1. **Django History and Introduction**

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django was developed for a fast pace development environment for a news web application by *Adrian Holovaty* and *Simon Willison*. In 2003, They stopped using PHP for their development and started using python. Then they made continuous improvements to their framework and finally it was made open source in 2005.

Django is named after [*Django Reinhardt*](https://en.wikipedia.org/wiki/Django_Reinhardt)*,* a jazz guitarist.

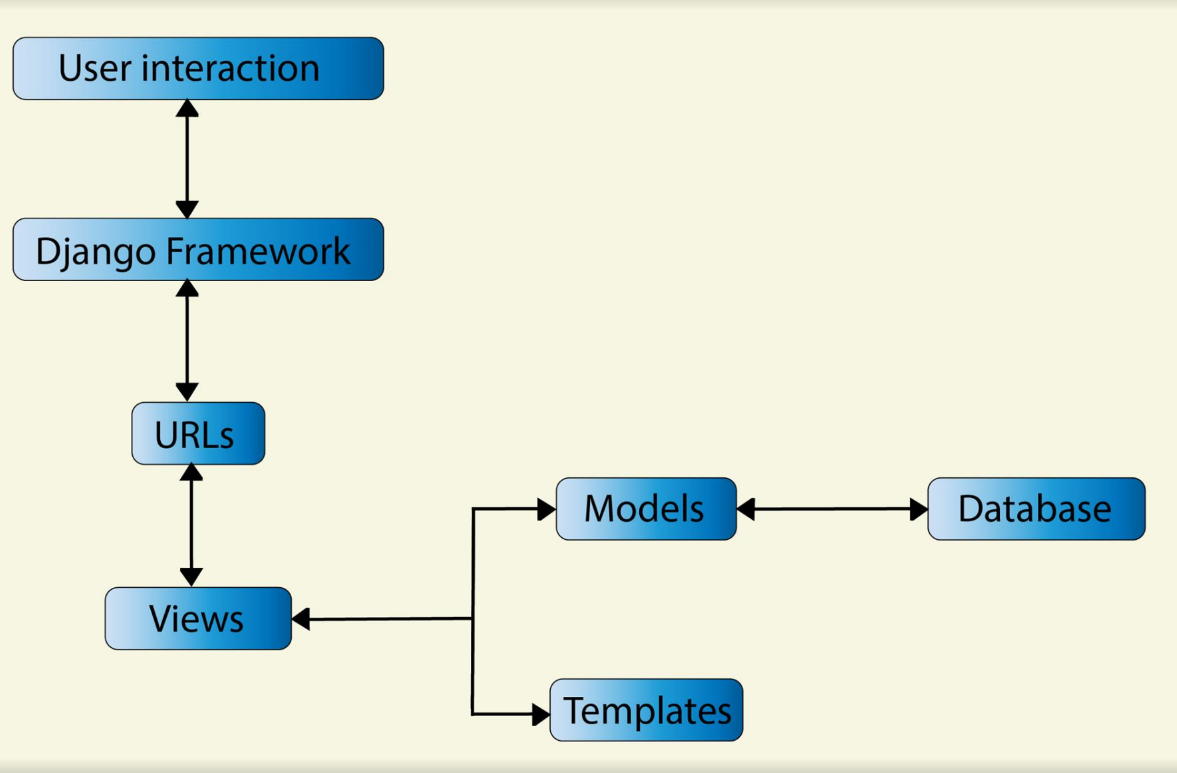
* 1. **Django Features:**

1. Extremely fast.
2. Extremely secure
3. Extremely scalable
   1. **Django Design philosophies:**
4. **Loose coupling:** Loose coupling means that the different modules/layers are more or less independent of changes in each other.
5. **Less Code:** The developer should have to write lesser number of lines of code. Most of the common things related to the framework will be generated by Django itself so that the developers can only focus on writing the business logic.
6. **Quick Development:** As the number of lines to be written is reduced, and the framework itself takes care of the dependencies, the development/ enhancement time gets drastically reduced.
7. **Explicit is better than implicit:** This feature gives the developer a lot of control and flexibility to write whatever they want instead of doing unnecessary things implicitly.
8. **Consistency:** The framework should be consistent at all levels. Consistency applies to everything from low-level (the Python coding style used) to high-level (the “experience” of using Django).

1. **The MVT Architecture:**

You already might have heard about MVC(Model View Controller) architecture. Django’s architecture is called MVT(Model View Template).

1. **Model:** Model corresponds to the data model that we are going o use in the application.
2. **View:** View here is similar to the controller in MVC. The job of the controller is to control the application flow (along with the URLs file) and execute the business logic.
3. **Template:** Template is similar to the view in MVC. It control’s the look and feel of your web application.



1. **Django Installation:**

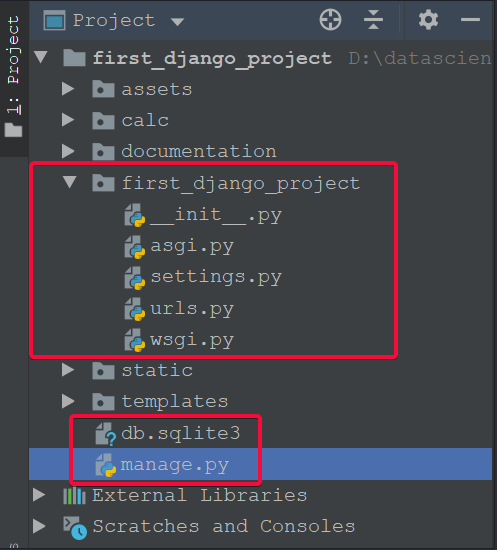
Django can be easily installed using the following commands:

* pip install django
* python -m pip install django
* conda install Django

**Note**: It is recommended to have separate environments for separate projects

1. **Steps to create a Django Project(with UI):**

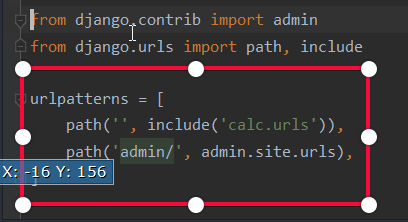
* Navigate to the project directory where you want to create your Django project and enter the command django-admin startproject project name.
* After this, the following project structure will automatically get created, as highlighted below:



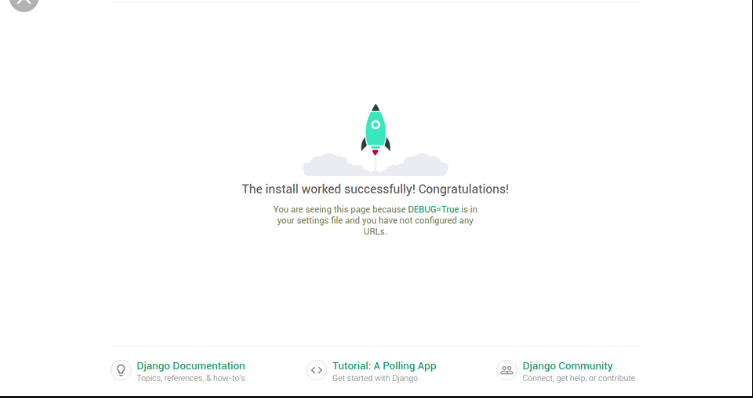
* The ‘manage.py’ file will be used to start and run your web server.
* The ‘ settings.py’ file controls the settings for your project like:

1. Debug is set to true or not
2. Installed apps
3. Installed middlewares
4. The file location for the static content like css.

* The ‘urls.py’ file contains the information about the URL routing as shown below:



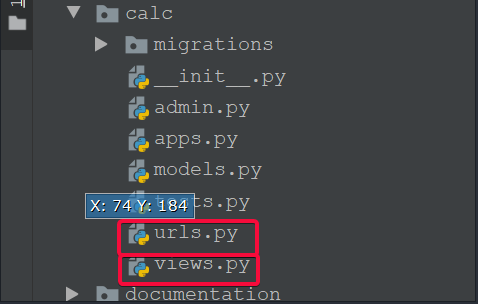
* After this, if you go ahead and enter the command: python manage.py runserver, you’ll see the following page(default – localhost:8000):



This means that the Django installation is working fine.

Now, we need to create our custom components for the project. *Our goal is to build a simple UI for a calculator app*.

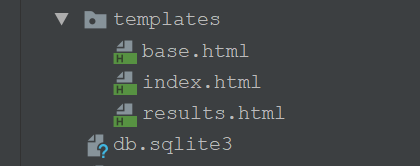
* Run the command python manage.py startapp calc to create a new app called ‘calc’. We’ll write the calculator logic here. The project structure looks like:



a) We’ll use ‘views.py’ to write the actual logic for calculator

b) We’ll use ‘urls.py’ to redirect the call to the appropriate method in the view.

* Now, we need to create the templates folder for UI. The folder structure looks like:



* The code for all the files above are as follows:

**base.html:** the file to maintain a common structure for the pages.

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <link rel="stylesheet" href="{% static './css/main.css' %}">  
 {% block head %}  
  
 {% endblock %}  
</head>  
<body>  
 {% block body %}  
  
 {% endblock %}  
</body>  
</html>

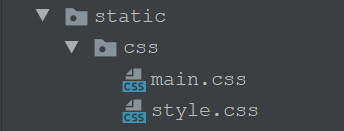
**Index.html:** the landing page or the home page.

{% block head %}  
  
<title>Calculator</title>  
  
  
{% endblock %}  
  
{% block body %}  
<div class="content">  
 <h1 style="text-align: center">Calculator</h1>  
  
 <div class="form">  
 <form action="/math\_operation" method="POST">  
 {% csrf\_token %}  
 <label for="operation">Choose a Mathematical Operation</label>  
  
<select id="operation" name="operation">  
 <option value="add">add</option>  
 <option value="subtract">subtract</option>  
 <option value="multiply">multiply</option>  
 <option value="divide">divide</option>  
</select>  
 <input type="text" name="num1" id="num1">  
 <input type="text" name="num2" id="num2">  
 <input type="submit" value="Calculate">  
 </form>  
 </div>  
</div>  
{% endblock %}

**Results.html:** the page to show the results after calculation.

<!DOCTYPE html>  
<html lang="en" >  
  
<head>  
 <meta charset="UTF-8">  
 <title>Review Page</title>  
 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/normalize.min.css"/>  
 <link rel="stylesheet" href="./style.css">  
  
</head>  
<body>  
 <div class="table-users">  
 <div class="header">Calculation Result <br></div>  
  
 {{result}}  
</div>  
</body>  
</html>

* We can keep the CSS in the static folder.



* But for the HTML files to look for CSS, we need to add the path of the static directory in the ‘settings.py’ as shown below:



For the changes to work, we need to run the command: python manage.py collectstatic

* Now, we’ll write the business logic(to build calculator) in the ‘views.py’ file.

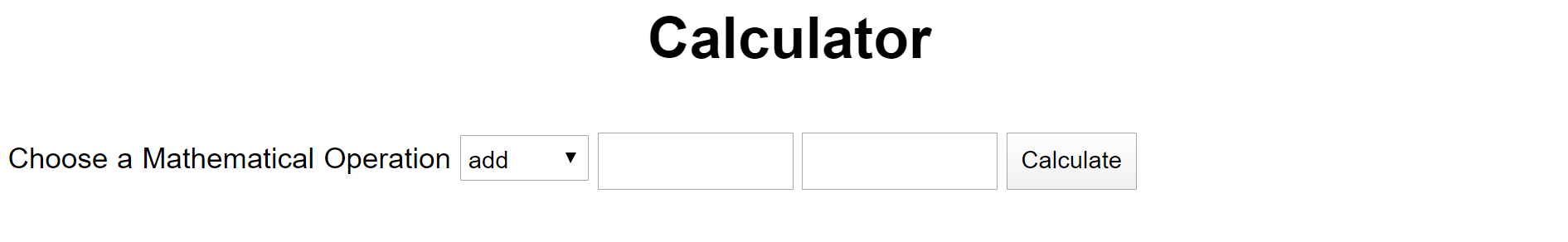
from django.shortcuts import render  
from django.http import HttpResponse, JsonResponse, HttpRequest  
from rest\_framework.views import APIView  
from rest\_framework.decorators import api\_view  
from django.utils.datastructures import MultiValueDictKeyError  
import json  
from bs4 import BeautifulSoup as bs  
  
# Create your views here.  
  
def home(request):  
 return render(request, 'index.html')  
  
  
def math\_operation(request):  
 operation = request.POST['operation']  
 num1 = int(request.POST['num1'])  
 num2 = int(request.POST['num2'])  
 if operation == 'add':  
 r = num1 + num2  
 result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)  
 if operation == 'subtract':  
 r = num1 - num2  
 result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)  
 if operation == 'multiply':  
 r = num1 \* num2  
 result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)  
 if operation == 'divide':  
 r = num1 / num2  
 result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)  
 return render(request, 'results.html', {"result": result})  
  
  
@api\_view(["POST"])  
def via\_postman(request):  
 request\_str=(request.body).decode("utf-8")  
 request\_json=json.loads(request\_str)  
 print(request\_json)  
 operation = request\_json["operation"]  
 print("Operation : ", operation)  
 num1 = int(request\_json["num1"])  
 num2 = int(request\_json["num2"])  
 #operation = "add"  
 #num1 = 2  
 #num2 = 3  
 if operation == 'add':  
 r = num1 + num2  
 result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)  
 if operation == 'subtract':  
 r = num1 - num2  
 result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)  
 if operation == 'multiply':  
 r = num1 \* num2  
 result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)  
 if operation == 'divide':  
 r = num1 / num2  
 result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)  
 return HttpResponse(result)

* Now, we need to update the ‘urls.py’ file to include the new methods:

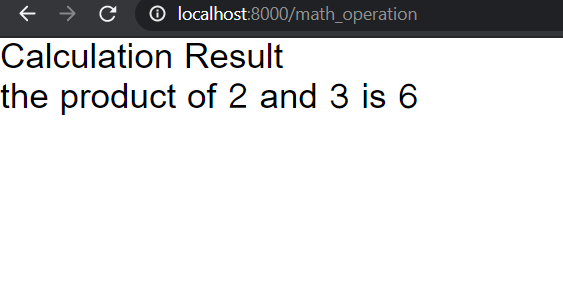
from django.urls import path  
from rest\_framework import routers  
from . import views  
  
router=routers.DefaultRouter()  
router.register('via\_postman',views.via\_postman, basename='via\_postman')  
urlpatterns=[  
 path('',views.home, name='home'),  
 path('math\_operation', views.math\_operation, name='math\_operation'),  
 path('via\_postman', views.via\_postman, name='via\_postman')  
]

* We can go ahead and run the Django application by entering the command: python manage.py runserver

The homepage looks like:



* You can enter any two numbers to do the mathematical calculation on it. For example, if you enter 2 and 3 and operation as multiply, you’ll get:



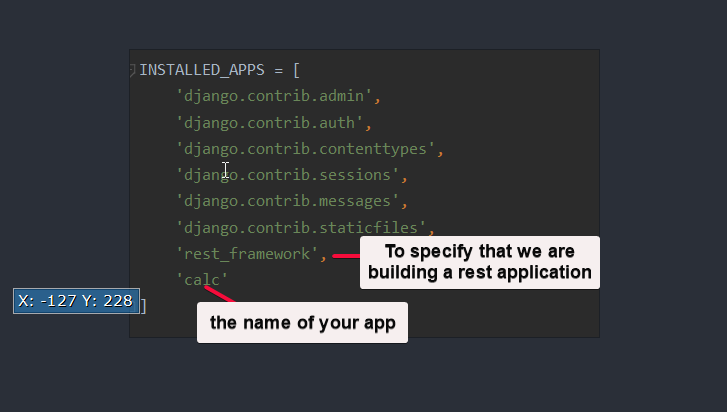
1. **Steps to create a Django Project(REST API):**

Instead of building an application with UI, we can also build a simple REST API using Django. The steps to do that are as follows:

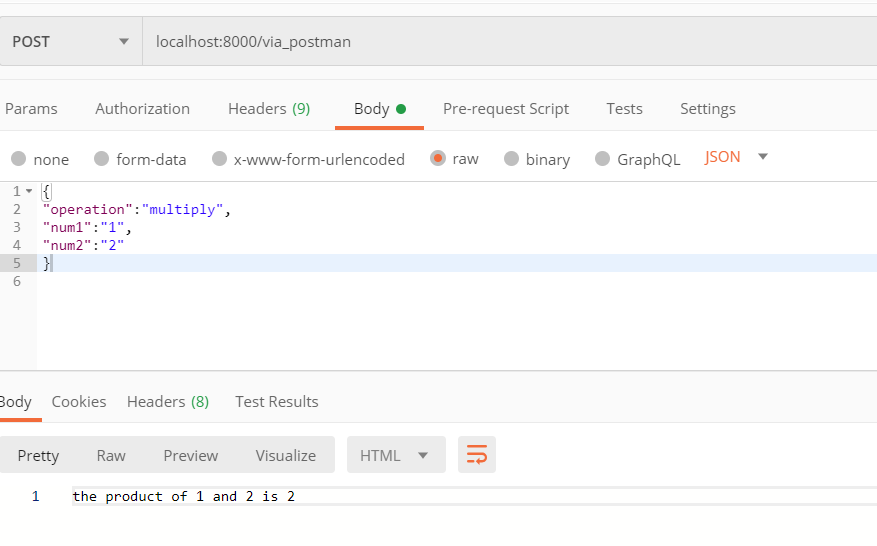
* All of the above steps will remain the same, except:

1. the part where we are creating the UI using the templates folder
2. creation of static folder and CSS inside it
3. addition of the path of static folder in ‘settings.py’

* The extra thing to be done is🡪 in ‘settings.py’, in the installed apps section, add the following two entries:



* We can go ahead and run the Django application by entering the command: python manage.py runserver
* It can be tested from SOAPUI or postman as:



References:

* Django documentation
* Wikipedia