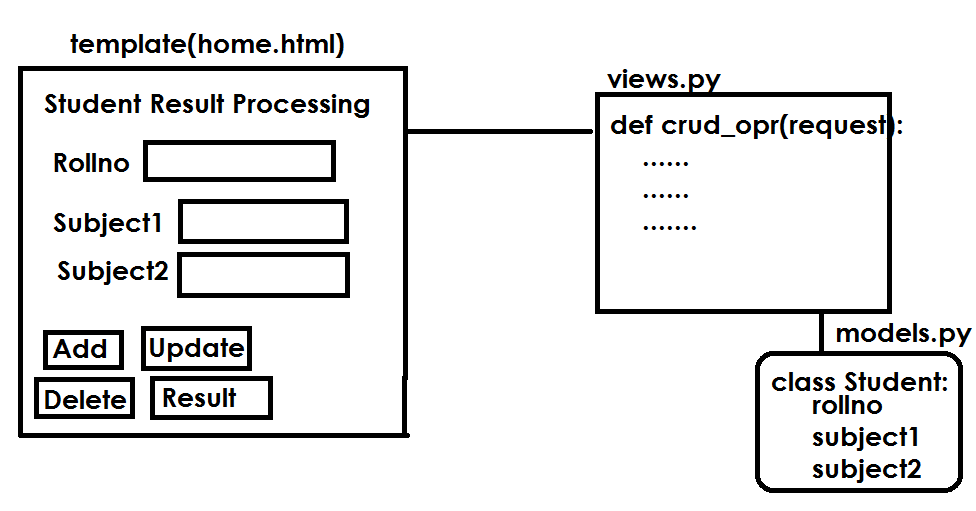
**Perform CRUD operations**

Application to perform the following operations

1. C 🡪 Inserting
2. R 🡪 Reading
3. U 🡪 Updating
4. D 🡪 Deleting



**Request methods**

1. GET
2. POST

POST request is to perform operations which do changes at server side. Inserting, updating, deleting

GET request is used to read information from server. This request does not do any changes at server

request.GET[‘parameter-name’]

request.POST[‘parameter-name’]

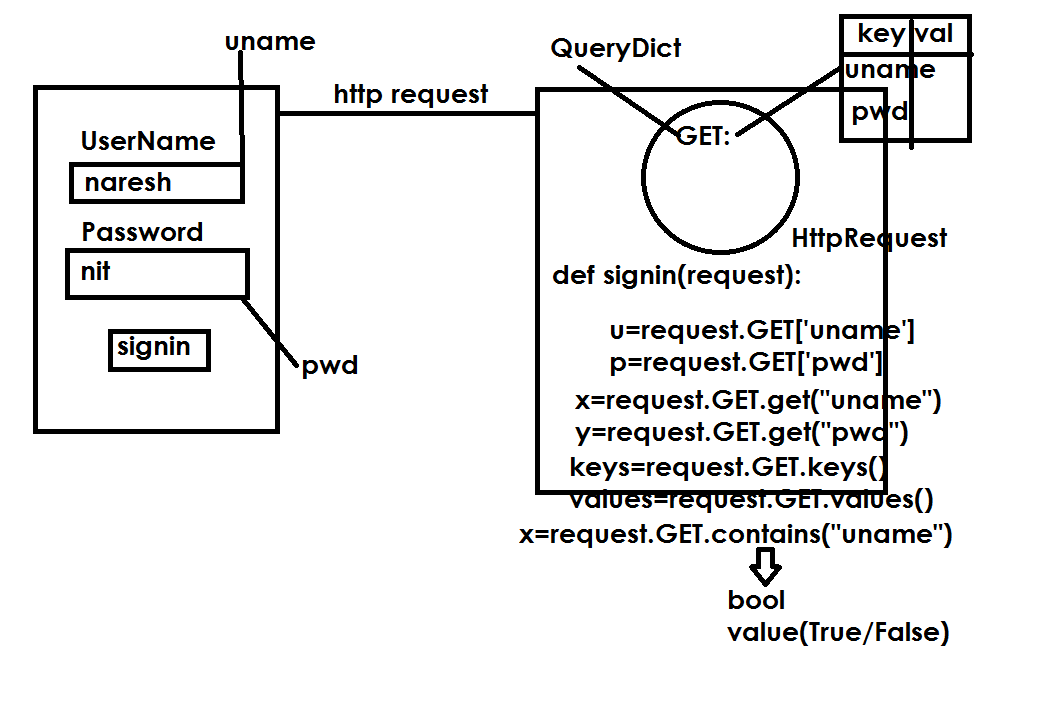
request.GET.get(‘parameter’)

**QueryDict objects**

In an **[HttpRequest](https://docs.djangoproject.com/en/4.0/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** object, the [**GET**](https://docs.djangoproject.com/en/4.0/ref/request-response/#django.http.HttpRequest.GET) and [**POST**](https://docs.djangoproject.com/en/4.0/ref/request-response/#django.http.HttpRequest.POST) attributes are instances of **django.http.QueryDict**, a dictionary-like class customized to deal with multiple values for the same key. This is necessary because some HTML form elements, notably **<select multiple>**, pass multiple values for the same key.

The **QueryDict**s at **request.POST** and **request.GET** will be immutable when accessed in a normal request/response cycle

QueryDict object contains request parameters or request string. The values send by client to server.



**Student Result Processing**

1. Create a project

django-admin startproject crudproject

1. Create application within project

python manage.py startapp crudapp

1. models.py

from django.db import models

# Create your models here.

class Student(models.Model):

    rollno=models.IntegerField(primary\_key=True)

    subject1=models.IntegerField()

    subject2=models.IntegerField()

1. templates

home.html

<html>

<body>

<form action="/crud">

    <h1>

        Rollno  <input type="text" name="rno"><br>

        Subject1<input type="text" name="sub1"><br>

        Subject2<input type="text" name="sub2"><br>

        <input type="submit" value="Add" name="b">

        <input type="submit" value="Update" name="b">

        <input type="submit" value="Delete" name="b">

        <input type="submit" value="Result" name="b">

    </h1>

</form>

<h1>{{msg}}</h1>

</body>

</html>

1. views.py

from django.http import HttpResponse

from django.shortcuts import render

from crudapp.models import Student

# Create your views here.

def crud\_opertions(request):

    if request.method=="GET":

        b=request.GET['b']

        if b=="Add":

            resp=insert\_data(request)

            return resp

        elif b=="Update":

            resp=update\_data(request)

            return resp

        elif b=="Delete":

            res=delete\_data(request)

            return res

        elif b=="Result":

            res=result(request)

            return res

def insert\_data(request):

    rno=request.GET.get("rno")

    s1=request.GET.get("sub1")

    s2=request.GET.get("sub2")

    stud=Student(rollno=rno,subject1=s1,subject2=s2)

    stud.save()

    msg={'msg':"Marks are Added"}

    r=render(request,"home.html",context=msg)

    return r

def update\_data(request):

    rno=request.GET.get("rno")

    stud=Student.objects.get(rollno=rno)

    stud.rollno=rno

    stud.subject1=request.GET.get("sub1")

    stud.subject2=request.GET.get("sub2")

    stud.save()

    msg={'msg':"Marks updated"}

    r=render(request,"home.html",context=msg)

    return r

def delete\_data(request):

    rno=request.GET.get("rno")

    stud=Student.objects.get(rollno=rno)

    stud.delete()

    msg={'msg':"Marks Deleted"}

    r=render(request,"home.html",context=msg)

    return r

def result(request):

    rno=request.GET.get("rno")

    stud=Student.objects.get(rollno=rno)

    output=f'''<h2>Rollno {rno}<br>

         Subject1 {stud.subject1}<br>

         Subject2 {stud.subject2}<br>'''

    res="pass" if stud.subject1>=40 and stud.subject2>=40 else "fail"

    output=output+"Result "+res

    return HttpResponse(output)

def home(request):

    r=render(request,"home.html")

    return r

1. settings.py

import os.path

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'crudapp',

]

TEMPLATE\_PATH=os.path.join(BASE\_DIR,"templates")

TEMPLATES = [

    {

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

        'DIRS': [TEMPLATE\_PATH],

        'APP\_DIRS': True,

1. urls.py

from django.contrib import admin

from django.urls import path

from crudapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('home/',views.home),

    path('crud/',views.crud\_opertions),

]

**Views are two types**

1. **Function Based Views (FBV)**
2. **Class Based Views (CBV)**

Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

* Organization of code related to specific HTTP methods (**GET**, **POST**, etc.) can be addressed by separate methods instead of conditional branching.
* Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

**Generic View**

**Generic View** does not have any predefined functionality.

A generic class based view inherits **View class** from django.views module.

class <class-name>(View): 🡪 Inheritance

def get(self,request):

pass

def post(self,request):

Class based view override two methods of View class to handle request.

If request method is GET, override get() method of View

If request method is POST,override post() method of View

|  |  |
| --- | --- |
| FBV | CBV |
| def fun1(request):  if request.method==”GET”:  ….  If request.method==”POST”:  …. | class CBV(View):  def get(self,request):  ……  def post(self,request):  ….. |

Urls.py

url-pattern

path(‘x/’,views.fun1)

path(‘y/’,views.CBV.as\_view())

Django’s URL resolver expects to send the request and associated arguments to a callable function, not a class, class-based views have an **[as\_view()](https://docs.djangoproject.com/en/4.0/ref/class-based-views/base/" \l "django.views.generic.base.View.as_view" \o "django.views.generic.base.View.as_view)** class method which returns a function that can be called when a request arrives for a URL matching the associated pattern. The function creates an instance of the class, calls [**setup()**](https://docs.djangoproject.com/en/4.0/ref/class-based-views/base/#django.views.generic.base.View.setup) to initialize its attributes, and then calls its [**dispatch()**](https://docs.djangoproject.com/en/4.0/ref/class-based-views/base/#django.views.generic.base.View.dispatch) method. **dispatch** looks at the request to determine whether it is a **GET**, **POST**, etc, and relays the request to a matching method if one is defined, or raises **[HttpResponseNotAllowed](https://docs.djangoproject.com/en/4.0/ref/request-response/" \l "django.http.HttpResponseNotAllowed" \o "django.http.HttpResponseNotAllowed)** if not: