

**5****Introduction to web Framework Django****TOPICS COVERED :**

- Introduction to Django
- MVC Design Pattern
- Django Installation
- Starting project
- Django project architecture
- HTTP Client-Server Request – Response, concept of web framework and web application.

**Introduction to Django:**

- 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.

**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 2011	Timezones, in browser testing, app templates.
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. <i>Supported until at least April 2018</i>
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. <i>Supported until at least April 2020</i>
2.0	Dec 2017	First Python 3-only release, Simplified URL routing syntax, Mobile friendly admin.

### **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:**

#### **1. 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.

#### **2. 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.

#### **3. Scalable:**

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

**4. 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.

**5. 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.

**6. 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.

**7. Vast and Supported Community:**

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

**MVC Design Pattern:**

- The MVC pattern is a software architecture pattern that separates data presentation from the logic of handling user interactions(in other words, saves you stress:), it has been around as a concept for a while, and has invariably seen an exponential growth in use since its inception.
- It has also been described as one of the best ways to create client-server applications, all of the best frameworks for web are all built around the MVC concept
- To break it down, here's a general overview of the MVC Concept;

**1. Model:**

- This handles your data representation, it serves as an interface to the data stored in the database itself, and also allows you to interact with your data without having to get perturbed with all the complexities of the underlying database.

**2. View:**

- As the name implies, it represents what you see while on your browser for a web application or In the UI for a desktop application.

**3. Controller:**

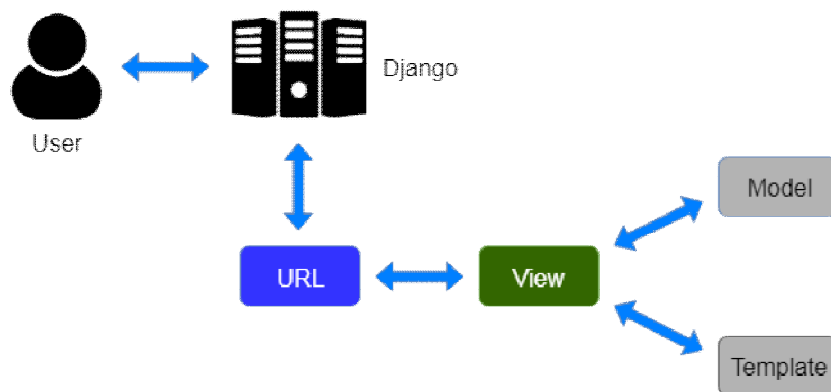
- Provides the logic to either handle presentation flow in the view or update the model's data i.e it uses programmed logic to figure out what is pulled from the database through the model and passed to the view,also gets information from the user through the view

and implements the given logic by either changing the view or updating the data via the model , To make it more simpler, see it as the engine room.

- Now that we understand the general concept of the MVC, understanding how it is implemented in different frameworks can be another task as some frameworks(Django inclusive) like to implement this same functionality in another way making it a bit difficult understanding what actually happens at each layer.
- There's a slight difference between Django's MVC and MVT pattern.
- In MVC pattern, the framework itself deals with the Controller part of the application. While in MVT pattern, it leaves the Template part for the developers. Well, Controller is that component of the software which regulates the interaction between the other components, which are Model and View.
- On the other hand, Template is a file written in HTML and DTL (Django Template Language).

### Django MVT:

- The MVT (Model View Template) is a software design pattern.
- It is a collection of three important components Model View and Template.
- The Model helps to handle database.
- It is a data access layer which handles the data.
- The Template is a presentation layer which handles User Interface part completely.
- The View is used to execute the business logic and interact with a model to carry data and renders a template.
- Although Django follows MVC pattern but maintains its? own conventions.
- So, control is handled by the framework itself.
- There is no separate controller and complete application is based on Model View and Template. That?s why it is called MVT application.
- See the following graph that shows the MVT based control flow.

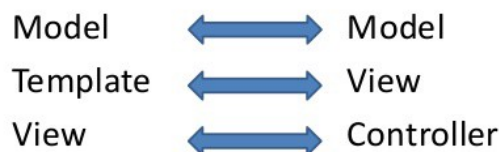


- Here, a user **requests** for a resource to the Django, Django works as a controller and check to the available resource in URL.

- If URL maps, **a view is called** that interact with model and template, it renders a template.
- Django responds back to the user and sends a template as a **response**.

## Is it MVC or MTV??

- In Django it is called MTV rather than MVC.



Models	Describes your data
Views	Controls what users sees
Templates	How user sees it
Controller	URL dispatcher

### 1. Model:

- Just like the Model explanation in the MVC pattern, this also takes the same position as the interface or relationship between the data and contains everything related to data access and validation.

### 2. Template:

- This relates to the View in the MVC pattern as it is the presentation layer that handles the presentation logic in the framework and basically controls what should be displayed and how it should be displayed to the user.

### 3. View:

- This part relates to the Controller in the MVC pattern and handles all the business logic that throws down back to the respective templates.
- It serves as the bridge between the model and the template.

## Django Installation:

1. Go to Windows Power Shell and execute as Administrator.
2. Go to C:\ root folder and type "Set-Execution-Policy Unrestricted" command. This will allow you to write the command from anywhere. Say YES when it is asking for confirmation

**(Please note that if you don't have Windows 10 / Power Shell, then you can go to simple command prompt as well)**

3. Check version of Python and PIP

```
C:\> python -V
```

```
C:\> pip -V
```

4. Now install Virtual Environment for Python

Virtual Environment is not compulsory but it creates a separate environment for our computer to work. Especially Python has so many different versions and every version has some change in syntax. So creating Virtual Environment helps when you have to work with different python versions in a single PC.

For example, you are working on Python 3.8 but some company wants to develop website in Python 2.7, here the syntax is different. Then how to cope up with this situation. Virtual environment helps in this situation.

Remember that using Virtual Environment is JUST A BEST PRACTICE. NOT COMPULSORY TO DO. You can do that without Virtual Environment as well

Create a folder for installing Virtual Environment and go inside that folder. In our case the folder is “BCA”. Now give following command.

```
D:\BCA>pip install virtualenv
```

Above command will create virtual environment.

5. Create virtual environment for your “BCA” folder and then turn Virtual Environment ON  
D:\BCA>virtualenv .

Go to the “Scripts” folder which is created inside “BCA” folder and run “activate”

```
file.D:\BCA\Scripts>activate
```

It will look as follows (BCA)  
D:\BCA\Scripts>

This means your virtual environment has started now. Now whatever you do, will be for this particular environment only.

**Note:** You can write “deactivate” after your work is over to deactivate the virtual environment.

6. Install “Django” in your python virtual environment. If don’t have virtual environment, then also can do the same step.

```
(BCA) D:\BCA> pip install django
```

If you want specific version of django, you can write as follows(BCA) D:\BCA> pip install django==2.0 or something

**Note :** Remember that if you have installed virtual environment, then it will install django againfor that virtual environment even though you have installed it.

**Note :** If you want to confirm that Django is installed or not, go to “Scripts” folder. There you will see a file called “django-admin”. It indicates that django has been installed. That file will be used for further commands of django to create project and all.

### **Starting Project:**

- We have installed Django successfully. Now, we will learn step by step process to create a Django application.
- 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 **djangpapp** in the current directory.  
**django-admin startproject djangpapp**

### **Locate into the Project:**

- Now, move to the project by changing the directory. The Directory can be changed by using the following command.  
**cd djangpapp**
- A Django project contains the following packages and files. The outer directory is just a container for the application. We can rename it further.
- 1. **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 that we will see later on in this tutorial.

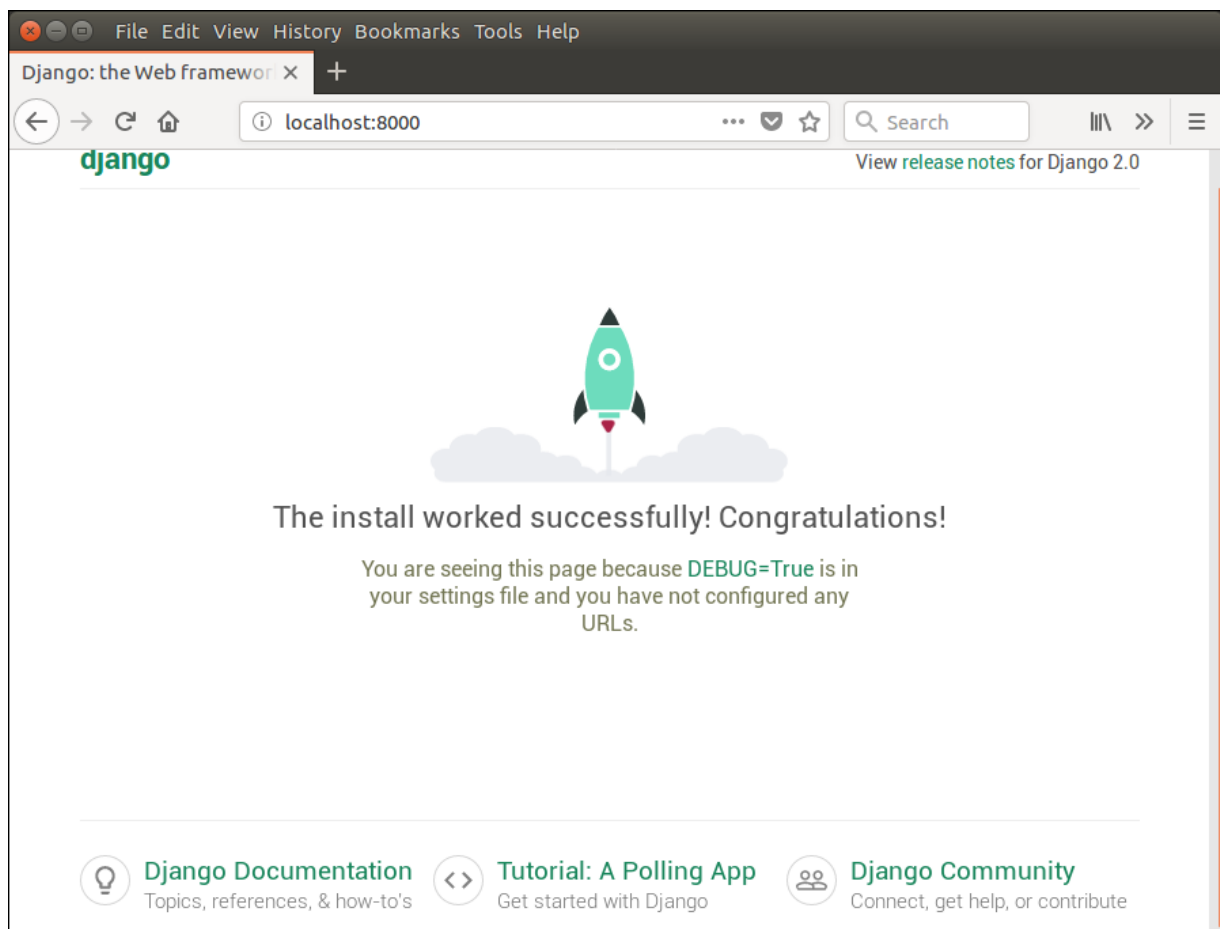
A directory (djangpapp) located inside, is the actual application package name. Its name is the Python package name which we'll need to use to import module inside the application.

- 2. **\_\_init\_\_.py:** It is an empty file that tells to the Python that this directory should be considered as a Python package.

3. **settings.py:** This file is used to configure application settings such as database connection, static files linking etc.
  4. **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.
  5. **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. Let's access it using the browser, it looks like the below.

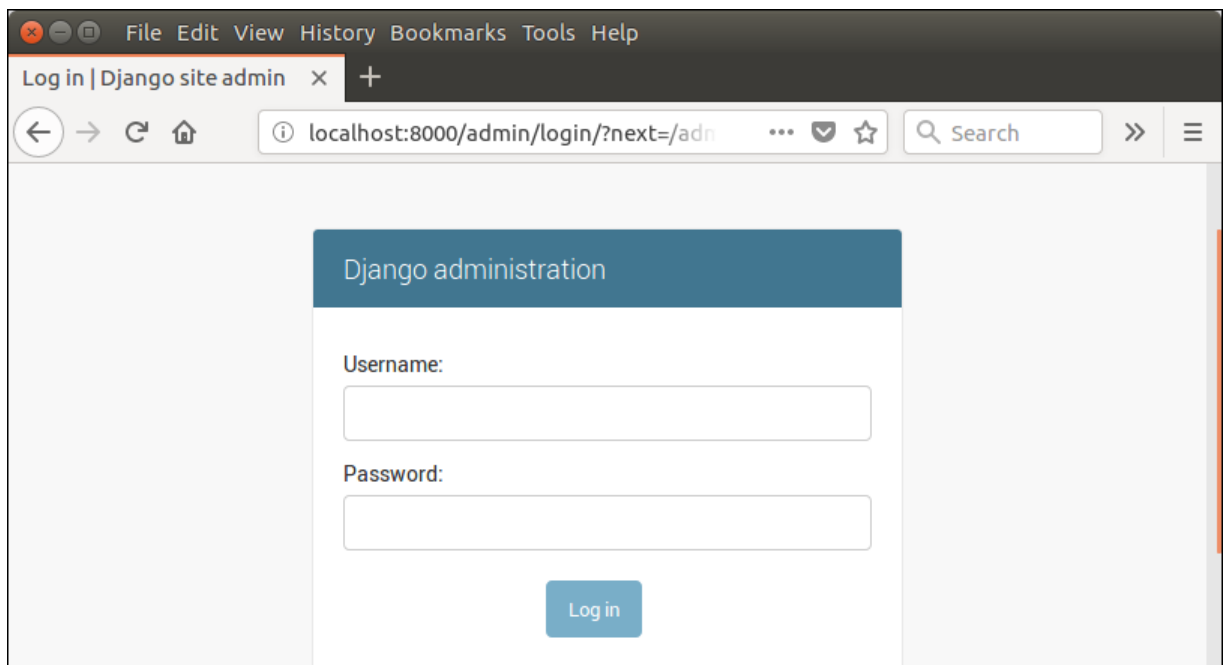




- The application is running successfully. Now, we can customize it according to our requirement and can develop a customized web application.

### **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:

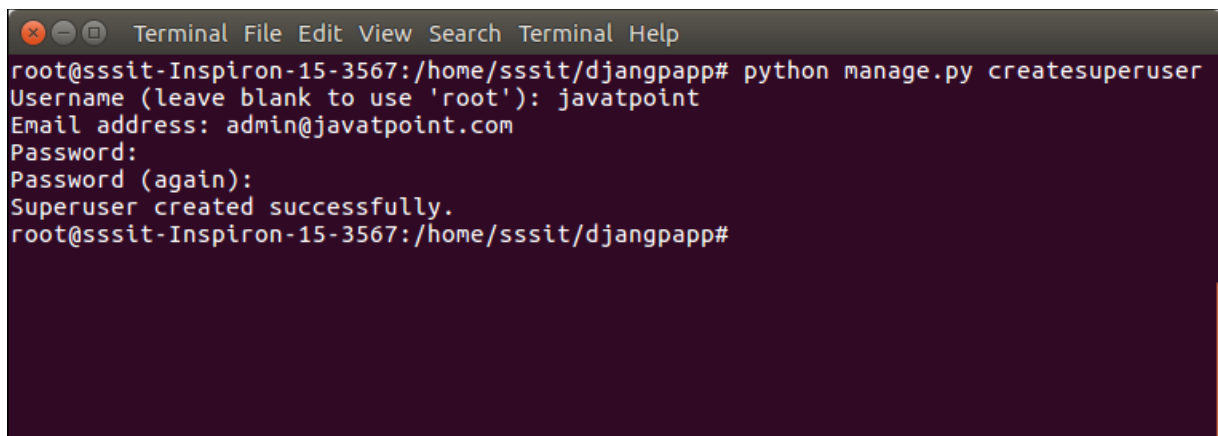


- It prompts for login credentials if no password is created yet, use the following command to create a user.

### **Migrations:**

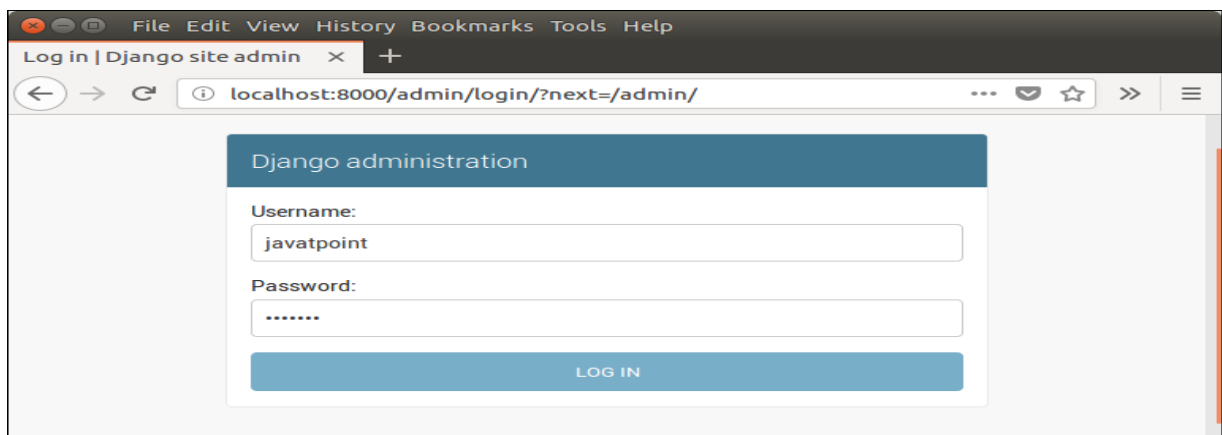
- Migration is a way of applying changes that we have made to a model, into the database schema.
- Django creates a migration file inside the migration folder for each model to create the table schema, and each table is mapped to the model of which migration is created.
- Django provides the various commands that are used to perform migration related tasks.
- After creating a model, we can use these commands.

1. **makemigrations** : It is used to create a migration file that contains code for the tabled schema of a model.
  2. **migrate** : It creates table according to the schema defined in the migration file.
  3. **sqlmigrate** : It is used to show a raw SQL query of the applied migration.
  4. **showmigrations** : It lists out all the migrations and their status.
- You are required to launch the server so that you can access your admin interface. The following command will help you to initiate the database:  
**python manage.py migrate**
  - To create all the necessary tables or collections as required by your admin interface, you need to use syncdb.
  - This command works as per your database type.
  - In case, if you haven't created any superuser yet, then you will be prompted to create one.
  - To create a superuser, run the following command on the terminal.  
**python manage.py createsuperuser**

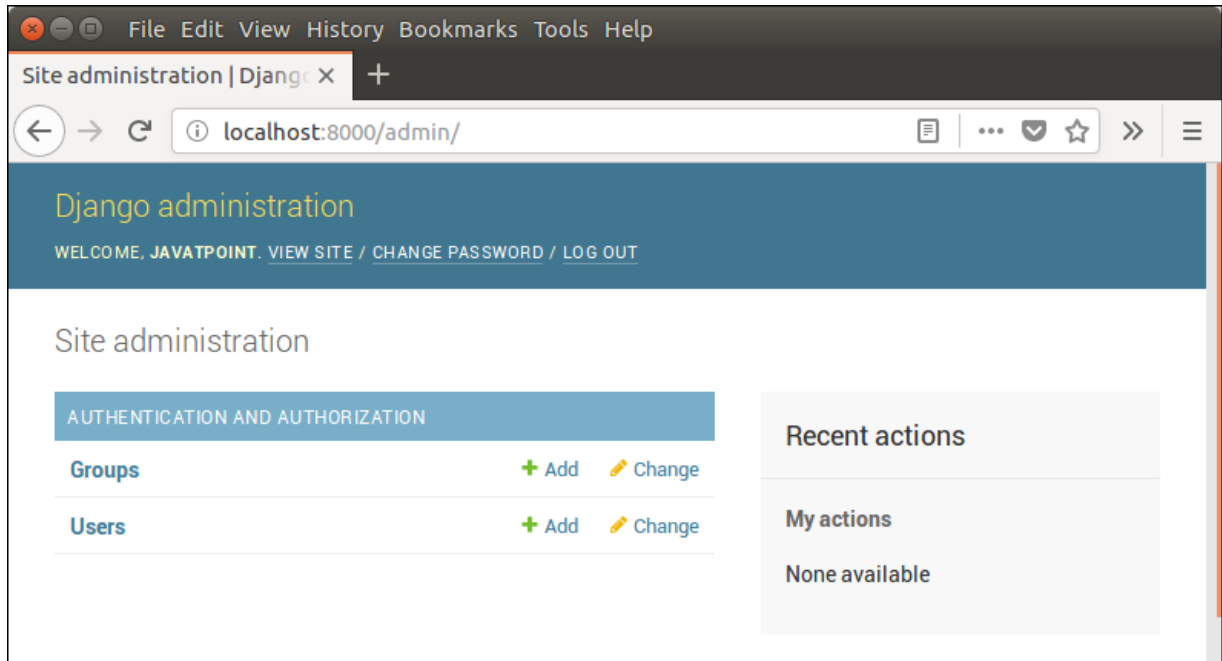


```
Terminal File Edit View Search Terminal Help
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp# python manage.py createsuperuser
Username (leave blank to use 'root'): javatpoint
Email address: admin@javatpoint.com
Password:
Password (again):
Superuser created successfully.
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp#
```

- Now start development server and access admin login.  
**python manage.py runserver**
- Provide created username and password and login.



- After login successfully, it shows the following interface.



- It is a Django Admin Dashboard. Here, we can add and update the registered models.

### **Django App:**

- We have seen a procedure to create a Django project. Now, in this topic, we will create app inside the created project.
- 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.

### **Creating an App:**

- To create an app, we can use the following command.  
`python manage.py startapp appname`

### **Django App Example:**

`python manage.py startapp myapp`

- See the directory structure of the created app, it contains the **migrations** folder to store migration files and model to write business logic.
- Initially, all the files are empty, no code is available but we can use these to implement business logic on the basis of the MVC design pattern.

- To run this application, we need to make some significant changes which display **hello world** message on the browser.
- Open **views.py** file in any text editor and write the given code to it and do the same for **urls.py** file too.

// **views.py**

```
from django.shortcuts import render

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

def hello(request):
    return HttpResponse("<h2>Hello, Welcome to Django!</h2>")
```

// **urls.py**

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

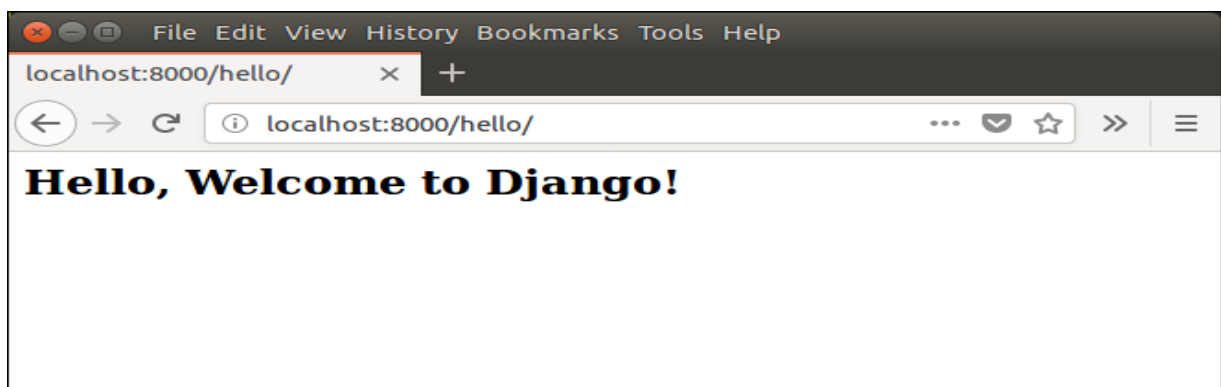
urlpatterns = [
    path('admin/', admin.site.urls),
    path('hello/', views.hello),
]
```

- We have made changes in two files of the application. Now, let's run the it by using the following command. This command will start the server at port 8000.

### Run the Application:

python manage.py runserver

- Open any web browser and enter the URL **localhost:8000/hello**. It will show the output given below.



## Django Project Architecture:

### 1. Django Model:

- In Django, a model is a class which is used to contain essential fields and methods.
- Each model class maps to a single table in the database.
- Django Model is a subclass of **django.db.models.Model** and each field of the model class represents a database field (column).
- Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table.
- Model is defined in **Models.py** file.
- This file can contain multiple models.
- Let's see an example here, we are creating a model **Employee** which has two fields **first\_name** and **last\_name**.

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

```
class Employee(models.Model):  
    first_name = models.CharField(max_length=30)  
    last_name = models.CharField(max_length=30)
```

- The **first\_name** and **last\_name** fields are specified as class attributes and each attribute maps to a database column.
- This model will create a table into the database that looks like below.

```
CREATE TABLE appname_employee (  
    "id" INT NOT NULL PRIMARY KEY,  
    "first_name" varchar(30) NOT NULL,  
    "last_name" varchar(30) NOT NULL  
);
```

- The created table contains an auto-created id field.
- The name of the table is a combination of app name and model name that can be changed further.

### Register / Use Model:

- After creating a model, register model into the **INSTALLED\_APPS** inside **settings.py**.

#### For example:

```
INSTALLED_APPS = [  
    #...  
    'appname',  
    #...  
]
```

**Django Model Fields:**

- The fields defined inside the Model class are the columns name of the mapped table. The fields name should not be python reserve words like clean, save or delete etc.
- Django provides various built-in fields types.

Field Name	Class	Particular
AutoField	class AutoField(**options)	It An IntegerField that automatically increments.
BigAutoField	class BigAutoField(**options)	It is a 64-bit integer, much like an AutoField except that it is guaranteed to fit numbers from 1 to 9223372036854775807.
BigIntegerField	class BigIntegerField(**options)	It is a 64-bit integer, much like an IntegerField except that it is guaranteed to fit numbers from - 9223372036854775808 to 9223372036854775807.
BinaryField	class BinaryField(**options)	A field to store raw binary data.
BooleanField	class BooleanField(**options)	A true/false field. The default form widget for this field is a CheckboxInput.
CharField	class DateField(auto_now=False, auto_now_add=False, **options)	It is a date, represented in Python by a datetime.date instance.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **options)	It is a date, represented in Python by a datetime.date instance.
DateTimeField	class DateTimeField(auto_now=False, auto_now_add=False, **options)	It is used for date and time, represented in Python by a datetime.datetime instance.
DecimalField	class DecimalField(max_digits=None, decimal_places=None, **options)	It is a fixed-precision decimal number, represented in Python by a Decimal instance.
DurationField	class DurationField(**options)	A field for storing periods of time.

EmailField	class EmailField(max_length=254, **options)	It is a CharField that checks that the value is a valid email address.
FileField	class FileField(upload_to=None, max_length=100, **options)	It is a file-upload field.
FloatField	class FloatField(**options)	It is a floating-point number represented in Python by a float instance.
ImageField	class ImageField(upload_to=None, height_field=None, width_field=None, max_length=100, **options)	It inherits all attributes and methods from FileField, but also validates that the uploaded object is a valid image.
IntegerField	class IntegerField(**options)	It is an integer field. Values from -2147483648 to 2147483647 are safe in all databases supported by Django.
NullBooleanField	class NullBooleanField(**options)	Like a BooleanField, but allows NULL as one of the options.
PositiveIntegerField	class PositiveIntegerField(**options)	Like an IntegerField, but must be either positive or zero (0). Values from 0 to 2147483647 are safe in all databases supported by Django.
SmallIntegerField	class SmallIntegerField(**options)	It is like an IntegerField, but only allows values under a certain (database-dependent) point.
TextField	class TextField(**options)	A large text field. The default form widget for this field is a Textarea.
TimeField	class TimeField(auto_now=False, auto_now_add=False, **options)	A time, represented in Python by a datetime.time instance.

**Django Model Fields Example:**

```

first_name = models.CharField(max_length=50) # for creating varchar column
release_date = models.DateField()           # for creating date column
num_stars = models.IntegerField()           # for creating integer column

```

**Field Options:**

- Each field requires some arguments that are used to set column attributes. For example, CharField requires max\_length to specify varchar database.
- Common arguments available to all field types. All are optional.

Field Options	Particulars
Null	Django will store empty values as NULL in the database.
Blank	It is used to allow field to be blank.
Choices	An iterable (e.g., a list or tuple) of 2-tuples to use as choices for this field.
Default	The default value for the field. This can be a value or a callable object.
help_text	Extra "help" text to be displayed with the form widget. It's useful for documentation even if your field isn't used on a form.
primary_key	This field is the primary key for the model.
Unique	This field must be unique throughout the table.

**Django Model Example:**

- We created a model Student that contains the following code in **models.py** file.

**//models.py**

```
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=30)
    contact = models.IntegerField()
    email = models.EmailField(max_length=50)
    age = models.IntegerField()
```

**//admin.py**

```
from django.contrib import admin

from mypro(projectname).models import Student(Tablename)

admin.site.register(Student)
```



**Setup for database:**

- Write following code in “settings.py”

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'abc',  
        'USER': 'root',  
        'PASSWORD': '',  
        'HOST': 'localhost',  
        'PORT': '3306',  
        'OPTIONS': {  
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'"  
        }  
    }  
}
```

**Migrating Database:**

- This will create several tables in your “mysql”.
- Remember that we have given the name of our database as “abc”.
- So before migrating project, you should create a database in mysql with name “abc”.
- First you need to inform Django that you have several models.
- For that you give following commands

**python manage.py migrate --run-syncdb**

- It will create a table **Student**. The table structure looks like the below.

The screenshot shows the phpMyAdmin interface for a database named 'djangomodel'. The selected table is 'myapp\_student'. The 'Table structure' tab is active, displaying a table with 6 columns: id, first\_name, last\_name, contact, email, and age. Each column has a 'Change' and 'Drop' action link.

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop
2	first_name	varchar(20)	latin1_swedish_ci		No	None		Change Drop
3	last_name	varchar(30)	latin1_swedish_ci		No	None		Change Drop
4	contact	int(11)			No	None		Change Drop
5	email	varchar(50)	latin1_swedish_ci		No	None		Change Drop
6	age	int(11)			No	None		Change Drop

## 2. Django Views:

- A view is a place where we put our business logic of the application.
- The view is a python function which is used to perform some business logic and return a response to the user.
- This response can be the HTML contents of a Web page, or a redirect, or a 404 error.
- All the view function are created inside the **views.py** file of the Django app.

### Django View Simple Example:

//views.py

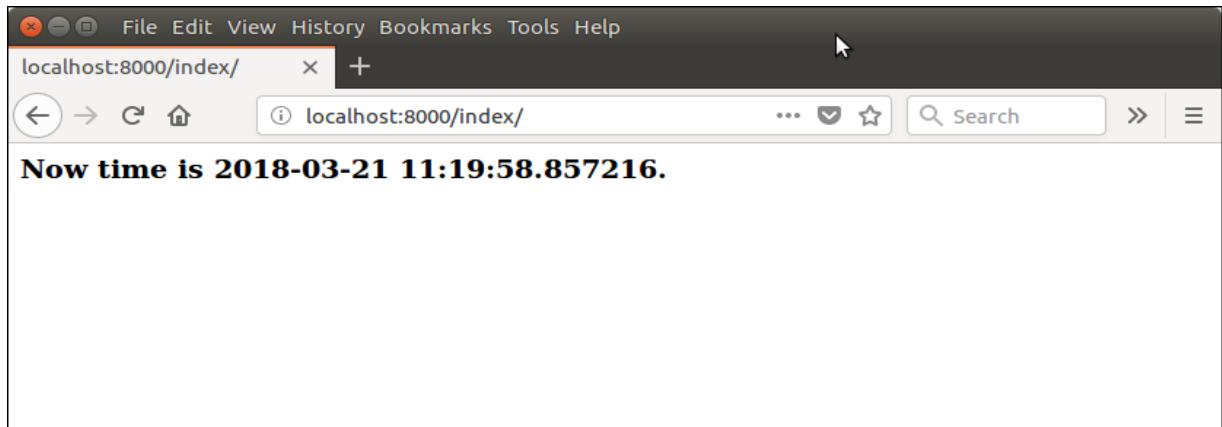
```
import datetime
# Create your views here.
from django.http import HttpResponse

def index(request):
    now = datetime.datetime.now()
    html = "<html><body><h3>Now time is %s.</h3></body></html>" % now
    return HttpResponse(html) # rendering the template in HttpResponse
```

- Let's step through the code.
- First, we will import DateTime library that provides a method to get current date and time and HttpResponse class.
- Next, we define a view function index that takes HTTP request and respond back.
- View calls when gets mapped with URL in **urls.py**. For example

```
from myapp import views
```

```
path('index/', views.index),
```

**Output:****3. Django Templates:**

- Django provides a convenient way to generate dynamic HTML pages by using its template system.
- A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

**Why Django Template?**

- In HTML file, we can't write python code because the code is only interpreted by python interpreter not the browser.
- We know that HTML is a static markup language, while Python is a dynamic programming language.
- Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.

**Django Template Configuration:**

- To configure the template system, we have to provide some entries in **settings.py** file.

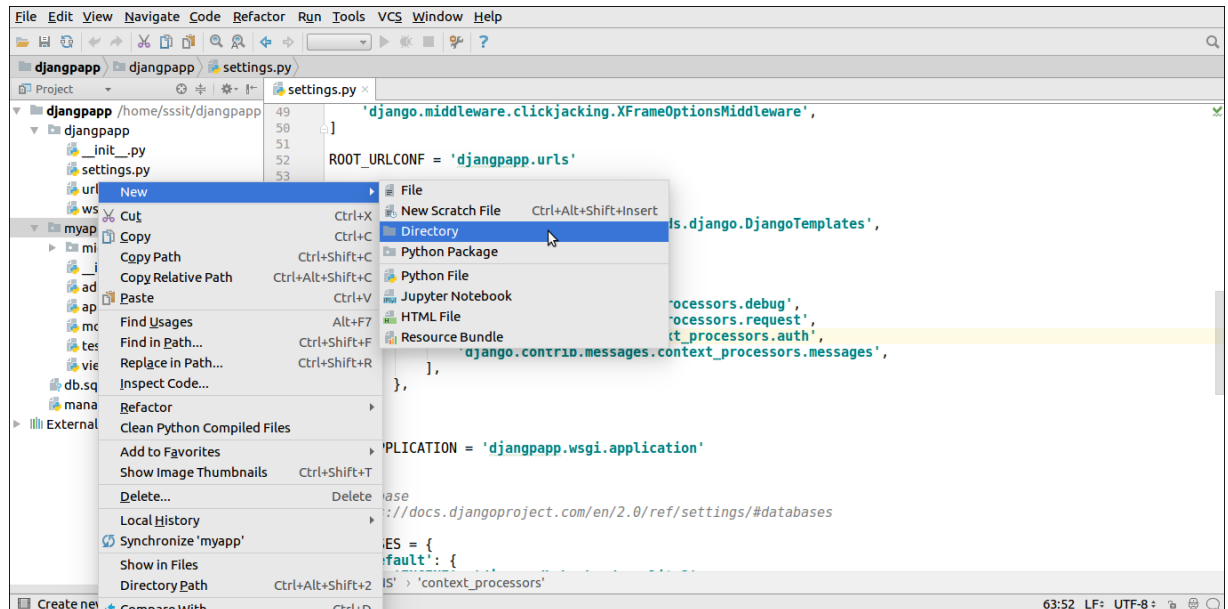
```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',
```

```
    ],  
    },  
},  
]
```

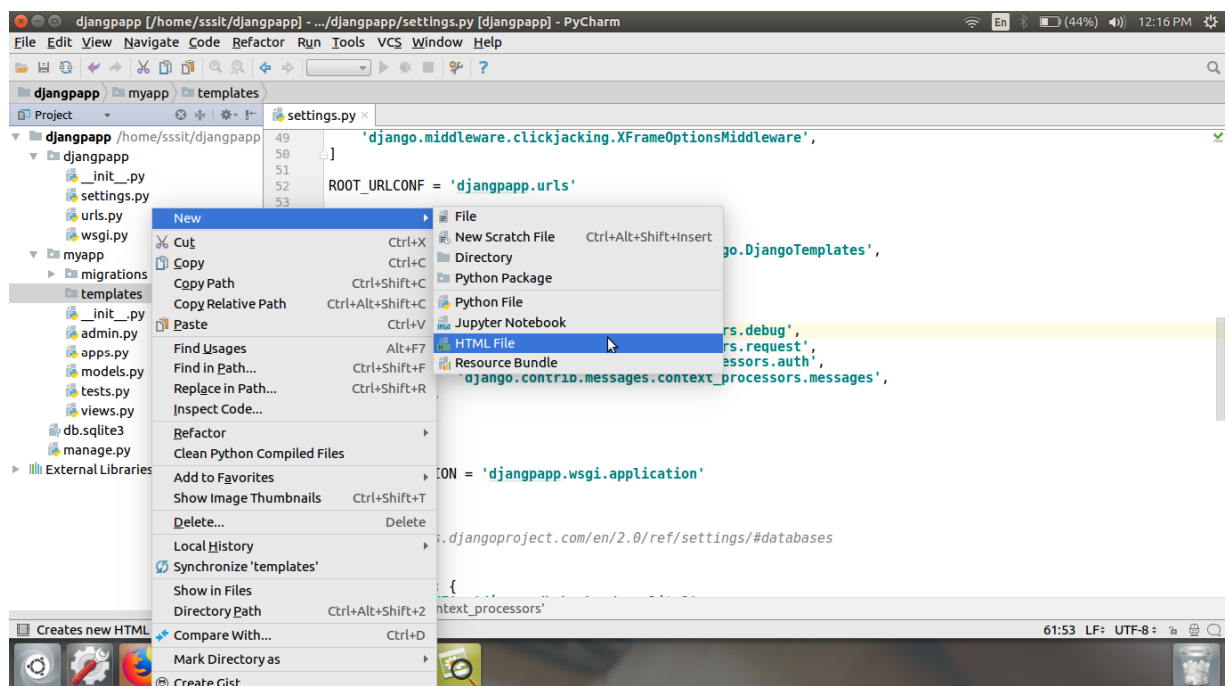
- Here, we mentioned that our template directory name is **templates**.
- By default, DjangoTemplates looks for a **templates** subdirectory in each of the INSTALLED\_APPS.

## Django Template Simple Example:

- First, create a directory **template** inside the project app as we did below.



- After that create a template **index.html** inside the created folder.



- Our template **index.html** contains the following code.

**// index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Index</title>
</head>
<body>
<h2>Welcome to Django!!!</h2>
</body>
</html>
```

**Loading Template:**

- To load the template, call `get_template()` method as we did below and pass template name.

**//views.py**

```
from django.shortcuts import render
#importing loading from django template
from django.template import loader
# Create your views here.
from django.http import HttpResponse
def index(request):
    template = loader.get_template('index.html') # getting our template
    return HttpResponse(template.render()) # rendering the template in HttpResponse
```

- Set a URL to access the template from the browser.

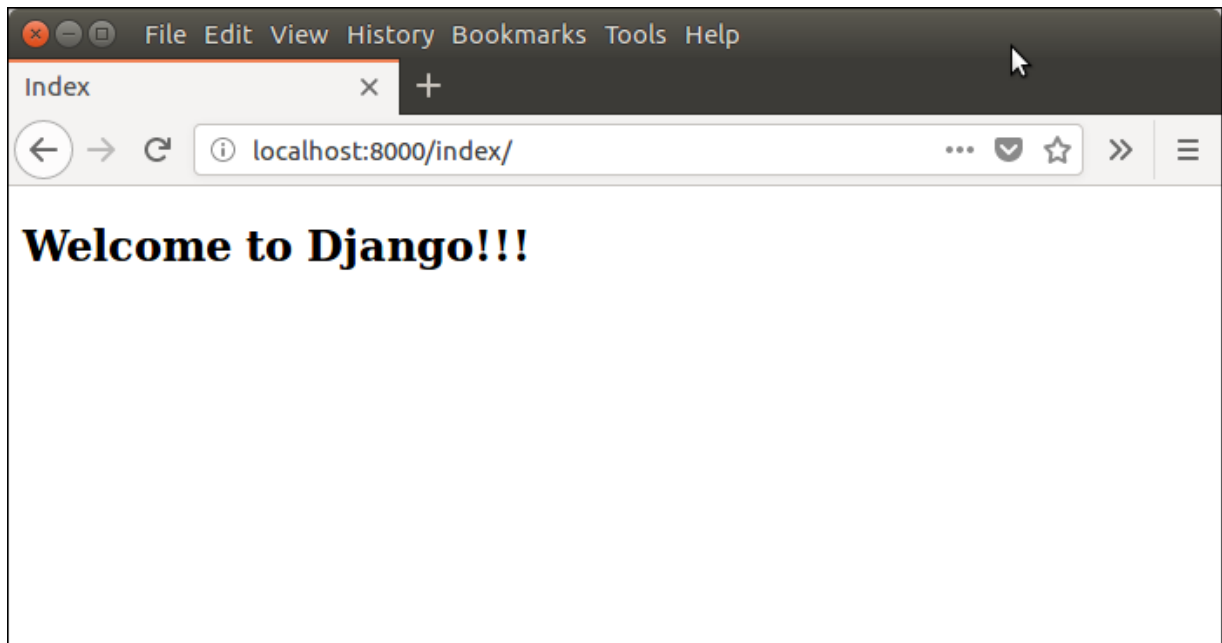
**//urls.py**

```
path('index/', views.index),
• Register app inside the INSTALLED_APPS
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp'
]
```

**Run Server:**

- Execute the following command and access the template by entering localhost:8000/index at the browser.

```
python manage.py runserver
```

**HTTP Client-Server Request – Response:**

- The client-server architecture includes two major components request and response.
- The Django framework uses client-server architecture to implement web applications.
- When a client requests for a resource, a HttpRequest object is created and correspond view function is called that returns HttpResponse object.
- To handle request and response, Django provides HttpRequest and HttpResponse classes. Each class has it's own attributes and methods.

**Django Request and Response:**

- The client-server architecture includes two major components request and response.
- The Django framework uses client-server architecture to implement web applications.
- When a client requests for a resource, a HttpRequest object is created and correspond view function is called that returns HttpResponse object.
- To handle request and response, Django provides HttpRequest and HttpResponse classes.
- Each class has it's own attributes and methods.
- Let's have a look at the HttpRequest class.

**Django HttpRequest:**

- This class is defined in the **django.http** module and used to handle the client request.
- Following are the attributes of this class.

**Django HttpRequest Attributes:**

Attribute	Description
HttpRequest.scheme	A string representing the scheme of the request (HTTP or HTTPS usually).
HttpRequest.body	It returns the raw HTTP request body as a byte string.
HttpRequest.path	It returns the full path to the requested page does not include the scheme or domain.
HttpRequest.path_info	It shows path info portion of the path.
HttpRequest.method	It shows the HTTP method used in the request.
HttpRequest.encoding	It shows the current encoding used to decode form submission data.
HttpRequest.content_type	It shows the MIME type of the request, parsed from the CONTENT_TYPE header.
HttpRequest.content_params	It returns a dictionary of key/value parameters included in the CONTENT_TYPE header.
HttpRequest.GET	It returns a dictionary-like object containing all given HTTP GET parameters.
HttpRequest.POST	It is a dictionary-like object containing all given HTTP POST parameters.
HttpRequest.COOKIES	It returns all cookies available.
HttpRequest.FILES	It contains all uploaded files.
HttpRequest.META	It shows all available Http headers.
HttpRequest.resolver_match	It contains an instance of ResolverMatch representing the resolved URL.

- And the following table contains the methods of HttpRequest class.



**Django HttpRequest Methods:**

Attribute	Description
HttpRequest.get_host()	It returns the original host of the request.
HttpRequest.get_port()	It returns the originating port of the request.
HttpRequest.get_full_path()	It returns the path, plus an appended query string, if applicable.
HttpRequest.build_absolute_uri( <i>location</i> )	It returns the absolute URI form of location.
HttpRequest.get_signed_cookie( <i>key</i> , <i>default=RAISE_ERROR</i> , <i>salt=""</i> , <i>max_age=None</i> )	It returns a cookie value for a signed cookie, or raises a django.core.signing.BadSignature exception if the signature is no longer valid.
HttpRequest.is_secure()	It returns True if the request is secure; that is, if it was made with HTTPS.
HttpRequest.is_ajax()	It returns True if the request was made via an XMLHttpRequest.

**Django HttpRequest Example:**

// views.py

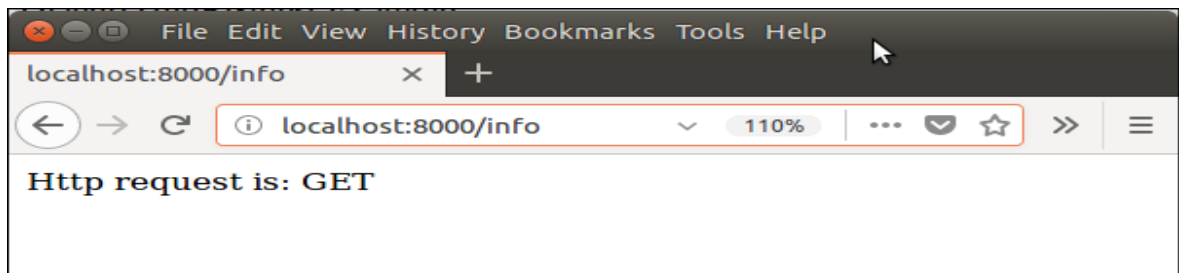
```
def methodinfo(request):  
    return HttpResponse("Http request is: "+request.method)
```

// urls.py

```
path('info',views.methodinfo)
```

- Start the server and get access to the browser. It shows the request method name at the browser.

**Output:**



### Django HttpResponse:

- This class is a part of django.http module.
- It is responsible for generating response corresponds to the request and back to the client.
- This class contains various attributes and methods that are given below.

### Django HttpResponse Attributes:

Attribute	Description
HttpResponse.content	A bytestring representing the content, encoded from a string if necessary.
HttpResponse.charset	It is a string denoting the charset in which the response will be encoded.
HttpResponse.status_code	It is an <b>HTTP status code</b> for the response.
HttpResponse.reason_phrase	The HTTP reason phrase for the response.
HttpResponse.streaming	It is false by default.
HttpResponse.closed	It is True if the response has been closed.

### Django HttpResponse Methods:

Method	Description
HttpResponse.__init__(content="", content_type=None, status=200, reason=None, charset=None)	It is used to instantiate an HttpResponse object with the given page content and content type.
HttpResponse.__setitem__(header, value)	It is used to set the given header name to the given value.
HttpResponse.__delitem__(header)	It deletes the header with the given name.
HttpResponse.__getitem__(header)	It returns the value for the given header name.
HttpResponse.has_header(header)	It returns either True or False based on a case-insensitive check for a header with the provided name.
HttpResponse.setdefault(header, value)	It is used to set default header.
HttpResponse.write(content)	It is used to create response object of file-like object.

<code>HttpResponse.flush()</code>	It is used to flush the response object.
<code>HttpResponse.tell()</code>	This method makes an <code>HttpResponse</code> instance a file-like object.
<code>HttpResponse.getvalue()</code>	It is used to get the value of <code>HttpResponse.content</code> .
<code>HttpResponse.readable()</code>	This method is used to create stream-like object of <code>HttpResponse</code> class.
<code>HttpResponse.seekable()</code>	It is used to make response object seekable.

- We can use these methods and attributes to handle the response in the Django application.

### **Concept of web framework and web application:**

- Django is an MVT web framework that is used to build web applications.
- The huge Django web-framework comes with so many “batteries included” that developers often get amazed as to how everything manages to work together.
- The principle behind adding so many batteries is to have common web functionalities in the framework itself instead of adding latter as a separate library.
- One of the main reasons behind the popularity of Django framework is the huge Django community.
- The community is so huge that a separate website was devoted to it where developers from all corners developed third-party packages including authentication, authorization, full-fledged Django powered CMS systems, e-commerce add-ons and so on.
- There is a high probability that what you are trying to develop is already developed by somebody and you just need to pull that into your project.