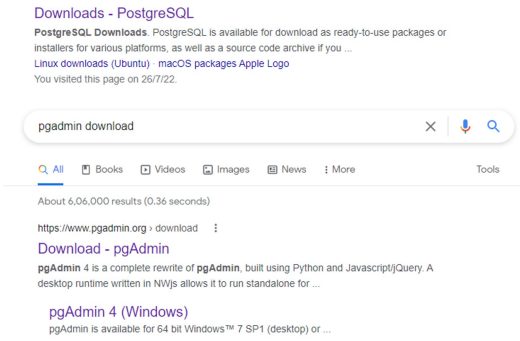


Passing data from a Database

Before this. Refer to ORM, Database setup sections.

Models and Migrations

Modification in the seetings.py to say that the DB we are using is Postgres and provide the required details to connect to the DB



Download and install these two software.

While installing Postgres, it will ask for the password, port number that are needed to access the database, provide and continue.
- rootpass: 1234
- user: postgres
- Port: 5432

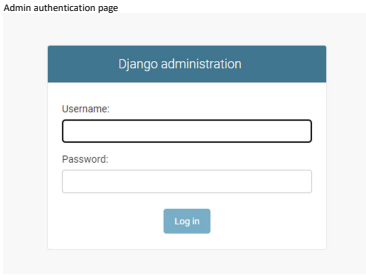
Admin panel

Admin panel is used to manage users, create data and to do lot more stuff.
Django will provide a admin page for us. We just need to create a username and password for it.

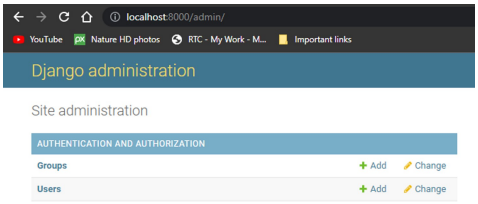
```
PS C:\Users\rtejac\django_1\demo> python manage.py createsuperuser
Username (leave blank to use 'rtejac'): teja
Email address:
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
PS C:\Users\rtejac\django_1\demo>
```

Username: teja
Password: teja

Admin authentication page

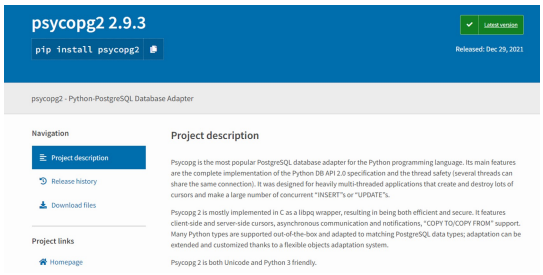


Admin page after login



```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydb_django',
        'USER': 'postgres',
        'PASSWORD': '1234',
        'HOST': 'localhost'
    }
}
```

Along this we need a connector in between Python and DB as they are basically two different softwares.
Package: **psycopg2**



Now that we have DB and its setup ready, now in order for django to create Table(Refer ORM), we need to migrate the model.
For that we need to define the model saying the parameters the DB will be having, etc.

Previously we have defined the parameters like int, str, etc. Now since we are creating model, we need to make that class a model and define the fields it's going to have.
Refer this for more info on fields: https://docs.djangoproject.com/en/4.0/ref/models/fields/

First add the app to INSTALLED_APPS list in settings.py

```python
INSTALLED_APPS = [
    'myapp.apps.MyappConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Then do the changes in the models.py

```python
# Create your models here.
class Destination(models.Model):

    # id feild will be given automatically by the DB
    name = models.CharField(max_length=100)
    description = models.TextField()
    image = models.ImageField(upload_to='pics')
    price = models.IntegerField
```

We need to have the **pillow python package** since we are working with images

Once done adding models and modifying settings.py, we need to migrate the model.
Python manage.py makemigrations

```
PS C:\Users\rtejac\django_1\demo> python .\manage.py makemigrations
Migrations for 'myapp':
  myapp\migrations\0001_initial.py
    - Create model Destination
PS C:\Users\rtejac\django_1\demo>
```

After this we'll have a migration file in the migrations directory of this app

```
myapp
  __pycache__
  migrations
    __pycache__
      __init__.py
      0001_initial.py
```

Run this command to check the SQL command for the model we created.

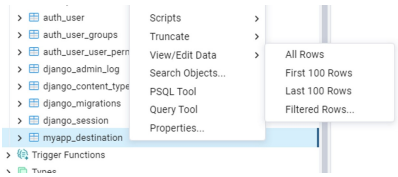
```
PS C:\Users\rtejac\django_1\demo> python .\manage.py makemigrations
Migrations for 'myapp':
  myapp\migrations\0001_initial.py
    - Create model Destination
PS C:\Users\rtejac\django_1\demo> python manage.py sqlmigrate myapp 0001
BEGIN;
--
-- Create model Destination
--
CREATE TABLE "myapp_destination" ("id" bigserial NOT NULL PRIMARY KEY, "name" varchar(100) NOT NULL, "description" text NOT NULL, "image" varchar(100) NOT NULL);
COMMIT;
PS C:\Users\rtejac\django_1\demo>
```

Once that is done, run migrate command to migrate the model

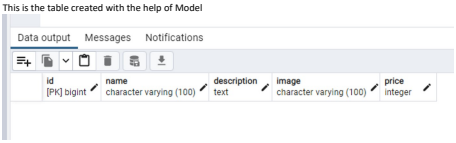
```
PS C:\Users\rtejac\django_1\demo> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying myapp.0001_initial... OK
  Applying sessions.0001_initial... OK
PS C:\Users\rtejac\django_1\demo>
```

Verify this by going to the PgAdmin, we need to see a table with

Select DBNAME --> Schema --> Tables --> Table Name --> View --> All



This is the table created with the help of Model



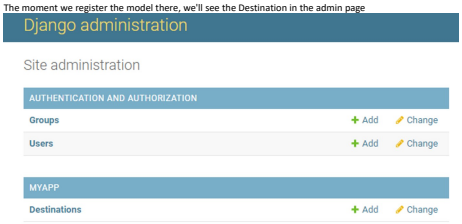
Now that we have a table in DB, we can add data to it and that can be seen in the web app.
Before that we need to some changes.
Refer to Admin panel and come back

Once after admin page setup,

We need to register our model in the admin.py of app(myapp)

```python
from django.contrib import admin
from .models import Destination

# Register your models here.

admin.site.register(Destination)
```

The moment we register the model there, we'll see the Destination in the admin page



Add and fetch the data from database

Before that....Currently images are coming from the static folder. It should not be like that. It should go into a particular folder.

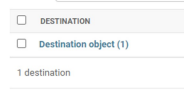
In settings.py

```python
#For Media files
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR,'media')
```

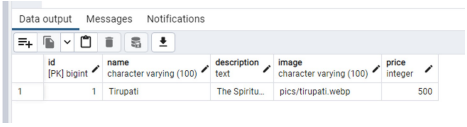
And also we need to modify urls.py of project

```python
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static


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

urlpatterns = urlpatterns + static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

When we add data to Destinations in admin panel,

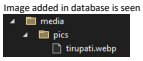


We can observe it in the database also



This make sure that the data we gave in admin panel(Python) is stored in DB(postgres)

Image added in database is seen in the pics directory in project




```
{% for dest in dests %}
<div class="item">
    <img class="img-responsive" src="{{dest.image.url}}" alt="#" />
    <h3>{{dest.name}}</h3>
    <h3>Price ${{dest.price}} </h3>
    <p>{{dest.description}}</p>
</div>
{% endfor %}
```

After adding 2 more cities in the database
Website:

THE BEST TOURS

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters,

Tirupati

Price $500

The Spiritual Capital

Bangalore

Price $700

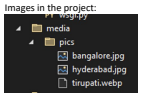
Silicon valley of India

Hyderabad

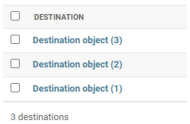
Price $600

The City of Biryani

Images in the project:



Snapshot of admin panel:



Snapshot of database:

	id [PK] bigint	name character varying (100)	description text	image character varying (100)	price integer
1	1	Tirupati	The Spiritu...	pics/tirupati.webp	500
2	2	Bangalore	Silicon vall...	pics/bangalore.jpg	700
3	3	Hyderabad	The City of ...	pics/hyderabad.jpg	600

p