Django

- **Client-Server architecture**
- Server

--> it is a high configuration machine/system

- Client

--> client uses server's services

- e.g. If we make a flipkart's application in a laptop then it can only be used in that laptop. If we want to use that application world wide then we need to deploy that application on server

- Communication between client and server

--> it happens over a protocol called as http i.e. hyper text transfer protocol
--> http is a stateless protocol
--> http defines rules that 2 electronic entities which communicate over the internet follow
--> http defines methods to indicate desired action to be performed on the identified resource
e.g.

1. GET
2. PUT
3. POST
4. DELETE

- GET

--> GET method requests that the target resource transfer a representation of its state
--> GET requests should only retrieve data and should have no other effect

- POST

--> POST method is used to store data from client on server
--> It requests that the target resource process the representation enclosed in the request according to the semantics of the target resource.
e.g. It is used for posting a message to an internet forum, subscribing to a mailing list, or completing an online shopping transaction

- PUT

--> It is an update method. It makes changes to an already present data
--> It requests that the target resource create pr update its state with the state defined by the representation enclosed in the request.
--> A distinction from POST is that the client specifies the target location on the server

- DELETE

--> This method requests that the target resource delete its state

- Types of applications:

1. Mobile application
2. Web application
3. Desktop application
4. Cloud application--> Netflix, Amazon prime
5. Network application-->

- Web application

--> e.g. flipkart web application
--> we use web application through browser using url

- Designing a web application

--> There are different platforms to design a web application
e.g.

1. .Net

-->.Net framework is a software development framework for building and running applications on windows
--> .Net framework is part of .Net platform, a collection of technologies for building apps for Linux, macOS, Windows, iOS, Android, and more

2. Django

--> It is a web framework using which you can design web applications
--> The reasons to use Django:

1. Ridiculously fast
2. Fully loaded
3. Reassuringly secure
4. Exceedingly scalable
5. Incredibly versatile

> Run `django-admin version` or `python -m django --version` to display the current Django version

3. Flask

--> It is almost 99% same as Django.
--> It has lightweight architecture therefore not flexible
--> It is used to deploy Machine Learning models

- Framework

--> It consists of an inbuilt server along with all the useful environment

- Project

--> It is a collection of applications along with the setting to configure the applications
e.g.
--> a complete website of JetBrains can be called as a project

- Application

--> It is a computer program designed to carry out a specific task other than one relating to the operation of the computer itself, typically to be used by end-users
e.g. Developer Tools, Team Tools, Learning Tools, Solutions in the JetBrains project can be called as applications
-->> Any application is made up of layer of architect
-->> e.g.
computer architecture consists of 3 layers :

1. high level layer i.e. operating system layer
2. middle level layer i.e. kernel
3. hardware (final layer)

-->> Any application has :

1. front end

--> html, css, java script

2. middle layer that contains business logic

--> programming language like python, java, .Net

3. back end

-->contains database
-->> the above layers collectively represent an application

Creating a new project

- Command to create a project

--> django-admin startproject projectname

- Command to enter into project

--> cd projectname

- Command to run server

--> python manage.py runserver
--> Quit the server with CONTROL-C

---

16 Sep, 2022

- **Architecture of Django**

--> It follows MVT architecture

1. Model

--> It is the back end.
--> It contains database logic.
--> MySQL, Oracle, PostgreSQL, MongoDB

2. View

--> It is a middle layer.
--> It contains business logic.
--> It is written in python

3. Template

--> It is a front end.
--> It contains presentation logic.
--> HTML, CSS are used

• .Net architecture --> It follows MVC architecture 1. Model --> It contains business logic 2. Views --> It contains presentation logic 3. Controller --> It manages server as well as it integrates Model and View --> It is an intermediary	• Django architecture --> It follows MVT architecture 1. Model --> It is the back end. --> It contains database logic. --> MySQL, Oracle, PostgreSQL, MongoDB 2. View --> It is a middle layer. --> It contains business logic. --> It is written in python 3. Template --> It is a front end. --> It contains presentation logic. --> HTML, CSS are used

• Creating a project with single application

Step 1: Creating a new project

1. Command to create a project

--> **django-admin startproject projectname** or **python -m django startproject projectname**
--> **django-admin** is Django's command-line utility for administrative tasks
--> e.g.
django-admin startproject ICICI

>> This command creates ICICI project directory with ICICI python package
>> The outer ICICI/ root directory is a container for the project. Its name doesn't matter to Django; you can rename it to anything you like.
>> The inner ICICI python package/ directory is the actual Python package for the project. Its name is the Python package name you'll need to use to import anything inside it (e.g. ICICI.urls).
>> manage.py
    file gets created within ICICI project directory
    manage.py is automatically created in each Django project
    it is a command-line utility that lets you interact with this Django project in various ways
>> In addition to that 5 python files, namely

1. __init__.py,
2. asgi.py,
3. settings.py,
4. urls.py,
5. wsgi.py get created within ICICI python package

2. Command to enter into project

--> cd projectname
--> e.g.
  cd ICICI

3. Command to run server

--> python manage.py runserver
--> This command generates an url
--> This means Django project has been successfully executed

- __init__.py: An empty file that tells Python that this directory should be considered a Python package.

- wsgi.py and asgi.py are used for deployment of application

- wsgi.py

--> stands for web server gateway interface
--> It is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.

- asgi.py

--> asgi stands for asynchronous server gateway interface
--> asgi is a spiritual successor to wsgi

- settings.py

--> It is used for application and project level settings
--> A Django settings file contains all the configuration of the Django installation. This document explains how settings work and which settings are available.

- urls.py

--> It consists of application and project level urls

Step 2: Creating an application

Here creating an application named loan within the ICICI project
--> command to create an application

```
python manage.py startapp appname
```

--> e.g.

```
python manage.py startapp loan
```

--> the above command creates an app called loan
--> i.e. a python package named loan gets created
within loan package 6 files, namely

1. __init__.py,
2. admin.py,
3. apps.py,
4. models.py,
5. tests.py,
6. views.py and a python package named migrations get created

Step 3: Integrating the application with the project

register app in settings.py
--> add application name in list INSTALLED_APPS in settings.py module

Step 4: Writing business logic in views.py in loan package

--> python code is written in views.py
--> write

```
from django.http import HttpResponse  #this generates HttpResponse
def show(r):
    return HttpResponse('<h1>Welcome to portfolio app</h1>)
```

--> A view function, or *view* for short, is a Python function that takes a web request and returns a web response. This response can be the HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response. This code can live anywhere you want, as long as it’s on your Python path. There’s no other requirement–no “magic,” so to speak. For the sake of putting the code *somewhere*, the convention is to put views in a file called views.py, placed in your project or application directory.

Step 5: Creating url

--> in urls.py
--> write
from app_name import views
--> Note:
    from django.contrib import admin
    from django.urls import path are already present in urls.py
--> in urlpatterns list add path('url_name/', views.views_function_name),
--> The `urlpatterns` list routes URLs to views

Step 6: Execution

--> run server
--> python manage.py runserver

http://localhost:8000/welcome/
is same as
http://127.0.0.1:8000/welcome/

The IP address **127.0. 0.1** is called a loopback address.

---

17 Sep, 2022

- ## Creating a project with multiple applications

Step 1: Creating a new project

--> run the command
    django-admin startproject projectname
or
    python-m django startproject projectname
--> enter into the project
    cd projectname
--> check whether django project has successfully executed by using the command:
    python manage.py runserver
    The above command generates an url which means django project has been successfully executed

Step 2: Creating multiple applications

--> command to create an application is:

```
    python manage.py startapp appname1
    python manage.py startapp appname2
    .
    .
    .
```

Step 3: Integrating applications with the project

--> register the apps in project by writing application names in list INSTALLED_APPS in settings.py module

Step 4: Writing business logic in views.py of each application

--> python code is written in views.py

```
--> from django.http import HttpResponse     # This generates HttpResponse
--> def views_function_name1(r):
        return HttpResponse("<h1> Content </h1>")
.
.
.
```

Step 5: Creating url

--> in urls.py write

```
    from appname1 import views as v1
    from appname2 import views as v2
    from appname3 import views as v3
    .
    .
    .
```

--> in urlpatterns list add

```
    path('url_name/', v1.views_function_name1)
    path('url_name/', v2.views_function_name2)
    path('url_name/', v3.views_function_name3)
```

Step 6: Execution

--> runserver by using the command:

```
    python manage.py runserver
```

18 Sep, 2022

Creating a project with multiple applications that have multiple fuctions (Project level urls)

Step 1: Creating a new project

--> run the command

```
    django-admin startproject projectname
```

or

```
    python-m django startproject projectname
```

--> enter into the project

```
    cd projectname
```

--> check whether django project has successfully executed by using the command:

```
    python manage.py runserver
```

The above command generates an url which means django project has been successfully executed

Step 2: Creating multiple applications

--> command to create an application is:

```
    python manage.py startapp first_app_name
    python manage.py startapp second_app_name
    .
    .
    .
```

Step 3: Integrating applications with the project

--> register the apps in project by writing application names in list INSTALLED_APPS in settings.py module

Step 4: Writing business logic in views.py of each application

--> python code is written in views.py

--> create multiple functions in every application of project

--> for 1st application:

```
    from django.http import HttpResponse     # This generates HttpResponse

    def first_app_function1(r):
        return HttpResponse("<h1> Content </h1>")

    def first_app_function2(r):
        return HttpResponse("<h1> Content </h1>")
```

--> for 2nd application:

```
from django.http import HttpResponse     # This generates HttpResponse

def second_app_function1(r):
        return HttpResponse("<h1> Content </h1>")

def second_app_function2(r):
        return HttpResponse("<h1> Content </h1>")
```

--> for 3rd application:

```
from django.http import HttpResponse     # This generates HttpResponse

def  third_app_function1(r):
        return HttpResponse("<h1> Content </h1>")

def third_app_function2(r):
        return HttpResponse("<h1> Content </h1>")
```

Step 5: Creating url

--> in urls.py write

```
from first_app_name import views as v1
from second_app_name import views as v2
from third_app_name import views as v3
.
.
.
```

--> in urlpatterns list add

```
path('url_name/', v1.first_app_function1)
path('url_name/', v1.first_app_function2)
path('url_name/', v2.second_app_function1)
path('url_name/', v2.second_app_function2)
path('url_name/', v3.third_app_function)
path('url_name/', v3.third_app_function)
.
.
.
```

Step 6: Execution

--> runserver by using the command:
```
    python manage.py runserver
```

Creating a project with multiple applications that have multiple functions(Application level urls)

Step 1: Creating a new project

--> run the command
```
    django-admin startproject projectname
```
or
```
    python-m django startproject projectname
```
--> enter into the project
```
    cd projectname
```
--> check whether django project has successfully executed by using the command:
```
    python manage.py runserver
```
    The above command generates an url which means django project has been successfully executed

Step 2: Creating multiple applications

--> command to create an application is:
```
    python manage.py startapp first_app_name
    python manage.py startapp second_app_name
    .
    .
    .
```

Step 3: Integrating applications with the project

--> register the apps in project by writing application names in list INSTALLED_APPS in settings.py module

Step 4: Writing business logic in views.py of each application

--> python code is written in views.py
--> create multiple functions in every application of project

--> for 1st application:
```
    from django.http import HttpResponse    # This generates HttpResponse

    def first_app_function1(r):
        return HttpResponse("<h1> Content </h1>")

    def first_app_function2(r):
        return HttpResponse("<h1> Content </h1>")
```

--> for 2nd application:

```
    from django.http import HttpResponse     # This generates HttpResponse

    def second_app_function1(r):
        return HttpResponse("<h1> Content </h1>")

    def second_app_function2(r):
        return HttpResponse("<h1> Content </h1>")
```

--> for 3rd application:

```
    from django.http import HttpResponse     # This generates HttpResponse

    def  third_app_function1(r):
        return HttpResponse("<h1> Content </h1>")

    def third_app_function2(r):
        return HttpResponse("<h1> Content </h1>")
```

Step 5: Creating urls

--> copy urls.py module from project level and paste it in every application

--> In first_app_name's urls.py , write :

```
    from first_app_name import views
    In urlpatterns list, there is no need for admin path, so it can be deleted
    In urlpatterns list, write :
    path('url_name/', views.first_app_function1),
    path('url_name/', views.first_app_function2),
    .
    .
```

--> In second_app_name's urls.py , write :

```
    from second_app_name import views
    In urlpatterns list, there is no need for admin path, so it can be deleted
    In urlpatterns list, write :
    path('url_name/', views.second_app_function1),
    path('url_name/', views.second_app_function2),
    .
    .
```

--> In third_app_name's urls.py , write :

from third_app_name import views

In urlpatterns list, there is no need for admin path, so it can be deleted

In urlpatterns list, write :

path('url_name/', views.third_app_function1),

path('url_name/', views.third_app_function2),

.

.

Step 6: Linking application level url to project

--> In project level urls.py, write :

from django.urls import include

--> In urlpatterns list add

path('first_app_name/', include('first_app_name.urls')),

path('second_app_name/', include('second_app_name.urls')),

path('third_app_name/', include('third_app_name.urls')),