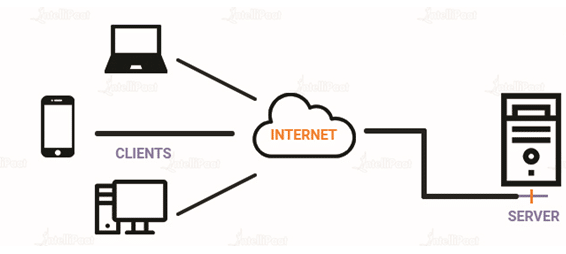


* Django/flask is a web framework not a library.
* Framework is a structure intended to serve as a guide for building applications that expands the structure into something useful.
* Framework is an application uses our code to build an application.
* Library is a predefined code that we use to build an application.
* Django is a python based free and open source web framework which follows the Model View Template (MVT) architectural pattern.
* Django is maintained by Django software foundation.
* Django based applications are YouTube, Dropbox, Mozilla, Spotify, Instagram, etc.
* Features of Django are Fast Development, Provide web server for development and testing, scalability, Loaded with apps, Secure, etc.

**Client Server Architecture :**



**Django Installation :**

* Check whether python is installed or not (python –version or python).
* Check the pip command (pip –version or pip).
* Install Django (pip install django).
* Check the Django (django-admin –version or python –m django --version).

All applications in Django are called as web app and All projects in Django are called as website.

Django is a framework. We can create our project with simple command.

**Django Project Creation :**

* Create a workspace directory.
* Create a first project (Website).

1. django-admin startproject firstproject or python -m django startproject firstproject

* \_\_init\_\_.py is an empty file its presence make the folder as python package.
* Setting.py contains project related setting statements.
* Urls.py contains url pattern of web pages.
* Wsgi.py is a webserver gateway interface, it is a special file, no modification in this file is required.
* Asgi.py is an asynchronous server gateway interface, like wsgi.py, it describes a common interface between a python web app and the web server.
* Manage.py is a program to do some initial stuff, never edit this file.

1. Come in your project directory (cd firstproject).
2. Run the web server (python manage.py runserver).

When project is not creating then create virtual environment.

1. Create virtual environment (py –m venv myworld).
2. Activate virtual environment (myworld\scripts\activate.bat).
3. Go to your environment or workspace (cd myworld).
4. Create project (django-admin startproject firstproject).
5. Change directory (cd firstproject).
6. Run server (python manage.py runserver).

Create a web app (python manage.py startapp myapp).

Update setting.py in child myproject directory (in installed apps).

Define the view in myapp view.py file :

|  |
| --- |
| from django.http import HttpResponse  def greeting(request):  s = "Hello and welcome to my firstproject"  return HttpResponse(s)  def myclass(request):  c = "This is my AIDS class"  return HttpResponse(c)  def myname(request):  n = "Nikhlesh Shukla  return HttpResponse(n) |

Set url in myproject urls.py file.

|  |
| --- |
| from myapp import views  urlpatterns = [  path('hello/',views.greeting),  path('class/',views.myclass),  path('name/',views.myname),  path('admin/', admin.site.urls),  ] |

Request and Response :

**Multiple Apps Multiple Views :** Steps to perform.

* Create project (django-admin startproject firstproject or python -m django startproject firstproject).
* Create apps (python manage.py startapp myapp) for one app.
* Update settings.py file.
* Define views in views.py file.
* Set urls in urls.py file.

**App Level urls :** for them we have to create a new urls file.

* Click app name.
* Click new file.
* Assign name urls.py
* First app content in urls.py file :

from django.urls import path

from third1app import views

urlpatterns = [

path('home/',views.home),

path('about/',views.about),

path('contact/',views.contact)

]

* Second app content in urls.py file :

from django.urls import path

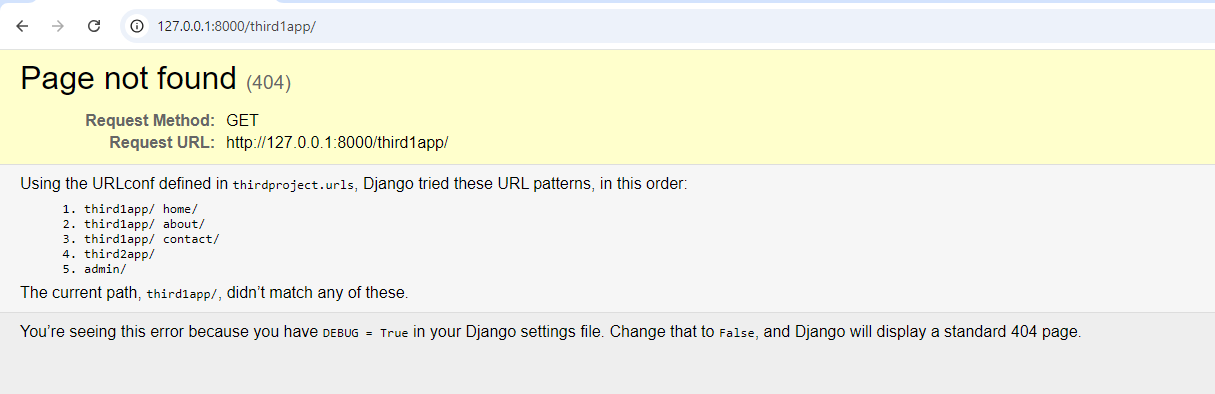
from third2app import views

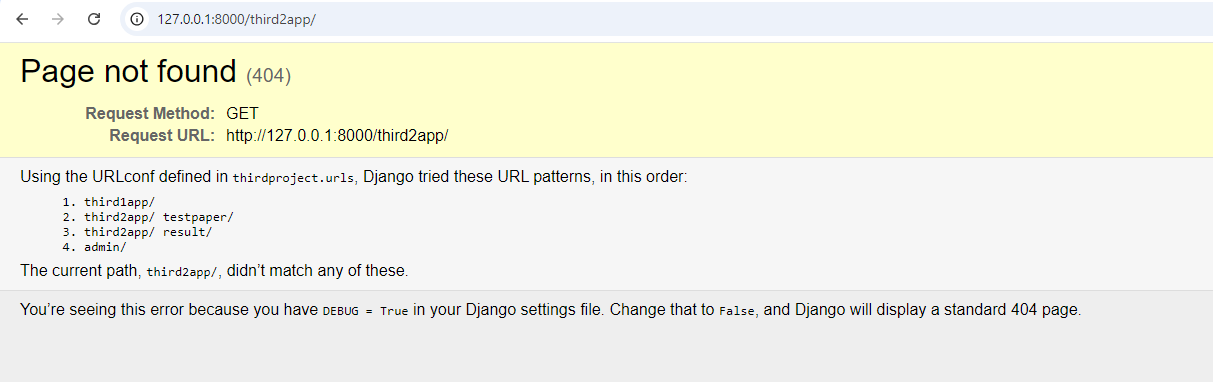
urlpatterns = [

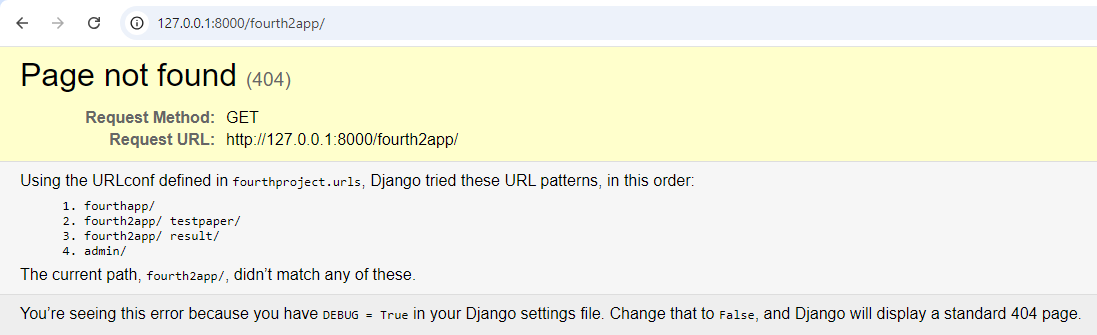
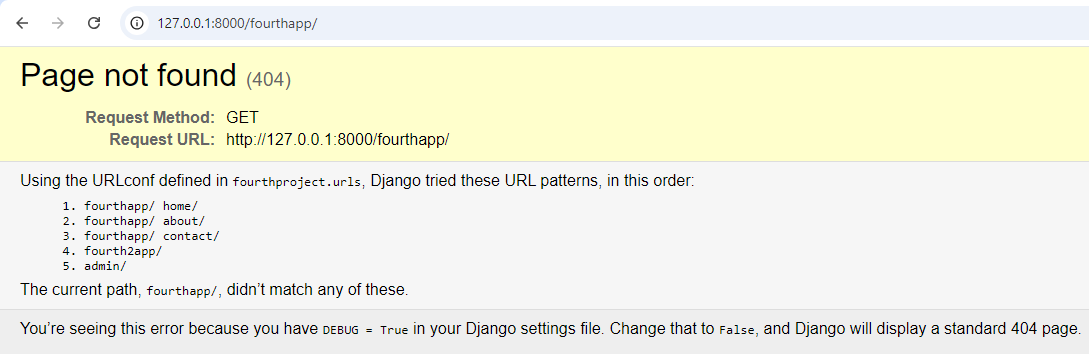
path('testpaper/', views.testpaper),

path('result/', views.result)

]

* After running server :





**Dynamic Content in Template :**

Jinja2 technology is used to implement this concept in html file which is created in app.

Usually we cannot write language ingredients like variable decision and iterative controls in html, but with the help of Jinja2, it become possible.

There are three things related to templates you need to explore :

* Template Tags
* Template Variable
* Template filter

**Template Variable :**

Template variable can be used in html using {{variable}} (see quiz\_eg project).

**Template Tags :**

We can perform programming logic in html using template tags.

There are several built-in tags and we can define our own custom template tag.

Example : {% Template Tag %}

Some built-in template tags are :

* block : Define a block that can be overwritten by child template.
* csrf\_token : This is used for CSRF (Cross Site Request Forgiries) Protection.
* for : Loop.
* extends : Signals that this template extends a parent template.
* if : Decision control.
* load : Load a custom template tag.
* now : Display current date and time or both.

**Template Filters :**

Filters are used to transform the values of variables and tags arguments.

There are several built-in template filters and we have an option to define our own, known as custom template filters.

* add : {{ variable | add : “2” }}
* capfirst : {{ text | capfirst }}
* cut : {{ text | cut : “ ” }} (Cut arguments from the text).
* dictsort : {{ value | dictsort : “key” }} (Sort list of dict by given key).
* dictsortreversed :
* join : Join a list with a string.
* length : Count length.
* lower :
* upper :
* random : Returns a random item from the list.
* wordcount :
* static :

**Extending Templates & Static Folder :**

**Database :** Database is a place where your all application data is stored. This is business data.

**DBMS :** DBMS stands for Database Management System. It is used to manage database with the help of like Oracle, MySQL, DB2, SQlite, etc.

**SQLite :** Already configured database software in Django is SQLite. To use another database system you need to make changes in setting.py file. To change database we have to change “db.sqlite3” to that database.

**How to access database ?**

Usually we need to use query language SQL (Structured Query Language) to communicate with database software in order to perform any operation o database. But Django provides simple way to communicate with database system.

**Model :** Django uses the concept of models to perform database operations and takes care of SQL statements. We have to create models in our application and map them with database tables. Models concepts is an abstraction to use databases. Our App :

|  |  |  |  |
| --- | --- | --- | --- |
| Uses Models | Django Library | SQLite | Database |

Model is a key concept of MVT design pattern. Model concept is use to handle database. In Django we have to write only python code and SQL statements to communicate database. Inside the app model.py file exist.Model is a python class. We have to create a subclass of Model to represent a database table.

**Define Model :** any\_app/models.py

from django.db import models

class employee(models.model):

eno = models.IntegerField()

ename = models.charField(max \_length = 30)

esal = models.floatfield()

|  |  |  |  |
| --- | --- | --- | --- |
| eid | eno | ename | esal |
|  |  |  |  |

If any field is primary key then not generate id column else id column will be generate.

**Creating database Table :** After defining model class you need to perform following two steps to actually create database table.

1. Make Migration (Preparation for table creation) (python manage.py makemigrations appname).
2. Migrate (create a table) (python manage.py migrate appname).

Model to SQL statements converter code (python manage.py sqlmigrate myapp 0001).

**Admin App :** Django admin app is a built-in app. It is a CRUD (Create Read Update Delete) user interface of all your models. It is an app for admin of the website.

**How to use admin app :** In the browser window (<https://localhost:8000/admin>).

You might have noticed prewritten entry in urlpatterns for admin.

We can see the admin login page, it requires login credientials, which has to be created.

We can create any number of users but one has to be a super user with all the priviledges.

To create super user (python manage.py createsuperuser

Username : name\_of\_user

Email :

Password : [name\_of\_user@gmail.com](mailto:name_of_user@gmail.com)

We need to register model to the admin interface In testapp/admin.py

|  |
| --- |
| from Django.contrib import admin  from testapp.models import Employee  admin.site.register(Employee) |

testapp/models.py

|  |
| --- |
| def \_\_str\_\_(self):  return self.ename |

If any updation in models.py then run two command (python manage.py makemigration) and (python manage.py migrate).

**CRUD Operations :** C-Create, R-Read, U-Update, D-Delete.

**QuerySet :** QuerySet is built up as list of objects. It is a collection of data from a database.

**Create :** Create is used to add new record in the table.

|  |
| --- |
| employee = Employee(eno = 4, ename = “Nikhlesh”, esal = 1500)  employee.save() |

**Read :** Read is extracting data from the table without deleting them from the table.

|  |
| --- |
| employees = Employee.objects.all() (QuerySet of employee objects).  emp = Employee.objects.all()[2] (An employee at index 2).  employees = Employee.objects.all().values() (QuerySet of employees each example as a dictionary with column name as a key).  Enames = Employee.objects.values\_list(‘ename’) (QuerySet of list of tuples, where each tuple contain only employee name).  X = Employee.objects.values\_list(‘ename’, ‘esal’) (QuerySet of list of tuples, where each tuple contain only employee name and salary).  employees = Employee.objects.filter(esal\_\_get = 50000).values() (QuerySet of employees with salary is greater than 50000).  employees = Employee.objects.all().order\_by(‘ename’).values() (QuerySet of employees ordered by employee names in ascending order)  employees = Employee.objects.all().order\_by(‘-ename’).values() (QuerySet of employees ordered by employee names in descending order because of ‘-ename’). |

**Field Lookup :** gt, lt, gte, lte, isnull, startswith, endswith, exact, range.

**Update :** Update is used to modify data of record or orders of database table.

|  |
| --- |
| emp = Employee.objects.filter(‘ename’ = ‘Nikhlesh’).values()  e = Employee()  e.id = emp[0][‘id’]  e.eno = emp[0][‘eno’]  e.ename = emp[0][‘ename’]  e.sal = 28500  e.save() |

**Delete :** Delete is used to delete record or records from database table.

|  |
| --- |
| E = Employees.objects.filter(id = 3)  e.delete() |

**Steps to perform CRUD Operations through terminal :**

* Go to shell prompt (python manage.py shell).
* Import created model with name Employee {from dbapp.models import Employee}.
* Store them into a variable {employee = Employee.objects.all().values()}.
* Read all records using print statement {print(employee)}.
* Create a new record using {employee = Employee(eno = 4, ename = “Nikhlesh”, esal = 1500)}.
* Save created record { employee.save()}.
* Read all records using print statement {print(employee)}.
* Delete a record from Employee using {e = Employee.objects.filter(ename = 'Pankaj')}.
* Delete {e.delete()}.
* Store them into a variable {employee = Employee.objects.all().values()}.
* Read all records using print statement {print(employee)}.
* Update this record salary from 300 to 1000 using {emp = Employee.objects.filter(ename = 'Nikhil Vishwakarma').values()}.
* e = Employee()
* >>> e.id = emp[0]['id']
* >>> e.eno = emp[0]['eno']
* >>> e.ename = emp[0]['ename']
* e.esal = 1000
* e.save()
* Store them into a variable {employee = Employee.objects.all().values()}.
* Print the result after update using {print(employee)}.