**Django**

It is a python based web framework used for rapid web development.

**Framework:-**

These are libraries which are bound with rules.

**Other web framework of python:-**

* ***TurboGears*** - With TurboGears, you can create a database-driven, ready-to-extend application in just a few minutes.
* ***CherryPy*** - It is a pythonic, object-oriented HTTP framework.
* ***Flask*** - Flask is a microframework for Python. It includes a built-in development server, and unit-testing support.
* ***Pylons*** - Pylons is a lightweight Web framework aiming at flexibility and rapid development.
* ***Tornado*** - While Tornado isn’t that famous, it is great with non-blocking I/O.

**Latest Version of Django:-**

Version :- 4.2.3

**Advantages of Django:-**

* **Security**
* **Faster Processing**
* **Rapid development**
* **Versatility** - You will easily manage content management, scientific computing platforms, even large organizations, with help of Django.
* **Easy to work with** - learn it quickly and easily.
* **Open source** - access easily
* **Advanced functionalities** - implement anything eg: machine learning
* **Scalability** - it is logical that most downloaded sites prefer Django to carry out traffic-related requirements.
* **Endless resources** - Python can boast a large library that can be used for building large-scale applications

**Features**:-

* **IOC -** Inversion of Control
* Model / ORM queries
* Administration
* Applications (Module)

**SoftWare Patterns (Structure)**

1. MVC [ Model View Controller ]
2. MVT [ Model View Template]

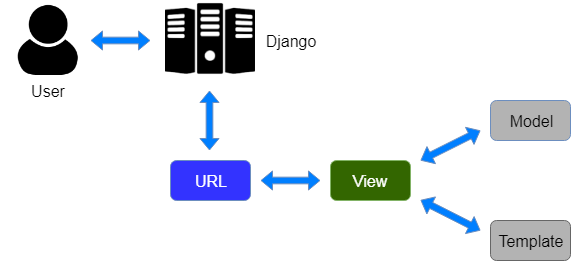
**MVC**



**MVT**



**Django Architecture**



**Django project Creation & Environment setup**

* *Create a new folder for your project*
* *Open folder and open cmd in desired folder*

1. **Virtual Environment** - Using virtual environments instead of installing one version of django globally or at the system level, we install different versions to each virtual environment, so that we can work on both the django projects in a single machine.

* Install virtualenv package.

**py -m pip install virtualenv**

* Create virtual environment in desired location

**py -m virtualenv env\_name**

* Activate virtual environment

**Env\_name\scripts\activate**

* Install needed package.

1. **Django Project creation**

* Install Django

**pip install django**

* Create a project using following code

**django-admin startproject project\_name**

**Skelton folder (Project1):**

* Project1( project directory)
* db.sqlite3( database file)
* Manage.py (command line management file)

**Important files in project1** ( project directory)

manage.py

It is a command line utility that helps us in interacting with our django project.

Just used to serve our app.

db.sqlite3

It is a database file used to store data.

init.py

It is an empty file that tells python that this directory should be considered as a python project or python module.

settings.py

It is considered as all settings or configurations for the project of the file

urls.py

It contains all the url declarations of the project of the file.

Asgi.py - asynchronous server gateway interface

It is the entry point for asgi compatible web servers to serve the project.

Wsgi.py - web server gateway interface

It is the entry point for wsgi compatible web servers to serve the project.

**Important contents in settings.py**

BASE\_DIR

BASE\_DIR points to the top hierarchy of the project i.e. **project1**,

whatever paths we define in the project are all relative to BASE\_DIR.

SECRET\_KEY

It is a key used for encryption and decryption i.e., security purposes.

ALLOWED\_HOSTS

ALLOWED\_HOSTS is a list having addresses of all domains which can run your Django Project.

DEBUG

Django provides an inbuilt Debugger which makes the developer’s life very easy. We can use it by doing:

**DEBUG = True** // It is Default value and is preferred in only the Development Phase.

In production **DEBUG = False** is preferred.

INSTALLED\_APPS

In this section we mention all apps that will be used in our Django project.

MIDDLEWARES

These are software components that act in between client and server sides for specific functionalities.

CSRF - cross site request forgery

DATABASES

Django officially supports the following databases

* PostgreSQL
* MariaDB
* MYSQL
* Oracle
* SQLite - Default

URL variables

URL variables are relative to BASE\_DIR. These variables are used to store files either media or static.

MEDIA\_URL

MEDIA\_URL is the relative path to BASE\_DIR. This variable is used to store the media files.

STATIC\_URL

STATIC\_URL is the relative path to BASE\_DIR. This variable is used to store the static files.

1. **Application creation**

Django projects are collections of applications.

* Open the project path where the application is created using following code

**cd project\_name**

* Create an application using the following code.

**py manage.py startapp app-name**

**Important files in account**( application directory)

\_\_init\_\_.py

This file provides the same functionality as that in the \_init\_.py file in the Django project structure.

It is an empty file and does not need any modifications.

It just represents that the app directory is a package.

admin.py

#### admin.py file is used for registering the Django models into the Django administration.

It is used to display the Django model in the Django admin panel.

It performs three major tasks:

* Registering models
* Creating a Superuser
* Logging in and using the web application

apps.py

#### apps.py is a file that is used to help the user include the application configuration for their app.

models.py

#### models.py represents the models of web applications in the form of classes.

Models define the structure of the database. It tells about the actual design, relationships between the data sets, and their attribute constraints.

views.py

#### Views are also an important part when we talk about the Django app structure.

#### Views provide an interface through which a user interacts with a Django web application

tests.py

#### Tests.py allows the user to write test code for their web applications. It is used to test the working of the app.

**PROJECT ACTIVITIES**

1. **Django Views**

* Function based views
* Class based views

Function based views

**view.py**

from django.http import HttpResponse

def members(request):

return HttpResponse("Hello world!")

Django loads the appropriate view, passing the [**HttpRequest**](https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpRequest) as the first argument to the view function. Each view is responsible for returning an [**HttpResponse**](https://docs.djangoproject.com/en/4.2/ref/request-response/#django.http.HttpResponse) object.

from django.shortcuts import render

def login(request):

return render(request,"login.html")

**render** method is used to return a html page.

1. **Django URLs**

Each view needs to be mapped to a corresponding URL pattern. This is done via a Python module called URLConf(URL configuration)

Function based urls

from account.views import first

Urlpatterns = [

Path (‘admin/’,admin.site.urls),

Path (‘fst’,first), ]

*fst - path url*

*first - function name imported from views*

1. **Django Templates**

Create a templates folder inside the application folder, and create a HTML file named login.html.

Look up the INSTALLED\_APPS[] list and add the application app like this:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

**'account'**

]

**Django Template Language**

1. **Presenting a variable [ { variable } ]**

* "**render**" is the most used function in django. It combines a given template with a given context dictionary and returns an [HttpResponse](https://docs.djangoproject.com/en/dev/ref/request-response/#django.http.HttpResponse) object with that rendered text.
* It takes three arguments: "request", "template\_name" and "context" dictionary.
* In template we can access the context dict keys as **names\*\*or** variables\*\* and display them like "{{ <variable/name>}}".

Example:-

views.py

from django.shortcuts import render

def user\_data(request):

name = “narjas”

names = [“name1”,“name2”,“name3”]

dict= [

{"name": "name1", "age": "10"} ,

{"name": "name2", "age": "10"} ,

{"name": "name3", "age": "10"} ,

]

return render(request, "user\_template.html", {“data”:name, “lname”:names, “dname”:dict)

user\_template.html

<html>

<head>

<title>User Information</title>

</head>

<body>

<p>Name: {{data }}</p>

<p>Names: {{ lname }}</p>

</body>

</html>

1. **Tags[ {% tag %} ]**

Some tags require beginning and ending tags (i.e. {% tag %} ... tag contents ... {% endtag %}).

* [**for**](https://docs.djangoproject.com/en/4.2/ref/templates/builtins/#std-templatetag-for) - Loop over each item in an array.

Example1:-

{% for i in dname %}

<p> {{ i }} </p>

{% endfor %}

Example2:-

{% for i in dname %}

<p> {{ i.name }} </p>

<p> {{ i.age }} </p>

{% endfor %}

* **if else**.

Example:-

{% if dname %}

<p> {{ dname }} </p>

{% else %}

<p> No item </p>

{% endif %}

**Django Working With Forms**

In HTML, a form is a collection of elements inside <form>..........</form> that allow a visitor to do things like enter text, select options, manipulate objects or controls and so on and then send that information back to the server.

A form must specify two things,

* **Where**: the URL to which the data corresponding to the user’s input should be returned
* **How**: the HTTP method the data should be returned by

**GET and POST**

GET and POST are the only HTTP methods to use when dealing with forms.

1. **GET**

Django’s login form is returned using the GET method, in which the browser bundles the submitted data into a string, and uses this to compose a URL.

The URL contains the address where the data must be sent, as well as the data keys and values.

1. **POST**

Django’s login form is returned using the GET method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response.

**Building a form**

* **Login.html**

<**form** action**=**" " method**=**"post">

{% csrf\_token %}

<**label** for**=**" ">Your name: </**label**>

<**input** id**=**" " type**=**"text" name**=**"uname" placeholder**=**"Username">

<**input** id**=**" " type**=**"password" name**=**"pwd" placeholder**=**"Password">

<**button**  type**=**"submit" >OK</**button**>

</**form**>

{% csrf\_token %} - **CSRF** for Cross Site Request Forgery

That is used to prevent malicious attacks. When generating the page on the server, it generates a token and ensures that any requests coming back in are cross-checked against this token.

* **views.py**

def login\_post(request):

if request.method=="GET":

return render(request,"login\_post.html")

elif request.method == "POST":

print(request.POST.get('uname'))

print(request.POST.get('pswd'))

return HttpResponse("post request")

* **urls.py**

from account.views import first, login\_post

Urlpatterns = [

Path (‘admin/’,admin.site.urls),

Path (‘fst’,first),

Path (‘login’,login\_post),

]

Class based views

**view.py**

from django.http import HttpResponse

from django.views.generic import View

class AddView(View):

def get(self,request,\*args,\*\*kwargs):

return render(request,"addition.html")

def post(self,request,\*args,\*\*kwargs):

n1=int(request.POST.get('num1'))

n2=int(request.POST.get('num2'))

return HttpResponse("Sum of %d and %d is %d " %(n1,n2,n1+n2))

* Import **View** class from **view.generic**
* Create a class that is inherited from **View** class
* Define a method ‘ **get ‘** that is inherited from the base class **View**.
  + The get method has 4 arguments
    - self
    - request
    - \*args
    - \*\*kwargs
* Define a method ‘ **post ‘** that is inherited from the base class **View**.
  + The get method has 4 arguments
    - self
    - request
    - \*args
    - \*\*kwargs

**urls.py**

from account.views import first, login\_post,AddView

Urlpatterns = [

Path (‘admin/’,admin.site.urls),

Path (‘fst’,first),

Path (‘login’,login\_post),

Path (‘addc’,AddView.as\_view()),

]