Django Introduction

index\images (1).jfifIntroduction

- Django is a web application framework written in Python programming language.
- It is based on MVT (Model View Template) design pattern.
- The Django is very demanding due to its rapid development feature.
- It takes less time to build application after collecting client requirement.

This framework uses a famous tag line:**The web framework for perfectionists with deadlines.**

- By using Django, we can build web applications in very less time.
- Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.

History

- Django was design and developed by Lawrence journal world in 2003 and publicly released under BSD license in July 2005.
- Currently, DSF (Django Software Foundation) maintains its development and release cycle.
- Django was released on 21, July 2005.
- Its current stable version is 2.0.3 which was released on 6 March, 2018.
- Author names are Adrian Holovaty and Simon Willison
- Django name is taken by the name of guitarist name: Django Reinhardt.

- In built light weight server software is there in django no need to install other servers.
- It supports Object Relational Mapping (ORM).
- Multilingual support (based on customer support we will provide website for understanding customer language).
- Web development with light weight web server.
- Official website for django is www.djangoproject.com

Django Version History

Version	Date	Description
0.90	16 Nov 2005	
0.91	11 Jan 2006	magic removal
0.96	23 Mar 2007	newforms, testing tools
1.0	3 Sep 2008	API stability, decoupled admin, unicode
1.1	29 Jul 2009	Aggregates, transaction based tests
1.2	17 May 2010	Multiple db connections, CSRF, model validation
1.3	23 Mar	Timezones, in browser testing, app templates.

	2011	
1.5	26 Feb 2013	Python 3 Support, configurable user model
1.6	6 Nov 2013	Dedicated to Malcolm Tredinnick, db transaction management, connection pooling.
1.7	2 Sep 2014	Migrations, application loading and configuration.
1.8 LTS	2 Sep 2014	Migrations, application loading and configuration.
1.8 LTS	1 Apr 2015	Native support for multiple template engines.*Supported until at least April 2018*
1.9	1 Dec 2015	Automatic password validation. New styling for admin interface.
1.10	1 Aug 2016	Full text search for PostgreSQL. New-style middleware.
1.11 LTS	1.11 LTS	Last version to support Python 2.7.*Supported until at least April 2020*
2.0	Dec 2017	First Python 3-only release, Simplified URL routing syntax, Mobile friendly admin.

Now current Latest version of django is 3.1.4

Popularity

Django is widely accepted and used by various well-known sites such as:

- Instagram
- Mozilla
- Disqus
- Pinterest
- Bitbucket
- The Washington Times

Features of Django

Features of Django

- Rapid Development
- Secure
- Scalable
- Fully loaded
- Versatile
- Open Source
- Vast and Supported Community

Rapid Development

- Django was designed with the intention to make a framework which takes less time to build web application.

- The project implementation phase is a very time taken but Django creates it rapidly.

Secure

- Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc.
- Its user authentication system provides a secure way to manage user accounts and passwords.

Scalable

- Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

Fully loaded

- Django includes various helping task modules and libraries which can be used to handle common Web development tasks.
- Django takes care of user authentication, content administration, site maps, RSS feeds etc.

Versatile

- Django is versatile in nature which allows it to build applications for different-different domains.
- Now a days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

Open Source

- Django is an open source web application framework.
- It is publicly available without cost.
- It can be downloaded with source code from the public repository.

- Open source reduces the total cost of the application development.

Vast and Supported Community

- Django is a one of the most popular web framework.
- It has widely supportive community and channels to share and connect.

Django Project

Go to cmd prompt and set the path of Python Scripts

To check pip version

pip --version

To upgrade pip command

python -m pip install --upgrade pip

To check django version

pip –m django –version

or

django-admin --version

To install django

pip install django

To uninstall django

pip uninstall django

To Choose the Location in your computer

D:

D:\>mkdir djangoclass

D:\>cd djangoclass

D:\djangoclass

To set the python path in this current location.

After that only you will create your project

Django Project

- To create a Django project, we can use the following command. projectname is the name of Django application.

    django-admin startproject projectname

Django Project Example

- Here, we are creating a project myFirstProject in the current directory.

    django-admin startproject myFirstProject

- After creating a project in cmd prompt and choose the location of the project .
- After downloading the atom IDE and choose the set up and click yes and go to file menu and choose settings and click install Packages like **Platformio IDE Terminal** and **atom django** also.
- After finish the above process drags and drops this project in atom IDE.

A Django project contains the following packages and files. The outer directory is just a container for the application. We can rename it further.

- **manage.py:** It is a command-line utility which allows us to interact with the project in various ways and also used to manage an application.
- A directory (myFirstProject) located inside, is the actual application package name. Its name is the Python package name which we'll need to use to import module inside the application.
- **__init__.py:** It is an empty file that tells to the Python that this directory should be considered as a Python package.
- **settings.py:** This file is used to configure application settings such as database connection, static files linking etc.
- **urls.py:** This file contains the listed URLs of the application. In this file, we can mention the URLs and corresponding actions to perform the task and display the view.
- **wsgi.py:** It is an entry-point for WSGI-compatible web servers to serve Django project.

Initially, this project is a default draft which contains all the required files and folders.

Running the Django Project

- Django project has a built-in development server which is used to run application instantly without any external web server.
- It means we don't need of Apache or another web server to run the application in development mode.

To run the application, we can use the following command.

```
python manage.py runserver
```

- Look server has started and can be accessed at localhost with port 8000.

- Go to browser window and type this url http://127.0.0.1:8000 and it will shows django server is working properly.
- 127.0.0.1->localhost or ip address of current machine
- 8000->server port number
- http://127.0.0.1:8000 is a default server url to run django web applications or you can type in this format also localhost:8000
- The application is running successfully.
- Now, we can customize it according to our requirement and can develop a customized web application.

How to change the server port number

To type this below command

python manage.py runserver portno

For ex:

python manage.py runserver 7777

Stop the server by typing ctrl+c in cmd

Django App

Django App

- Django application consists of project and app; it also generates an automatic base directory for the app, so we can focus on writing code (business logic) rather than creating app directories.

- The difference between a project and app is, a project is a collection of configuration files and apps whereas the app is a web application which is written to perform business logic.

For Example:

Bank project consists of many applications like loan application, insurance application, marketing application, customer application.

Application means which is responsible to perform particular task.

Django provides Pluggable applications.

These applications are independent and can be used any project.

Creating an App

- To create an app, we can use the following command.

  python manage.py startapp appname

  or

  django-admin startapp appname

Django App Example

  python manage.py startapp firstapp

- Once you created the firstapp,this app consists of 1 folder and 6 files
- Folder name

  migrations

- Files names are

__init__.py

admin.py

apps.py

models.py

tests.py

views.py

views.py

- Service logic will be placed in views.py
- In Python django based application where required service to enduser is defined in views.py file.
- For every action we will create views.

  2 types of views are available

  1. Function based views
  2. Class based views
- For every response we have to provide a separate function inside view.

Open views.py file and type below code

Views.py

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse

def welcome(request):
    s='<h1>Hello, Welcome to Django</h1>'
```

return HttpResponse(s)

- After finishing the views logic and add application in settings.py in project folder

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'firstapp' and put the comma and save it.

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('hello/', views.welcome),
]
```

To run the application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view action hello

http://127.0.0.1:8000/hello

- hello is an action that we are representing in urls.py in project folder to calling the particular view defining logic.

Summary of the Project and first application:

1. Creating a Project by using this command

   django-admin startproject projectname

2. Creating an application by using this command

   django-admin startapp appname

   or

   python manage.py startapp appname

3. Add Application in settings.py in the Installed_Apps
4. Creating a view functions for every action
5. Defining url patterns in project level urls.py file
6. To run the server by using this command

   python manage.py runserver

7. Send the request in browser and typing this url http://127.0.0.1:8000/hello

Creating Multiple Apps in a Project:

Creating Multiple Apps in a Project:

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

    django-admin startproject SchoolProject

2. Move to that current directory

    cd SchoolProject

3. To run the server if server is working

    python manage.py runserver

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

1. Creating an application admissions
    django-admin startapp admissions
2. Click the admissions app and select views.py

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
def addadmission(request):
    return HttpResponse('this is add admission view')
def admissionreport(request):
    return HttpResponse('this is admission report view')
```

- After finishing the views logic and add application in settings.py in project folder

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path
from admissions import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('newadm/', views.addadmission),
    path('admreport/', views.admissionreport),

]
```

Creating an Second Application

Here we are creating the first application name as finance

1. Creating an application finance
   django-admin startapp finance
2. Click the finance app and select views.py

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
```

```
def feecollection(request):
    return HttpResponse('<h1>I will collect the fees from this view</h1>')


def feeduesreport(request):
    return HttpResponse('<h1>I will get the fee dues report from this view</h1>')


def feecollectionreport(request):
    return HttpResponse('<h1>I will get fee collection report from this view</h1>')
```

- After finishing the views logic and add application in settings.py in project folder

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'finance' and put the comma and save it.

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path
from finance import views

urlpatterns = [
```

```
    path('admin/', admin.site.urls),
    path('feecoll/', views.feecollection),
    path('duesreport/', views.feeduesreport),
    path('collectionreport/', views.feecollectionreport),


]
```

To run the application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/

After running this server it will shows error in finance application doesn't have addadmission view function.

If we are used to creating a multiple applications in a single project and the first application url_Patterns view are importing in the urls.py file will be override by the second application url_Patterns view are importing in the urls.py file.

Aliasing Concept:

To change the urls.py file in the project folder by adding the aliasing concept for importing the admissions and finance app.

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path
from admissions import views as ad
from finance import views as fin

urlpatterns = [
    path('admin/', admin.site.urls),
    path('newadm/', ad.addadmission),
    path('admreport/', ad.admissionreport),
    path('feecoll/', fin.feecollection),
    path('duesreport/', fin.feeduesreport),
    path('collectionreport/', fin.feecollectionreport),


]
```

To run the application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/

Another approach to overcome ambiguity

If we have same view name in admissions and finance apps then it will become again clash so views name should be unique.

urls.py

```
from django.contrib import admin
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport
from finance.views import feecollection
from finance.views import feeduesreport
from finance.views import feecolletionreport

urlpatterns = [
    path('admin/', admin.site.urls),
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
    path('feecoll/', feecollection),
    path('duesreport/', feeduesreport),
    path('collectionreport/', feecollectionreport),


]
```

Summary of multiple apps in a project:

8. Creating a Project by using this command

   django-admin startproject projectname

9. Creating an first application by using this command

   django-admin startapp appname

   or

   python manage.py startapp appname

10. Add First Application in settings.py in the Installed_Apps

11. Creating a first application view functions for every action
12. Defining first application url patterns in project level urls.py file
13. Creating an second application by using this command

    django-admin startapp appname

    or

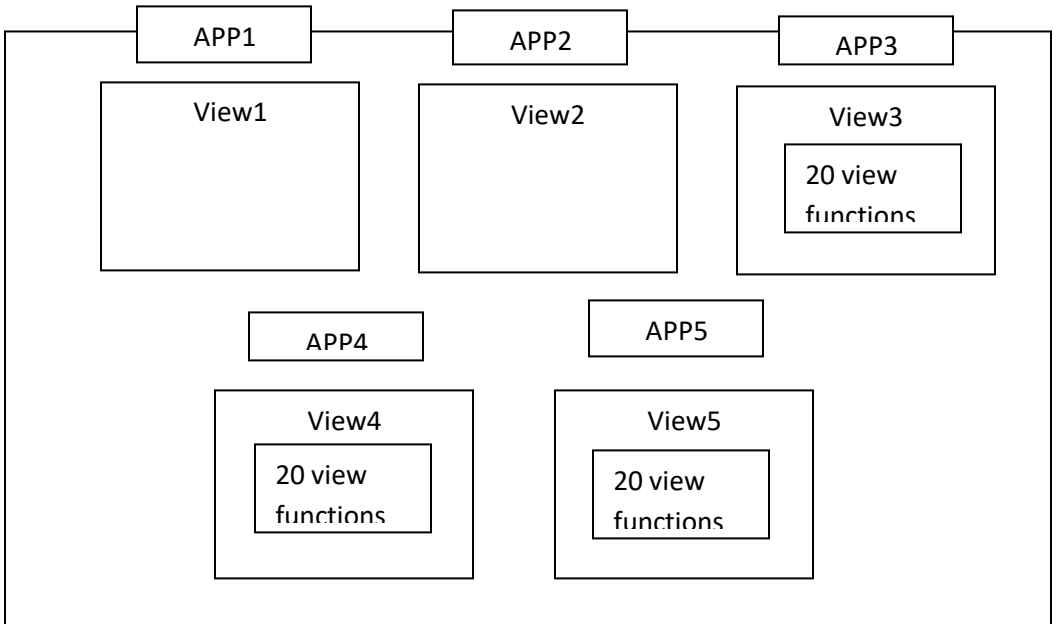
    python manage.py startapp appname

14. Add Second Application in settings.py in the Installed_Apps
15. Creating a second application view functions for every action
16. Defining second application url patterns in project level urls.py file
17. To run the server by using this command

    python manage.py runserver

18. Send the request in browser and typing this url http://127.0.0.1:8000/

Applications wise URL Mappings (or) Application Level URL:



For Example:

Every application contains 20 views function.

1. When we open 100 url patterns will be there at the time we will get confused which url pattern is corresponding to which view.

2. Readability will be reduced in url.

3. To use application in some other project we required all the url patterns related to that application have to copy in other applications.So, defining url at project level reduces reusability of the application.

4. To overcome about disadvantage maintain separate url files at application level instead of project level.

5. Django default url support is always project level.

6. For each and every application we have to create urls.py file.

7. At project level urls.py we have to tell to project urls.py go and check the application level.

8. Django project can contain multiple applications and each application can contain multiple views.

9. Defining url patterns for all views of all applications inside urls.py file of project creates maintenance problem and reduces reusability of applications.

10. We can solve this problem by defining url patterns at application level instead of project level.

11. For every application we have to create a separate urls.py file and we have to define all that application specific urls in that file.

12. We have to link this application level in urls.py file to project level urls.py file by using include ( ) method.

Creating Multiple Apps in a Project:

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

```
django-admin startproject SchoolProject
```

2. Move to that current directory

```
cd SchoolProject
```

3. To run the server if server is working

```
python manage.py runserver
```

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

3. Creating an application admissions

```
django-admin startapp admissions
```

4. Click the admissions app and select views.py

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
def addadmission(request):
    return HttpResponse('this is add admission view')
def admissionreport(request):
    return HttpResponse('this is admission report view')
```

- After finishing the views logic and add application in settings.py in project folder

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
]
```

Creating an Second Application

Here we are creating the first application name as finance

3. Creating an application finance
   django-admin startapp finance
4. Click the finance app and select views.py

views.py

```
from django.shortcuts import render
from django.http import HttpResponse


def feecollection(request):

    return HttpResponse('<h1>I will collect the fees from this view</h1>')


def feeduesreport(request):

    return HttpResponse('<h1>I will get the fee dues report from this view</h1>')


def feecollectionreport(request):

    return HttpResponse('<h1>I will get fee collection report from this view</h1>')
```

- After finishing the views logic and add application in settings.py in project folder

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'finance' and put the comma and save it.

Creating urls.py file in finance application

Right click on finance app and click new file and type urls.py file

Open urls.py in the finance app

urls.py

```
from django.urls import path
from finance.views import feecollection
from finance.views import feeduesreport
from finance.views import feecollectionreport


urlpatterns = [
    path('feecoll/', feecollection),
    path('duesreport/', feeduesreport),
    path('collectionreport/', feecollectionreport),

]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('ad/', include('admissions.urls')),
    path('fin/', include('finance.urls')),

]
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/ad/newadm

Making urls at project level

1. Create project

2. Create applications

3. Add applications in settings.py file

4. Define Views in applications

5. Create urls at project level->click applications on that right click and create urls.py file.

6. Define urls patterns at application level

7. Go to project level urls.py file and define include function.

8. Run the Server

9. Send the request http://127.0.0.1:8000/ad/newadm/

Advantage:

The main advantage of defining url patterns at application level instead of project level

1. If promotes reusability of django applications across multiple projects.

2. Project level urls.py file will be clean and more readable.

Creating Django Templates

Creating Django Templates:

It is not recommended to write an Html code inside Python Scripts (views.py file).

1.It reduces the readability because python code mixed with Html code.

2.No separation of roles python developer has to concentrate on both python code and html code.

3. It doesn't promote reusability of code. So, we can overcome these problems by separating html code into the separate html file.

4. This html file is nothing but template from the python file (views.py file).We can use these templates based on our requirement. So, we have to write templates at project level at only once and we can use this in multiple applications.

Creating a Project:

Here we are developing some School Project

1.Creating a projectSchoolProject in the current directory.

django-admin startproject SchoolProject

2. Move to that current directory

cdSchoolProject

3. To run the server if server is working

python manage.py runserver

4. Go to browser window and type this urlhttp://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

5. Creating an application admissions
   django-adminstartapp admissions

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

Creating an Second Application

Here we are creating the first application name as finance

5. Creating an application finance
   django-adminstartapp finance

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'finance' and put the comma and save it.

1. Go to main folder of SchoolProject and create new folder name templates.
2. Click templates and create new folder and type admissions.
3. Click templates and create new folder and type finance.
4. Click an admissions folder and create new file add addadmission.html
5. Click an admissions folder and create new file add admissionreport.html
6. Click an finance folder and create new file add feecollection.html
7. Click an finance folder and create new file add feeduesreport.html
8. Click an finance folder and create new file add feecollectionreport.html

addadmission.html

```
<html>

<head>

<title>add admission page</title>

</head>

<body>

<h1>Hello this is add admission page</h1>

<h3>Please fill below form</h3>

</body>
```

```
</html>
```

admissionreport.html

```
<html>
<head>
<title>admission  report page</title>
</head>
<body>
<h1>Hello this is admission report page</h1>
<h3>Please fill below form</h3>
</body>
</html>
```

feecollection.html

```
<html>
<head>
<title>fee collection page</title>
</head>
<body>
<h1>Hello this is fee collection page</h1>
<h3>Please fill below form</h3>
</body>
</html>
```

feeduesreport.html

```
<html>
<head>
<title>fee dues report page</title>
</head>
<body>
<h1>Hello this is fee dues report page</h1>
<h3>Please fill below form</h3>
</body>
</html>
```

feecollectionreport.html

```
<html>
<head>
<title>fee collection report page</title>
</head>
<body>
<h1>Hello this is fee collection report page</h1>
<h3>Please fill below form</h3>
</body>
</html>
```

Go to admissions app and click views.py file

render( ):

- This function uses the render() function to create the HttpResponse that is sent back to the browser.
- This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context").

views.py

```
fromdjango.shortcuts import render

defaddadmission(request):

    return render(request,'admissions/addadmission.html')

defadmissionreport(request):

    return render(request,'admissions/admissionreport.html')
```

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
   path('newadm/', addadmission),
```

```
    path('admreport/', admissionreport),
]
```

Go to finance app and click views.py file

views.py

```
fromdjango.shortcuts import render

deffeecollection(request):
        return render(request,'finance/feecollection.html')

deffeeduesreport(request):
        return render(request,'finance/feeduesreport.html')

deffeecollectionreport(request):
        return render(request,'finance/feecollectionreport.html')
```

Creating urls.py file in finance application

Right click on finance app and click new file and type urls.py file

Open urls.py in the finance app

urls.py

```
from django.urls import path
from finance.views import feecollection
from finance.views import feeduesreport
from finance.views import feecollectionreport


urlpatterns = [
    path('feecoll/', feecollection),
    path('duesreport/', feeduesreport),
```

```
    path('collectionreport/', feecollectionreport),

]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
path('ad/', include('admissions.urls')),
path('fin/', include('finance.urls')),

]
```

To know the current python filename

```
print (__file__) # for example: template.py
```

To know the absolute path of current python file name

```
print(os.path.abspath(__file__))
```

forexample:d:\djangoclass\python template.py

To know base directory name of the current file

```
print(os.path.dirname(os.path.abspath(__file__)))
```

Open the schoolproject and click the settings.py file and go to the templates and write the below command.

settings.py file

importos

TEMPLATES=[

{

'DIRS':[os.path.join(BASE_DIR,"templates")],

}

]

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/newadm

Steps to develop Template based applications:

1. Create a Project
2. Creation of Applications
3. Add these applications to the project in settings.py file.So, that django aware of applications.
4. Create atemplate folder inside main project in that templates older create a separate folder named with appname (admissions,finance) to hold that particular application specific templates.
5. Add templates folder path to settings.py file.So,thatdjango can aware of our templates.

TEMPLATES=[

```
{
        'DIRS'=['D:\djangoclass\SchoolProject']
}
```

It is not recommended to hardcore system specific locations in settings.py file to overcome this problem we can generate templates directory path programmatically as follows.

```
importos
BASE_DIR=os.path.dirname(os.path.dirname(os.path.abspath(__file__)
))

TEMPLATES=[
{
'DIRS'=[os.path.join(BASE_DIR,'templates')],
}
]
```

Sending the Data from view to Templates

Sending the Data from view to Templates

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

```
django-admin startproject SchoolProject
```

2. Move to that current directory

```
cd SchoolProject
```

3. To run the server if server is working

```
python manage.py runserver
```

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

6. Creating an application admissions

```
django-admin startapp admissions
```

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

9. Go to main folder of SchoolProject and create new folder name templates.
10. Click templates and create new folder and type admissions.
11. Click an admissions folder and create new file add addadmission.html

12. Click an admissions folder and create new file add admissionreport.html

Go to admissions app and click views.py file

render ( ):

- This function uses the render ( ) function to create the HttpResponse that is sent back to the browser.
- This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context").

views.py

```
from django.shortcuts import render

def addadmission(request):

    values={ "name":"sam","age":22,"address":"chennai"}

    return render(request, 'admissions/addadmission.html',values)

def admissionreport(request):

    return render(request, 'admissions/admissionreport.html')
```

addadmission.html

```
<html>

<head>

<title>add admission page</title>

</head>

<body>
```

```
<h1>Hello {{name}}…this is add admission page</h1>
<h2> My age is {{age}}</h2>
<h2>My address is {{address}}</h2>
<h3>Please fill below form</h3>
</body>
</html>
```

admissionreport.html

```
<html>
<head>
<title>admission  report page</title>
</head>
<body>
<h1>Hello this is admission report page</h1>
<h3>Please fill below form</h3>
</body>
</html>
```

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('ad/', include('admissions.urls')),

]
```

To know the current python filename

print (__file__) # for example: template.py

To know the absolute path of current python file name

print(os.path.abspath(__file__))

for example:d:\djangoclass\python template.py

To know base directory name of the current file

print(os.path.dirname(os.path.abspath(__file__)))

Open the SchoolProject and click the settings.py file and go to the templates and write the below command.

settings.py file

```
import os

TEMPLATES=[

{

'DIRS':[os.path.join(BASE_DIR,"templates")],

}

]
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/ad/newadm

Template Tags:

From python views.py we can inject dynamic content to the template file by using template tags.

Template tags are also known as Template Variables.

So, it is not a python syntax and it not an html syntax {{}} and just it is a special syntax.

Note: The values to the template variables should be passed from the view in the form of dictionary as arguments to context.

Inserting Images into Templates

Inserting Images into Templates:

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

    django-admin startproject SchoolProject

2. Move to that current directory

    cd SchoolProject

3. To run the server if server is working

    python manage.py runserver

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

7. Creating an application admissions

    django-admin startapp admissions

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

13. Go to main folder of SchoolProject and create new folder name templates.
14. Click templates folder and create new folder and type admissions.
15. Click an admissions folder and create new file add addadmission.html
16. Click an admissions folder and create new file add admissionreport.html
17. Go to application folder of admissions app and create new folder name static.
18. Click static folder and create new folder and type images.
19. Select an any image in your computer and copy that image and paste into the images folder.

Go to admissions app and click views.py file

render ( ):

- This function uses the render ( ) function to create the HttpResponse that is sent back to the browser.
- This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context").

views.py

from django.shortcuts import render

```
def addadmission(request):
    values={ "name":"sam","age":22,"address":"chennai"}
    return render(request, 'admissions/addadmission.html',values)
def admissionreport(request):
    return render(request, 'admissions/admissionreport.html')
```

addadmission.html

```
{% load static %}
<html>
<head>
<title>add admission page</title>
</head>
<body>
<h1>Hello {{name}}…this is add admission page</h1>
<img src="{% static 'images/Desert.jpg' %}" alt="Desert"
height="300px" width="700px"/>
<h2> My age is {{age}}</h2>
<h2>My address is {{address}}</h2>
<h3>Please fill below form</h3>
</body>
</html>
```

admissionreport.html

```
<html>

<head>

<title>admission  report page</title>

</head>

<body>

<h1>Hello this is admission report page</h1>

<h3>Please fill below form</h3>

</body>

</html>
```

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('ad/', include('admissions.urls')),

]
```

To know the current python filename

```
print (__file__) # for example: template.py
```

To know the absolute path of current python file name

```
print(os.path.abspath(__file__))
```

for example:d:\djangoclass\python template.py

To know base directory name of the current file

```
print(os.path.dirname(os.path.abspath(__file__)))
```

Open the SchoolProject and click the settings.py file and go to the templates and write the below command.

settings.py file

```
import os

TEMPLATES=[

{
```

'DIRS':[os.path.join(BASE_DIR,"templates")],

}

]

STATIC_URL= '/static/'

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/newadm

Linking Templates Stylesheets in Django

Linking Templates Stylesheets in Django:

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

django-admin startproject SchoolProject

2. Move to that current directory

cd SchoolProject

3. To run the server if server is working

python manage.py runserver

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

8. Creating an application admissions

django-admin startapp admissions

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

20. Go to main folder of SchoolProject and create new folder name templates.
21. Click templates folder and create new folder and type admissions.
22. Click an admissions folder and create new file add addadmission.html
23. Click an admissions folder and create new file add admissionreport.html
24. Go to application folder of admissions app and create new folder name static.
25. Click static folder and create new folder and type stylesheets.
26. Click stylesheets folder and create new file style.css

Go to admissions app and click views.py file

render ( ):

- This function uses the render ( ) function to create the HttpResponse that is sent back to the browser.
- This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context").

views.py

```
from django.shortcuts import render

def addadmission(request):

    values={ "name":"sam","age":22,"address":"chennai"}

    return render(request, 'admissions/addadmission.html',values)

def admissionreport(request):

    return render(request, 'admissions/admissionreport.html')
```

addadmission.html

```
{% load static %}

<html>

<head>

<title>add admission page</title>

<link rel="stylesheet" href="{% static 'stylesheets/style.css' %}"/>

</head>

<body>
```

```
<h1>Hello {{name}}…this is add admission page</h1>
<h2 style= "color:red;"> My age is {{age}}</h2>
<h2>My address is {{address}}</h2>
<h3>Please fill below form</h3>
</body>
</html>
```

Open style.css file in stylesheets folder

style.css

```
body
{
	background-color:skyblue;
}
h1
{
	color:blue;
}
```

admissionreport.html

```
<html>
<head>
<title>admission  report page</title>
</head>
```

```
<body>

<h1>Hello this is admission report page</h1>

<h3>Please fill below form</h3>

</body>

</html>
```

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('ad/', include('admissions.urls')),
```

]

To know the current python filename

print (__file__) # for example: template.py

To know the absolute path of current python file name

print(os.path.abspath(__file__))

for example:d:\djangoclass\python template.py

To know base directory name of the current file

print(os.path.dirname(os.path.abspath(__file__)))

Open the SchoolProject and click the settings.py file and go to the templates and write the below command.

settings.py file

import os

TEMPLATES=[

{

'DIRS':[os.path.join(BASE_DIR,"templates")],

}

]

STATIC_URL= '/static/'

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action http://127.0.0.1:8000/ad/newadm

Maintaining stylesheet externally is good programming practice and also maintenance will become easy.

Stylesheets are 3 types

1. Inline ->It is applicable for particular tag

2. Internal-> It is applicable for particular page.

3. External-> It is applicable for multiple pages.

Where we have to place the stylesheets in the project and how to link with templates and apply.

static (folder)->stylesheets(folder)->style.css(file)

Inline Stylesheet

<h1 style= "color:red;">Hello</h1>

External Stylesheet

To link external stylesheet we have to write a link in head tag.

<link rel="stylesheet" href="{% static 'stylesheets/style.css' %}"/>

Inserting Javascript Code into Templates:

Inserting Javascript Code into Templates:

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

```
django-admin startproject SchoolProject
```

2. Move to that current directory

```
cd SchoolProject
```

3. To run the server if server is working

```
python manage.py runserver
```

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

9. Creating an application admissions

```
django-admin startapp admissions
```

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

27. Go to main folder of SchoolProject and create new folder name templates.
28. Click templates folder and create new folder and type admissions.
29. Click an admissions folder and create new file add addadmission.html
30. Click an admissions folder and create new file add admissionreport.html
31. Go to application folder of admissions app and create new folder name static.
32. Click static folder and create new folder and type js.
33. Click js folder and create new file script.js

Go to admissions app and click views.py file

render ( ):

- This function uses the render ( ) function to create the HttpResponse that is sent back to the browser.
- This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context").

views.py

```
from django.shortcuts import render

def addadmission(request):

    values={ "name":"sam","age":22,"address":"chennai"}
```

```
    return render(request, 'admissions/addadmission.html',values)
def admissionreport(request):
    return render(request, 'admissions/admissionreport.html')
```

addadmission.html

```
{% load static %}
<html>
<head>
<title>add admission page</title>
</head>
<body>
<script src="{% static '/js/script.js' %}" type="text/javascript"></script>
 <h1>Hello {{name}}…this is add admission page</h1>
<h2 style= "color:red;"> My age is {{age}}</h2>
<h2>My address is {{address}}</h2>
<h3>Please fill below form</h3>
</body>
</html>
```

Open style.css file in stylesheets folder

script.js

```
alert("Hello, Welcome to Django");
```

admissionreport.html

```
<html>

<head>

<title>admission  report page</title>

</head>

<body>

<h1>Hello this is admission report page</h1>

<h3>Please fill below form</h3>

</body>

</html>
```

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('ad/', include('admissions.urls')),

]
```

To know the current python filename

```
print (__file__) # for example: template.py
```

To know the absolute path of current python file name

```
print(os.path.abspath(__file__))
```

for example:d:\djangoclass\python template.py

To know base directory name of the current file

```
print(os.path.dirname(os.path.abspath(__file__)))
```

Open the SchoolProject and click the settings.py file and go to the templates and write the below command.

settings.py file

```
import os

TEMPLATES=[

{

'DIRS':[os.path.join(BASE_DIR,"templates")],
```

}

]

STATIC_URL= '/static/'

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/newadm

Setting Home Page

Setting Home Page

Creating a Project:

Here we are developing some School Project

1. Creating a project SchoolProject in the current directory.

django-admin startproject SchoolProject

2. Move to that current directory

cd SchoolProject

3. To run the server if server is working

python manage.py runserver

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an First Application

Here we are creating the first application name as admissions

10. Creating an application admissions
    django-admin startapp admissions

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'admissions' and put the comma and save it.

Creating an Second Application

Here we are creating the first application name as finance

6. Creating an application finance
    django-admin startapp finance

Settings.py

- In this settings.py file and go to the Installed_Apps and add the application 'finance' and put the comma and save it.

34. Go to main folder of SchoolProject and create new folder name templates.
35. Click templates and create new folder and type admissions.
36. Click templates and create new folder and type finance.

37. Click an admissions folder and create new file add addadmission.html
38. Click an admissions folder and create new file add admissionreport.html
39. Click an finance folder and create new file add feecollection.html
40. Click an finance folder and create new file add feeduesreport.html
41. Click an finance folder and create new file add feecollectionreport.html
42. Go to application folder of admissions app and create new folder name static.
43. Click static folder and create new folder and type images.
44. Select an any image in your computer and copy that image and paste into the images folder.
45. Click static folder and create new folder and type stylesheets.
46. Click stylesheets folder and create new file style.css
47. Click templates folder and create new file name index.html

Go to admissions app and click views.py file

render ( ):

- This function uses the render ( ) function to create the HttpResponse that is sent back to the browser.
- This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context").

views.py

from django.shortcuts import render

```
def homepage(request):
    return render(request,'index.html')
def addadmission(request):
    values={ "name":"sam","age":22,"address":"chennai"}
    return render(request, 'admissions/addadmission.html',values)
def admissionreport(request):
    return render(request, 'admissions/admissionreport.html')
```

addadmission.html

```
{% load static %}
<html>
<head>
<title>add admission page</title>
<link rel="stylesheet" href="{% static 'stylesheets/style.css' %}"/>
</head>
<body>
<img src="{% static 'images/Desert.jpg' %}" alt="Desert"
height="300px" width="700px"/>
<h1>Hello {{name}}…this is add admission page</h1>
<h2 style= "color:red;"> Your age is {{age}}</h2>
<h2>My address is {{address}}</h2>
</body>
```

```
</html>
```

Open style.css file in stylesheets folder

style.css

```
body
{
	background-color:skyblue;
}
h1
{
	color:blue;
}
```

admissionreport.html

```
<html>
<head>
<title>admission  report page</title>
</head>
<body>
<h1>Hello this is admission report page</h1>
</body>
</html>
```

feecollection.html

```
<html>
<head>
<title>fee collection page</title>
</head>
<body>
<h1>Hello this is fee collection page</h1>
</body>
</html>
```

feeduesreport.html

```
<html>
<head>
<title>fee dues report page</title>
</head>
<body>
<h1>Hello this is fee dues report page</h1>
</body>
</html>
```

feecollectionreport.html

```
<html>
```

```
<head>
<title>fee collection report page</title>
</head>
<body>
<h1>Hello this is fee collection report page</h1>
</body>
</html>
```

Creating urls.py file in admission application

Right click on admissions app and click new file and type urls.py file

Open urls.py in the admissions app

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport

urlpatterns = [
    path('newadm/', addadmission),
    path('admreport/', admissionreport),
]
```

Go to finance app and click views.py file

views.py

```
from django.shortcuts import render
```

```
def feecollection(request):
    return render(request, 'finance/feecollection.html')
def feeduesreport(request):
    return render(request, 'finance/feeduesreport.html')
def feecollectionreport(request):
    return render(request, 'finance/feecollectionreport.html')
```

Creating urls.py file in finance application

Right click on finance app and click new file and type urls.py file

Open urls.py in the finance app

urls.py

```
from django.urls import path
from finance.views import feecollection
from finance.views import feeduesreport
from finance.views import feecollectionreport


urlpatterns = [
    path('feecoll/', feecollection),
    path('duesreport/', feeduesreport),
    path('collectionreport/', feecollectionreport),

]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
```

```
from django.urls import path,include
from admissions.views import homepage

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',homepage),
    path('ad/', include('admissions.urls')),
    path('fin/', include('finance.urls')),

]
```

index.html

```
{% load static %}

<html>

<head>

<title>add admission page</title>

<link rel="stylesheet" href="{% static 'stylesheets/style.css' %}"/>

</head>

<body>

<h1>School Project</h1>

<div class="module">

<h1>Admissions</h1>

<a href="./ad/newadm">Add Admissions</a>

<a href="./ad/admreport"><button type="button"
name="button">Admission Report</button></a>
```

```
</div>

<div class="module1">

<h1>Finance</h1>

<a href="./fin/feecoll"><button type="button" name="button">Fee Collection</button></a>

<a href="./fin/duesreport"><button type="button" name="button">Fee Dues Report</button></a>

<a href="./fin/collectionreport"><button type="button" name="button">Fee Collection Report</button></a>

</div>

</body>

</html>
```

To know the current python filename

print (__file__) # for example: template.py

To know the absolute path of current python file name

print(os.path.abspath(__file__))

for example:d:\djangoclass\python template.py

To know base directory name of the current file

print(os.path.dirname(os.path.abspath(__file__)))

Open the SchoolProject and click the settings.py file and go to the templates and write the below command.

settings.py file

```
import os
TEMPLATES=[
{
'DIRS':[os.path.join(BASE_DIR,"templates")],
}
]
STATIC_URL= '/static/'
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

http://127.0.0.1:8000

Database Management System

Database Management System

Database: It is system which allows storing data.

Operating database is called database management system.

RDBMS: Relational Database Management System.

It is used to store data as table format.

SQL: Structure Query Language

All database will follow common structure standard is called SQL.

Crud Operations:

1) Create: Creating table and inserting data->create,insert
2) Read: Reading our extracting data from table->select
3) Update: Updating existing data into table->update
4) Delete: Deleting data from table->delete

Database Creation:

Syn: create database databasename;

Ex: create database School;

To check Databases:

Syn: show databases;

To Select Database:

Syn: use databasename;

Ex: use school;

To create Table:

Syn: create table tablename(column datatype,column datatype,…)

Ex: create table student(rollno int,name varchar(50),address varachar(50));

To insert data into table:

Syn: insert into tablename values(value1,value2,…)

Ex: insert into student values(1, 'sam','chennai');

To Select data into table:

Syn: select * from tablename;

Ex: select * from student;

To Select Particular record into the table:

Syn: select * from tablename where condition;

Ex: select * from student where rollno=1;

To Update Record in table:

Syn: update tablename set columnname=value where columnname=value;

Ex: update student set address=address='hyd' where rollno=1;

To Delete Record in Table:

Syn: delete from tablename where condition;

Ex: delete from student where rollno=1;

Introduction to model

Introduction to model:

- In MVT model is related to database. So, here we don't need much queries. This model will create queries.
- This is the advantage of django.

- For each and every table we will create the model in database.
- When we create this model to the table. This model creates a queries related to that table.
- In other technologies, we have to write the queries to insert, retrieve (select) the table.
- But in django, we don't need to write queries. Each and every table resent in database we will create models.

    Ex: students, marks etc.

- To represent a table in django also called as Models.
- When we create a class in models that user fields present in that class to create table columns automatically.

Where we will create models:

At application level models.py file in this file we will create a model.

models.py

```
class modelname(models.Model):
    fields
```

After creating models we have to run 2 commands

1) makemigration

2) migrate

makemigrations:

- It prepares the SQL Queries from defined models and it also prepares sufficient background to execute SQL Queries.

migrate:

- It executes the queries and also it will create admin,auth,sessions and tables.

Creating Models in Django

Creating Models in Django:

- SQL Lite presents in django which database we can use SQL Lite for medium scale applications.
- But in real time projects we can't use because it is light weight SQL Lite.

Example for creating a table in SQL Query:

create table student(name varchar(100),fathername varchar(100),classname int,contact varchar(30));

Steps to create Models:

1. Check database configurations in settings.py file in project folder.

2. Open models.py file in application folder at application level and define models (create a class for each model).

```
from django.db import models

class student(models.Model):
```

3. Attributes we have to define in that model

```
from django.db import models
```

```
class student(models.Model):
    name = models.CharField(max_length=100)
    fathername = models.CharField(max_length=100)
    classname = models.IntegerField()
    contact = models.CharField(max_length=100)
```

Basic Field Types:

1. AutoField ->Automatic increment used for integers
2. BigAutoField -> Big Integers
3. Boolean Field -> true or false
4. CharField -> String
5. DateField -> Date
6. DecimalField -> Double
7. DurationField -> Timestamp
8. FloatField -> Float
9. IntegerField -> Int
10. TextField -> Text
11. TimeField -> Time

4. Run 2 commands

makemigrations

migrate

makemigrations command:

python manage.py makemigrations

- When we execute that command 0001_initial.py file will be created.
- To see what happen in the SQL point of view in the background of python manage.py makemigrations the command is

  python manage.py sqlmigrate admissions 0001

5) migrate command:

python manage.py migrate

- admin,admissions,auth,contenttypes,sessions all files related to them will create by using above command.

All the Queries executes in database. This database we will set in settings.py file.

To find how makemigrations and migrate have run:

python manage.py shell

It will open the interactive console shell

- we have imported connection to connect databases(SQL Lite,MySQL)

  from django.db import connection

  cur=connection.cursor() #creating cursor object

  cur.execute("select * from admissions_student")

  res=cur.fetchall()

  print(res)

After typing this above code it will shows the output in the list format

o/p:[]

To exit the interactive console shell **ctrl+z**

How to use models into views

How to use models into views:

- Already we have created a model student (table)
- View send the request to model and model will response to views by creating an executing the queries and send the data to view.
- From that view the data sent to template and gives response to the output (user).

Open admissions app and clickviews.py

Views.py

```
from admissions.models import student

def admissionreport(request):

    result=student.objects.all();

    students={ 'allstudents':result}

    return render(request, 'admissions/admissionreport.html',students)
```

admissionreport.html

```
<html>
```

```
<head>
<title>Admission Report Page</title>
</head>
<body>
{% for s in allstudents %}
{{s.name}}--{{s.fathername}}
{% endfor %}
</body>
</html>
```

Or

admissionreport.html

```
<html>
<head>
<title>Admission Report Page</title>
</head>
<body>
<table border="1">
<tr>
<th>Student Name</th>
```

```
<th>Father Name</th>
<th>Class Name</th>
<th>contact</th>
</tr>
{% for s in allstudents %}
<tr>
<td>{{s.name}}</td>
<td>{{s.fathername}}</td>
<td>{{s.classname}}</td>
<td>{{s.contact}}</td>
</tr>
{% endfor %}
</table>
</body>
</html>
```

Before using models

Views	Templates
values={"name":"abc","fathername":"sam","classname":1,"contact":"8956127452"}	{{name}} {{fathername}} {{classname}}

	{{contact}}
students:{ 'allstudents':result}	{% for s in allstudents %} {{s.name}} {{s.fathername}} {% endfor %}
result->{s1,s2,s3} i.e,s1,s2,s3 is the record of the student table	

- Go to Project location in cmd prompt and type this command

  python manage.py shell

```
from django.db import connection

cur=connection.cursor()

cur.execute(("insert into admissions_student
values(1,'abc','sam',4,'8956127452')"))
```

- After type this exit the interactive console shell using **ctrl+z**

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000

Django MySQL Connectivity

Django MySQL Connectivity:

settings.py

```
DATABASES={
    'default':{
        'ENGINE':'django.db.backends.mysql',
        'NAME':'schoolapp',
        'USER':'root',
        'PASSWORD':'',
        'HOST':'localhost',
        'PORT':'3306',
    }
}
```

Note:

- For host and port no need to give why because mysql database also in the same system.

- If the database is not in our system and the company will be maintained server in another system we will provide host and port.

Open mysql command prompt

Set the path for mysql

path C:\xampp\mysql\bin

mysql –u root –p

enter password:

show databases;

create database schoolapp;

Open python command prompt

To check connection is successful or not

Go to project location and run this below command

python manage.py shell

It opens interactive console shell and type below code

from django.db import connection

cur=connection.cursor()

After mysql 8 version error: caching_sha2_password

Note:

- Here we don’t give password directly. We have to encrypt that password for security purpose here we are not using root user password.

This is applicable for after mysql 8 version

Open mysql command prompt

Create our own user

Create user 'schoolappuser'@'localhost' identified with mysql_native_password by 'passwordname';

grant all school.* to 'schoolapp'@'localhost';

- Here schoolapp is the database name
- schoolappuser is the user name

use schoolapp;

show tables;

Open python command prompt:

- Go to project location and run this below command

  python manage.py shell

It will shows any error like to install mysqlclient and install that package based on your python version.

Here I am using python 3.8 I would like to install this package

Before that I have to download this package and placed to project main location of django.

To run this below command and install mysqlclient package

pip install mysqlclient-1.4.6-cp38-cp38-win32.whl

- It will opens interactive console shell and type the below code to check connection is working or not.

  from django.db import connection

  cur=connection.cursor()

- If the connection is properly working means and it will successfully connected to mysql database.
- To exit the interactive console shell and type ctrl+z

Run makemigrations command

python manage.py makemigrations

Run migrate command

python manage.py migrate

Open mysql command prompt

show tables;

desc admissions_student;

insert into admissions_student values(1,'abc','def',4,'7854695412');

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

http://127.0.0.1:8000

Django Admin Interface

Django Admin Interface

- Django provides a built-in admin module which can be used to perform CRUD operations on the models.
- It reads metadata from the model to provide a quick interface where the user can manage the content of the application.
- This is a built-in module and designed to perform admin related tasks to the user.

Let's see how to activate and use Django's admin module (interface).

- The admin app **(django.contrib.admin)** is enabled by default and already added into INSTALLED_APPS section of the settings file.
- To access it at browser use '/**admin**/' at a local machine like **localhost:8000**/**admin**/ and it shows the following output:

Create an Admin User

Go to project location and type below command

python manage.py createsuperuser

It will ask the below details

Username:

Email:

Password:

Password (again):

Note: The password must consists of 8 characters

- After entering the details and run the server

python manage.py runserver

Go to browser and open the 127.0.0.1:8000/admin

- It will shows admin login and type the username and password that are created by using the super user
- After login into admin interface and it will shows the groups and users.

Register Django model:

Go to admisssions app and select admin.py file

admin.py

from admissions.models import student

admin.site.register(student)

- Go to that admin and see the student model will added and see the student objects (records) and you have to update, delete the data into the student model and it will also changed into the database.

Django ORM(Object Relational Mapping)

Django ORM(Object Relational Mapping)

- One of the most powerful features of Django is its Object-Relational Mapper (ORM), which enables you to interact with your database, like you would with SQL.
- In fact, Django's ORM is just a pythonical way to create SQL to query and manipulate your database and get results in a pythonic fashion.

Go to project location and type the below command

python manage.py shell

It will opens interactive console

Retrieving Data in SQL

Select * from admissions_student;

ORM

from admissions.models import student

s=student.objects.all()

print(s)

To select particular id in SQL

select * from admissions_student where id=3;

ORM

s=student.objects.get(id=3)

```
print(s)

s=student.objects.get(name="abc")

print(s)
```

To select the particular columns in SQL

```
select id,name from admissions_student;
```

Selective Columns in ORM (values_list or values or only)

values_list

```
s=student.objects.values_list('name', 'contact')  #output will be displayed in tuple format

for s1 in s:

    print(s1)
```

values

```
s=student.objects.values('name', 'contact')  #output will be displayed in dictionary format

for s1 in s:

    print(s1)
```

only

```
s=student.objects.only('name', 'contact')  #return rows as objects and adds id field.

for s1 in s:

    print(s1)
```

To select the records based on condition using SQL

Select id,name,contact from admissions_student where id>3;

ORM

Greater than condition

```
s=student.objects.filter(id__gt=3)
print(s)
```

Greater than equal to condition

```
s=student.objects.filter(id__gte=3)
print(s)
```

To select the records into the table by selecting the middle character using LIKE Operator SQL

select * from admissions_student where name LIKE '%bc%'

ORM

```
s=student.objects.filter(name__contains= 'bc')
for s1 in s:
    print(s1.name)
```

Here **contains** is **case sensitive** it will checks the record into the table must be in same case only

```
s=student.objects.filter(name__icontains= 'bc')
```

```
for s1 in s:
    print(s1.name)
```

Here **icontains** is **case insensitive** it will checks the record into the table must be same

To select the records into the table by selecting the first character using LIKE Operator SQL

```
select * from admissions_student where name LIKE  'b%'
```

ORM

```
s=student.objects.filter(name__startswith= 'b')
for s1 in s:
    print(s1.name)
```

To select the records into the table by selecting the first character using LIKE Operator SQL

```
select * from admissions_student where name LIKE  'b%'
```

ORM

```
s=student.objects.filter(name__startswith= 'b')
for s1 in s:
    print(s1.name)
```

To select the records into the table by selecting the lastcharacter using LIKE Operator SQL

```
select * from admissions_student where name LIKE  '%c'
```

ORM

```
s=student.objects.filter(name__endswith= 'c')
for s1 in s:
    print(s1.name)
```

In Operator in SQL

```
select * from admissions_student where id in(1,2,5);
```

ORM

```
s=student.objects.filter(id__in=[1,2,5])
for s1 in s:
    print(s1.id)
```

Logical Operator in SQL

And in SQL

```
select * from admissions_student where id>2 and classname>6;
```

ORM

```
s=student.objects.filter(id__gt=2)&
student.objects.filter(classname__gt=6)
```

Or in SQL

```
select * from admissions_student where id>2 or classname>6;
```

ORM

s=student.objects.filter(id__gt=2)|
student.objects.filter(classname__gt=6)

Not in SQL

select * from admissions_student where not id>2

ORM

s=student.objects.exclude(id__gt=2)

Inserting data into tables

SQL

Insert into admissions_student values(1, 'abc','bb',6,'8976547897');

ORM

To insert single record

s=student(name="sam",fathername="arun",classname=9,contact="8976
567890")

s.save()

print(s)

To insert multiple records

s=student.objects.bulk_create(

[

student(name="divya",fathername="vijay",classname=8,contact="9876509876"),

student(name="ajay",fathername="sam",classname=7,contact="8765789080")

])

s.save()

print(s)

Delete Record in SQL

delete from admissions_student where id=1;

ORM

s=student.objects.get(id=1)

s.delete() #row will be deleted

To delete multiple records

s=student.objects.filter(id__in=[1,3,5])

s.delete() #rows will be deleted

Updating Records in SQL

update admissions_student set classname=6,contact= '8562314567' where id=1;

ORM

```
s=student.objects.get(id=1)
s.classname=6
s.contact= "8562314567"
s.save() # now 1st record will be updated
```

Sorting

SQL using order by asc

```
select * from admissions_student orderby classname asc;
```

ORM

```
s=student.objects.all().order_by('classname')
for s1 in s:
    print(s1.name)
```

SQL using order by desc

```
select * from admissions_student orderby classname desc;
```

ORM

```
s=student.objects.all().order_by('-classname')
for s1 in s:
    print(s1.name)
```

Aggregate Functions

In SQL

Avg

select avg(id) from admissions_student;

Sum

select sum(id) from admissions_student;

Count

select count(id) from admissions_student;

Min

select min(id) from admissions_student;

Max

select max(id) from admissions_student;

In ORM

Avg

python manage.py shell

from admissions.models import student

from django.db.models import Avg,Sum,Min,Max,Count

s=student.objects.all().aggregate(Avg('id'))

print(s)

Sum

```
s=student.objects.all().aggregate(Sum('id'))
print(s)
```

Count

```
s=student.objects.all().aggregate(Count('id'))
print(s)
```

Min

```
s=student.objects.all().aggregate(Min('id'))
print(s)
```

Max

```
s=student.objects.all().aggregate(Max('id'))
print(s)
```

Model Forms in Django

Model Forms in Django:

- In WebPages automatically a form will creates so after that we click on submit button the entered data in the form saved into database automatically we click on this button.
- Remaining to other frameworks django code is very simple.

Steps to create Model Forms:

1. Create a file forms.py in our application folder (admissions app)

admissions->forms.py

2. Create a class that is inherited from forms.ModelForm

- For creating a student model for that model automatically form has to come and entered data in that form should be saved in database.

So to import forms

from django import forms

from admissions.models import student

class StudentModelForm(forms.ModelForm):

3. Create a class Meta inside above ModelForm Class.

class Meta:

model=student

fields='__all__'

forms.py

from django import forms

from admissions.models import student

class StudentModelForm(forms.ModelForm):

class Meta:

```
        model=student
        fields='__all__'
```

4) Use this ModelForm into views.py

views.py

```
from admissions,models import student
from admissions.forms import StudentModelForm
def addadmission(request):
    form=StudentModelForm
    studentform= {'form':form}
    return render(request,'admissions/addadmission.html',studentform)
```

5. Display the form in Template:

addadmissions.html

```
<html>
<head>
<title>add admission page</title>
</head>
<body>
<h1>Add Admission Page</h1>
<form method="POST">
```

```
<table>
{{form.as_table}}
</table>
<input type="submit" value="Add Student">
</form>
</body>
</html>
```

{{form.as_table}}->It shows data in tabular format

{{form.as_p}}->It shows data in paragraph format

{{form.as_ul}}->It shows data in unordered list format

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/ad/newadm

Saving form data into database

Saving form data into database

How to retrieve data from form on how to save into database

Read from input and save into Database

1. Get forms objects with inputs

2. Save

form =StudentModelForm(request.POST)

form.save ()

- We have to check initially that method is POST or GET for that if we write if condition.

```
if request.method== "POST":
        form.StudentModelForm(request.POST)
        if form.is_valid():
                form.save()
```

views.py

```
from admissions.models import student
from admissions.forms import StudentModelForm
def addadmission(request):
        form=StudentModelForm
```

```
    studentform= {'form':form}
    if request.method=="POST":
        form=StudentModelForm(request.POST)
        if form.is_valid():
            form.save()
    return render(request,'admissions/addadmission.html',studentform)
```

- When we run this code error will come that error is csrf.

Error: cross site request forgery (csrf)

- It is attacked by hacker on by database.
- So, for other languages we have to write a code for csrf but in django we can overcome this by using simple tag {% csrf_token %}.
- This token generates unique code in the form of alpha numeric and it is mandatory.

addadmissions.html

```
<html>
<head>
<title>add admission page</title>
</head>
<body>
<h1>Add Admission Page</h1>
```

```
<form method="POST">
<table>
{{form.as_table}}
</table>
<input type="submit" value="Add Student">
{% csrf_token %}
</form>
</body>
</html>
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/newadm

Creating HTML Forms using Django Forms

Creating HTML Forms using Django Forms

- Just we are taking the input and processing the process but we are not storing the entered data in database.

- Instead of ModelForm by using forms how we can create an html forms as follows.

To create temporary forms:

1. Create a file forms.py in our application folder(admission app)

2. Create a class that is inherited form forms.Form

3. Define Fields

4. Use forms in view

5. Display the form in template

Vendors:

name, address, contact,item

forms.py

```
from django import forms
class VendorForm(forms.Form):
        name=forms.CharField()
        address=forms.CharField()
        contact=forms.CharField()
        item=forms.CharField()
```

views.py

```
from admissions.forms import VendorForm
```

```
def addvendor(request):
    form=VendorForm
    vform={'form':form}
    if request.method=="POST":
        form=VendorForm(request.POST)
        if form.is_valid():
            print(form.cleaned_data['name'])
            print(form.cleaned_data['address'])
            print(form.cleaned_data['contact'])
            print(form.cleaned_data['item'])
    return render(request, 'admissions/addvendor.html',vform)
```

Go to admission app and select the urls.py file and type the below code

urls.py

```
from admissions.views import addvendor
urlpatterns=[
    path('newvendor/',addvendor),
]
```

Go to templates folder and select admissions folder and create new file addvendor.html

addvendor.html

```
<html>
<head>
<title>add vendor page</title>
</head>
<body>
<h1>Add Vendor Page</h1>
<form method="POST">
<table>
{{form.as_table}}
</table>
<input type="submit" value="AddVendor">
{% csrf_token %}
</form>
</body>
</html>
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action http://127.0.0.1:8000/ad/newvendor

CRUD Operations in Django:

CRUD Operations in Django:

1. Creating a project SchoolProject in the current directory.

    django-admin startproject SchoolProject

2. Move to that current directory

    cd SchoolProject

3. To run the server if server is working

    python manage.py runserver

4. Go to browser window and type this url http://127.0.0.1:8000

After checking the servers is worked and drag and drop the SchoolProject into the atom IDE.

Creating an Application

Here we are creating the first application name as admissions

11. Creating an application admissions
    django-admin startapp admissions

settings.py(SchoolProject)

import os

TEMPLATES=[

```
{
'DIRS':[os.path.join(BASE_DIR,"templates")],
}
]
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'admissions',
]
DATABASES={
        'default':{
                'ENGINE':'django.db.backends.mysql',
                'NAME':'schoolapp',
                'USER':'root',
                'PASSWORD':'',
                'HOST':'localhost',
                'PORT':'3306',
```

```
        }
}
STATIC_URL= '/static/'
```

models.py(admissions app)

```
from django.db import models
class student(models.Model):
        name = models.CharField(max_length=100)
        fathername = models.CharField(max_length=100)
        classname = models.IntegerField()
        contact = models.CharField(max_length=100)
```

makemigrations command:

```
python manage.py makemigrations
python manage.py sqlmigrate admissions 0001
```

migrate command:

```
        python manage.py migrate
```

forms.py (admissions app)

```
from django import forms
from admissions.models import student
class StudentModelForm(forms.ModelForm):
        class Meta:
```

```
        model=student
        fields='__all__'
```

views.py (admissions app)

```
from django.shortcuts import render
from admissions.models import student
from admissions.forms import StudentModelForm
def homepage(request):
    return render(request,'index.html')
def addadmission(request):
    form=StudentModelForm
    studentform= {'form':form}
    if request.method=="POST":
        form=StudentModelForm(request.POST)
        if form.is_valid():
            form.save()
    return render(request,'admissions/addadmission.html',studentform)
def admissionreport(request):
    result=student.objects.all();
    students={ 'allstudents':result}
    return render(request, 'admissions/admissionreport.html',students)
```

```
def delstudent(request,id):
    s=student.objects.get(id=id)
    s.delete()
    return admissionreport(request)
def updatestudent(request,id):
    s=student.objects.get(id=id)
    form=StudentModelForm(instance=s)
    dict={ 'form':form}
    if request.method== "POST":
        form=StudentModelForm(request.POST,instance=s)
        if form.is_valid():
            form.save()
    return render(request, 'admissions/updateadmissions.html',dict)
```

Creating urls.py file in admission application

urls.py

```
from django.urls import path
from admissions.views import addadmission
from admissions.views import admissionreport
from admissions.views import delstudent
from admissions.views import updatestudent


urlpatterns = [
   path('newadm/', addadmission),
```

```
    path('admreport/', admissionreport),
    path ('delete/<int:id>',delstudent),
    path ('update/<int:id>',updatestudent),
]
```

Open urls.py in the project folder

urls.py

```
from django.contrib import admin
from django.urls import path,include
from admissions.views import homepage

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',homepage),
    path('ad/', include('admissions.urls')),

]
```

index.html (templates folder)

```
{% load static %}

<html>

<head>

<title>add admission page</title>

<link rel="stylesheet" href="{% static 'stylesheets/style.css' %}"/>

</head>

<body>
```

```
<h1>School Project</h1>
<div class="module">
<h1>Admissions</h1>
<a href="./ad/newadm">Add Admissions</a>
<a href="./ad/admreport"><button type="button"
name="button">Admission Report</button></a>
</div>
</body>
</html>
```

style.css(admissions app->static folder)

```
body
{
    background-color:skyblue;
}
h1
{
    color:blue;
}
```

addadmissions.html(templates folder->admissions folder)

```
<html>
<head>
```

```
<title>add admission page</title>
</head>
<body>
<h1>Add Admission Page</h1>
<form method="POST">
<table>
{{form.as_table}}
</table>
<input type="submit" value="Add Student">
{% csrf_token %}
</form>
</body>
</html>
```

admissionreport.html(templates folder->admissions folder)

```
<html>
<head>
<title>Admission Report Page</title>
</head>
<body>
<table border="1">
```

```
<tr>
<th>Id</th>
<th>Student Name</th>
<th>Father Name</th>
<th>Class Name</th>
<th>contact</th>
<th>Delete Action </th>
<th>Update Action </th>
</tr>
{% for s in allstudents %}
<tr>
<td>{{s.id}}</td>
<td>{{s.name}}</td>
<td>{{s.fathername}}</td>
<td>{{s.classname}}</td>
<td>{{s.contact}}</td>
<td><a href="/ad/delete/{{s.id}}">Delete</a></td>
<td><a href="/ad/update/{{s.id}}">Update</a></td>
</tr>
{% endfor %}
</table>
```

```
</body>
</html>
```

updateadmissions.html(templates folder->admissions folder)

```
<html>
<head>
<title>update admissions page</title>
</head>
<body>
<h1>Update Student Details:</h1>
<form method="POST">
<table>
{{form.as_table}}
</table>
<input type="submit" value="Update Student">
{% csrf_token %}
</form>
</body>
</html>
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action http://127.0.0.1:8000

Class Based Views

Class Based Views:

Create subclass of view (django.views.generic)

CRUD Operations using class based views:

django.views.generic

View

ListView

DetailView

CreateView

UpdateView

DeleteView

Reading data from database:

- Create a model Teacher

models.py(admissions app)

```
from django.db import models

class Teacher(models.Model):

    name = models.CharField(max_length=100)
```

```
exp = models.IntegerField()
subject =models.CharField(max_length=100)
contact = models.CharField(max_length=100)
```

Go to python cmd prompt

Go to project location

python manage.py makemigrations

python manage.py migrate

Go to mysql cmd prompt

Set the path for mysql

path c:\xampp\mysql\bin

mysql –u root –p

show databases;

use schoolapp;

show tables;

desc admissions_teacher;

insert into admissions_teacher values(1, 'abc',10,'cse','8967564534');

insert into admissions_teacher values(2, 'def',12,'ece','9087654343');

Django provide a ListView to read data from database

To perform read operation we have to use ListView steps are as follows:

1. Create a class based view that is inherited to ListView.

2. Provide value to the model attribute (mandatory).

3. Default template name is modelname_list.html.You may pass your own template name by passing value to the template name (template name attributes).

4. Default Context object name (Object that receives all the objects from ORM) is modelname_list.So; you can set your own context_object_name by passing the value to this attribute.

5. Create a template with name modelname_list.html and print objects attribute from the list received (modelname_list).

6. Configure the url(classname.as_view())

views.py

```
from admissions.models import Teacher

from django.views.generic import ListView

class Teacherread(ListView):

        model=Teacher  #note:here data stored in teacher_list

                       #teacher_list=teacher.objects.all()

                       #context_object_name= 'result'

                       #default html page is teacher_list.html

                       #templatename='list.html'
```

- Go to templates folder and select admissions folder and create new file teacher_list.html

teacher_list.html:

```
<html>
<head>
<title>Teacher Page</title>
</head>
<body>
<table border="1">
<tr>
<th>Id</th>
<th>Name</th>
<th>Experience</th>
<th>Subject</th>
<th>Contact</th>
</tr>
{% for s in teacher_list %}
<tr>
<td>{{s.id}}</td>
<td>{{s.name}}</td>
```

<td>{{s.exp}}</td>

<td>{{s.subject}}</td>

<td>{{s.contact}}</td>

</tr>

{% endfor %}

</body>

</html>

urls.py(admissions app)

```
from admissions.views import Teacherread
urlpatterns = [
   path ('teacherlist/',Teacherread.as_view()),
]
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/teacherlist

Retrieving a Single row using DetailView:

To perform this we need to use DetailView steps are as follows:

1. Create a class based view that inherits from DetailedView.
2. Provide value to model attribute(mandatory).

3. Default Template name is modelname_detail.html.You may pass your own template name by passing value to the template name attribute.
4. Default context object name(object that received all the object from ORM) is model name.You can set your own context_object_name by passing the values to this attributes.
5. Create a template with name modelname_detail.html and print the object attributes from the list received(modelname_list).
6. Configure the url path('teacherdetail/<int:pk>/',classname.as_view())

views.py

```
from django.views.generic import DetailView

class Teacherdetail(DetailView):

    model=Teacher  #Teacherdetail data stores in that name

                   #default html page is teacher_detail.html

                   #templatename='detail.html'
```

- Go to templates folder and select admissions folder and create new file teacher_detail.html

teacher_detail.html:

```
<html>

<head>

<title>Teacher Detail Page</title>

</head>
```

```
<body>

<h1>Name:{{teacher.name}}</h1>

<h1>Experience:{{teacher.exp}}</h1>

<h1>Subject:{{teacher.subject}}</h1>

<h1>Contact:{{teacher.contact}}</h1>

</body>

</html>
```

urls.py(admissions app)

```
from admissions.views import Teacherdetail
urlpatterns = [
   path ('teacherdetail/<int:pk>/',Teacherdetail.as_view()),
]
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/teacherdetail

Create Operation Using Class Based Views:

To perform this operation we need to use CreateView steps are as follows:

1. Create a class based view that inherits from CreateView.

2. Provide value to the model attribute(mandatory).
3. Provide values to fields(fields=(fieldslist separated by comma))
4. Default template name is modelname_form.html.You may pass your own template name by passing value to your template name attributes.
5. Create a method get_absolute_url() in model class.

```
from django.urls import reverse
def get_absolute_url(self):
    return reverse('urlname',kwargs='pk':self.pk})
```

6. Create a template with name modelname_from.html
7. Configure the url classname.as_view().

views.py

```
from django.views.generic import CreateView

class Insertteacher(CreateView):

    model=Teacher #form=TeacherModelForm

    fields=( 'name','exp','subject','contact') #teacher_form.html
```

- Go to templates folder and select admissions folder and create new file teacher_form.html

teacher_form.html:

```
<html>

<head>
```

```
<title>Insert Teacher</title>
</head>
<body>
<form method= "POST">
<table>
{{form.as_table}}
</table>
<input type= "submit" value="Add Teacher">
{% csrf_token %}
</form>
</body>
</html>
```

urls.py(admissions app)

```
from admissions.views import Insertteacher

urlpatterns = [
   path ('insertteacher/',Insertteacher.as_view()),

]
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/ad/insertteacher

models.py(admissions app)

```
from django.urls  import reverse
from django.db import models
class Teacher(models.Model):
        name = models.CharField(max_length=100)
        exp = models.IntegerField()
        subject models.CharField(max_length=100)
        contact = models.CharField(max_length=100)

        def get_absolute_url(self):
                return reverse('listteacher')
```

urls.py(admissions app)

```
from admissions.views import Teacherread
from admissions.views import Insertteacher

urlpatterns = [
   path ('teacherlist/',Teacherread.as_view(),name='listteacher'),
   path ('insertteacher/',InsertTeacher.as_view()),
```

]

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/teacherlist

Note:

1. We can return to a page (based on url by using reverse())

2. We can send the arguments to the url by using args/kwargs in reverse function.

3. Generally listteacher url doesn't require any arguments so it returns error after adding the record to the database.

4. So, we have to send the arguments to this urls only when it is required.Here args is optional.

models.py(admissions):

```
def get_absolute_url(self):
    return reverse('listteacher',kwargs={'pk':self.pk})
```

urls.py(admissions):

```
from admissions.views import Teacherdetail
urlpatterns = [
   path
('teacherdetail.<int:pk>/',Teacherdetail.as_view(),name='listteacher'),
]
```

Update Operation Using Class Based Views:

To perform this operation we need to use UpdateView steps are as follows:

1. Create a class based view that inherits from UpdateView.
2. Provide value to the model attribute (mandatory).
3. Provide value to fields(fields=(fields list separated by comma))
4. Default template name modelname_form.html.You may pass your own template name by passing value to the template_name attribute.
5. Create a method get_absolute_url() in model class.

```
from django.urls import reverse
def get_absolute_url(self):
    return reverse('urlname',kwargs='pk':self.pk})
```

6. Create a template with name modelname_from.html and print the objects attributes from the list received(modelname_list)
7. Configure the url classname.as_view().

views.py

```
from django.views.generic import UpdateView
```

```
class Updateteacher(UpdateView):
    model=Teacher
    fields=( 'name','contact')
```

- Go to templates folder and select admissions folder and create new file teacher_form.html

teacher_form.html:

```
<html>
<head>
<title>UpdateTeacher</title>
</head>
<body>
<form method= "POST">
<table>
{{form.as_table}}
</table>
<input type= "submit" value="Update Teacher">
{% csrf_token %}
</form>
</body>
</html>
```

urls.py(admissions app)

```
from admissions.views import Updateacher
urlpatterns = [
   path ('updateteacher/<int:pk>/',Updateteacher.as_view()),
]
```

- Go to templates folder and select admissions folder and create new file teacher_list.html

teacher_list.html:

```
<html>

<head>

<title>Teacher Page</title>

</head>

<body>

<table border="1">

<tr>

<th>Id</th>

<th>Name</th>

<th>Experience</th>

<th>Subject</th>

<th>Contact</th>

<th>Update Action</th>
```

```
</tr>
{% for s in teacher_list %}
<tr>
<td>{{s.id}}</td>
<td>{{s.name}}</td>
<td>{{s.exp}}</td>
<td>{{s.subject}}</td>
<td>{{s.contact}}</td>
<td><a href= "/ad/updateteacher/{{s.id}}">Update</a></td>
</tr>
{% endfor %}
</body>
</html>
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/ad/updateteacher/2

Delete Operation Using Class Based Views:

To perform this operation we need to use DeleteView steps are as follows:

1. Create a class based view that inherits from DeleteView.
2. Provide value to the model attribute (mandatory).
3. Provide values to success_url=reverse_lazy('urlname')
4. Create modelname_confirm_delete.html to which django forward as and waits for our confirmation.
5. Configure the url(classname.as_view())

views.py

```
from django.views.generic import DeleteView

class Deleteteacher(DeleteView):

        model=Teacher
```

urls.py(admissions app)

```
from admissions.views import Deleteteacher
urlpatterns = [
   path ('deleteteacher/<int:pk>/',Deleteteacher.as_view()),
]
```

- Go to templates folder and select admissions folder and create new file teacher_list.html

teacher_list.html:

```
<html>
```

```
<head>
<title>Teacher Page</title>
</head>
<body>
<table border="1">
<tr>
<th>Id</th>
<th>Name</th>
<th>Experience</th>
<th>Subject</th>
<th>Contact</th>
<th>Update Action</th>
<th>Delete Action</th>
</tr>
{% for s in teacher_list %}
<tr>
<td>{{s.id}}</td>
<td>{{s.name}}</td>
<td>{{s.exp}}</td>
<td>{{s.subject}}</td>
<td>{{s.contact}}</td>
```

```
<td><a href= "/ad/updateteacher/{{s.id}}">Update</a></td>

<td><a href= "/ad/deleteteacher/{{s.id}}">Delete</a></td>

</tr>

{% endfor %}

</body>

</html>
```

- Go to templates folder and select admissions folder and create new file teacher_confirm_delete.html

teacher_confirm_delete.html:

```
<html>
<head>
<title>Delete Teacher Page</title>
</head>
<body>
<form method= "POST">
<button type="submit" value="confirm">Confirm</button>

{% csrf_token %}

</form>

<a href="/ad/teacherlist"><button type="button"
value="cancel">Cancel</button></a>

</body>

</html>
```

views.py

```
from django.views.generic import DeleteView
from django.urls import reverse_lazy
class Deleteteacher(DeleteView):
    model=Teacher
    success_url=reverse_lazy('listteacher')
```

To run the server application

```
python manage.py runserver
```

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/ad/deleteteacher/2

Authentication and Authorization

Authentication

We have 2 types of Security

1. Authentication

2. Authorization

Authentication:

- To enter into particular software applications by using login is nothing but authentication.

Authorization:

- After entering into an application to which links we have to give permissions and to which links not to give permissions is about authorization.

Authentication

1. Add the auth urls

2. Create the login form

3. Secure the views

4. Create the users

5. Test it

Django provides enough urls to authentication (login and logout)

urls.py at project level:

urls.py

```
urlpatterns = [
    path('accounts/',include('django.contrib.auth.urls')),
 ]
```

- Go to templates folder and create new folder name registration and create new file name login.html

login.html(templates/registration)

```
<html>
<head>
  </head>
  <body>
    <form method="POST">
      <table>
      {{form.as_table}}
      </table>
      <button type="submit" name="button">Login</button>
      {% csrf_token %}
      </form>
  </body>
  </html>
```

Secure the views:

views.py

```
from django.contrib.auth.decorators import login_required
```

- without login if we sends the request directly we have to show the login form
- Django provides decorator (@login_required).This decorator stops direct access before that login page will open.
- According to our requirement we have to write this decorator for views.

views.py

```
from django.contrib.auth.decorators import login_required

@login_required

def homepage(request):

    return render(request,'index.html')
```

For class based views:

urls.py(application level)

```
from django.contrib.auth.decorators import login_required

urlpatterns = [

path('getteacherdetail/<int:pk>/',login_required(Teacherdetail.as_view())
),

]
```

After logout where we have to render the page:

settings.py (at project level)

LOGOUT_REDIRECT_URL='/userlogout'

- Go to templates folder and create new file logout.html

logout.html (templates)

```
<h1>User successfully logged out</h1>
```

views.py (application level)

```
def logoutuser(request):
        return render(request,'logout.html')
```

urls.py (at project level)

```
from admissions.views import logoutuser
urlpatterns = [
   path('userlogout/',logoutuser),
]
```

index.html (templates)

```
<a href="./accounts/logout">Logout</a>
```

Authorization:

```
from django.contrib.auth.decorators import
login_required,permission_required

@login_required

@permission_required('admissions.add_student')

def addadmission(request):

        form=StudentModelForm

        studentform= {'form':form}

        if request.method=="POST":

                form=StudentModelForm(request.POST)

                if form.is_valid():

                        form.save()

        return
render(request,'admissions/addadmissions.html',studentform)


@login_required

@permission_required('admissions.view_student')

def admissionreport(request):

        result=student.objects.all();

        students={ 'allstudents':result}

        return render(request, 'admissions/admissionreport.html',students)

@login_required
```

```
@permission_required('admissions.delete_student')
def delstudent(request,id):
    s=student.objects.get(id=id)
    s.delete()
    return admissionreport(request)


@login_required
@permission_required('admissions.change_student')
def updatestudent(request,id):
    s=student.objects.get(id=id)
    form=StudentModelForm(instance=s)
    dict={ 'form':form}
    if request.method== "POST":
        form=StudentModelForm(request.POST,instance=s)
        if form.is_valid():
            form.save()
    return render(request, 'admissions/updateadmissions.html',dict)
```

To give Permissions

- Before that we have to create super user for admin UI

```
python manage.py createsuperuser
```

It will asks like this

username:

email:

password:

password(again):

Here I have to create username as **admin** and password as **fiit@123**

- After creating the super user and run the server

  python manage.py runserver

- Go to browser and open localhost/admin/
- Enter the username and password of admin UI what you have to created by using super user.
- Once login It will shows groups and users and corresponding models what you are register in admin UI(admin.py)

admin.py (application level)

```
from django.contrib import admin

from admissions.models import student,Teacher

# Register your models here.

admin.site.register(student)

admin.site.register(Teacher)

class studentAdmin(admin.ModelAdmin):

    list_display=['id','name','fathername','classname','contact']
```

Django Administration:

localhost/admin

create users

first user:

username:user1

password:fiit@321

confirm password:fiit@321

click save and user1 is created

second user:

username:user2

password:fiit@000

confirm password:fiit@000

click save and user2 is created

- select the user1 and give the permission for accessing views

user1->select filter in user permissions and select the required action(for ex: can change student)like wise you can give many actions after choosing actions and save it.

user2->select filter in user permissions and select the required action (for ex: can view student) likewise you can give many actions after choosing actions and save it.

- Once you will give permissions to users and open browser and type the particular url pattern it will ask to login the user first and then go to that corresponding url pattern and select the actions what you have to provide to user and check it.

For ex:127.0.0.1/ad/admreport

- It will ask the login for that corresponding user
- Here I am giving user2 login and password after that only I will see the admission report view details.

Django Cookie

Django Cookie

- A cookie is a small piece of information which is stored in the client browser.
- It is used to store user's data in a file permanently (or for the specified time).
- Cookie has its expiry date and time and removes automatically when gets expire.
- Django provides built-in methods to set and fetch cookie.
- The **set_cookie()** method is used to set a cookie and **get()** method is used to get the cookie.

The **request.COOKIES['key']** array can also be used to get cookie values.

Django Cookie Example

- In **views.py**, two functions setcookie() and getcookie() are used to set and get cookie respectively

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
 def setcookie(request):
    response = HttpResponse("Cookie Set")
    response.set_cookie('working', 'MNC')
    return response
def getcookie(request):
    data  = request.COOKIES['working']
    return HttpResponse("Working at: "+  data);
```

And URLs specified to access these functions.

urls.py

```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('setcookie/',views.setcookie),
```

```
    path('getcookie/',views.getcookie),
]
```

To run the server application

```
python manage.py runserver
```

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/setcookie

Django Session

Django Session

- A session is a mechanism to store information on the server side during the interaction with the web application.
- In Django, by default session stores in the database and also allows file-based and cache based sessions.
- It is implemented via a piece of middleware and can be enabled by using the following code.

- Put **django.contrib.sessions.middleware.SessionMiddleware** in MIDDLEWARE and **django.contrib.sessions** in INSTALLED_APPS of settings.py file.

To set and get the session in views, we can use **request.session** and can set multiple times too.

- The **class backends.base.SessionBase** is a base class of all session objects. It contains the following standard methods.

Method	Description
__getitem__(key)	It is used to get session value.
__setitem__(key, value)	It is used to set session value.
__delitem__(key)	It is used to delete session object.
__contains__(key)	It checks whether the container contains the particular session object or not.
get(key, default=None)	It is used to get session value of the specified key.

Let's see an example in which we will set and get session values. Two functions are defined in the views.py file.

Django Session Example

- The first function is used to set and the second is used to get session values.

views.py

```
from django.shortcuts import render

from django.http import HttpResponse

def setsession(request):
```

```
    request.session['sname'] = 'abc'
    request.session['semail'] = 'abc@gmail.com'
    return HttpResponse("session is set")
def getsession(request):
    studentname = request.session['sname']
    studentemail = request.session['semail']
    return HttpResponse(studentname+" "+studentemail);
```

- Url mapping to call both the functions.

urls.py

```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('setsession/',views.setsession),
    path('getsession/',views.getsession),
]
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/setsession

Create CSV with Django

Create CSV with Django

- Django uses Python's built-in CSV library to create Dynamic CSV (Comma Separated Values) file. We can use this library in our project's view file.
- Let's see an example; here we have a Django project to which we are implementing this feature. A view function getfile() is created.

Django CSV Example

In this example, we are creating CSV using static data.

Views.py

```
from django.http import HttpResponse

import csv

def getfile(request):

    response = HttpResponse(content_type='text/csv')

    response['Content-Disposition'] = 'attachment; filename="file.csv"'
```

```
    writer = csv.writer(response)
    writer.writerow(['10', 'sam', 'chennai', 'Developer'])
    writer.writerow(['11', 'arun', 'delhi', 'Tester'])
    return response
```

urls.py

```
    from django.contrib import admin
    from django.urls import path
    from admissions import views
    urlpatterns = [
        path('admin/', admin.site.urls),
        path('getfile/',views.getfile),
    ]
```

To run the server application

```
    python manage.py runserver
```

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

    http://127.0.0.1:8000/getfile

Apart from static data, we can get CSV from the database too. See, the following example in which we are getting data from the table by using the student model.

Dynamic CSV using Database

// views.py

```
from admissions.models import student
from django.http import HttpResponse
import csv
def getfile(request):
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="file.csv"'
    students = student.objects.all()
    writer = csv.writer(response)
    for s in students:
        writer.writerow([s.id,s.name,s.fathername,s.classname,s.contact])
    return response
```

To run the server application

python manage.py runserver

- Go to browser window and type this url http://127.0.0.1:8000
- It will shows the view action and choose the view particular action

http://127.0.0.1:8000/getfile

Django PDF

- Here, we will learn how to design and generate PDF file using Django view.
- To generate PDF, we will use ReportLab Python PDF library that creates customized dynamic PDF.

```
pip install reportlab
```

- After installing, we can import it by import keyword in the view file.

- Below is a simple PDF example, in which we are outputting a string message "Hello Django".
- This library provides a canvas and tools that are used to generate customized PDF.

views.py

```
from reportlab.pdfgen import canvas

from django.http import HttpResponse

def getpdf(request):

    response = HttpResponse(content_type='application/pdf')

    response['Content-Disposition'] = 'attachment; filename="file.pdf"'

    p = canvas.Canvas(response)

    p.setFont("Times-Roman", 55)

    p.drawString(100,700, "Hello, Django.")

    p.showPage()
```

```
p.save()
return response
```

- First, provide MIME (content) type as application/pdf, so that output generates as PDF rather than HTML,
- Set Content-Disposition in which provide header as attachment and output file name.
- Pass response argument to the canvas and drawstring to write the string after that apply to the save() method and return response.

urls.py

```
from django.contrib import admin
from django.urls import path
from admissions import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('getpdf/',views.getpdf),
]
```

To run the server application

```
python manage.py runserver
```

- Go to browser window and type this url http://127.0.0.1:8000

- It will shows the view action and choose the view particular action

  http://127.0.0.1:8000/getpdf