Introduction

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.*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. |

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

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

Django Installation

To install Django, first visit to **django official site (https://www.djangoproject.com)** and download django by clicking on the download section. Here, we will see various options to download The Django.

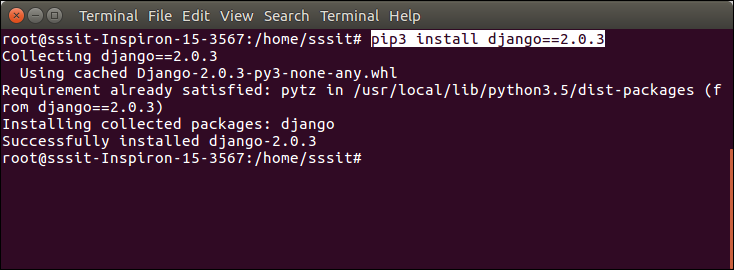
Django requires **pip** to start installation. Pip is a package manager system which is used to install and manage packages written in python. For Python 3.4 and higher versions **pip3** is used to manage packages.

In this tutorial, we are installing Django in Ubuntu operating system.

The complete installation process is described below. Before installing make sure **pip is installed** in local system.

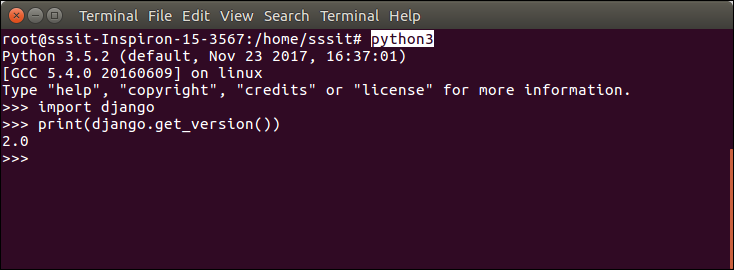
Here, we are installing Django using pip3, the installation command is given below.

1. $ pip3 install django==2.0.3



Verify Django Installation

After installing Django, we need to varify the installation. Open terminal and write **python3** and press enter. It will display python shell where we can verify django installation.



Look at the Django version displayed by the print method of the python. Well, Django is installed successfuly. Now, we can build Django web applications.

Django Project

In the previous topic, we have installed Django successfully. Now, we will learn step by step process to create a Django application.

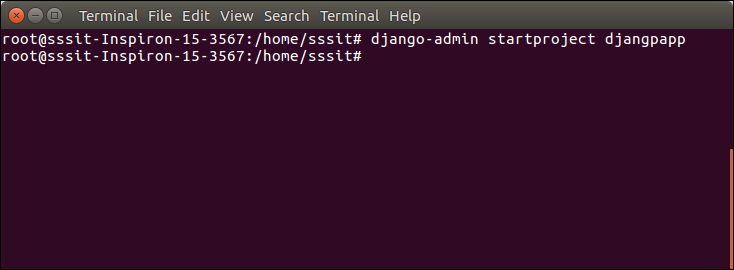
To create a Django project, we can use the following command. *projectname* is the name of Django application.

1. $ django-admin startproject projectname

Django Project Example

Here, we are creating a project **djangpapp** in the current directory.

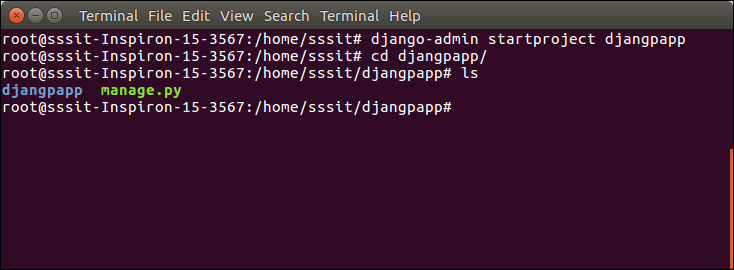
1. $ django-admin startproject djangpapp



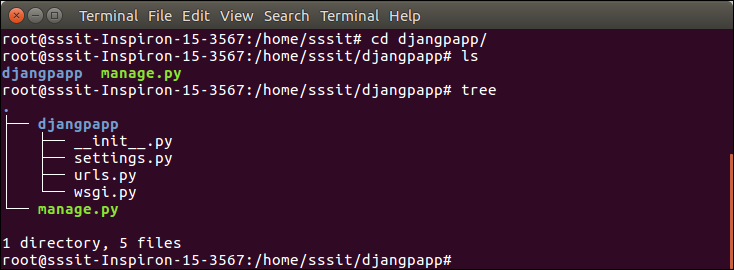
Locate into the Project

Now, move to the project by changing the directory. The Directory can be changed by using the following command.

1. cd djangpapp



To see all the files and subfolders of django project, we can use **tree** command to view the tree structure of the application. This is a utility command, if it is not present, can be downloaded via **apt-get install tree** command.



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 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.
* **\_\_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.

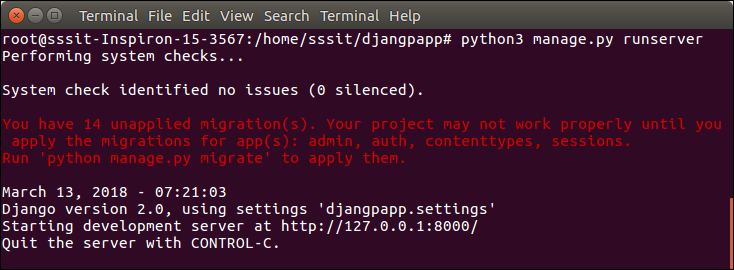
Initially, this project is a default draft which contains all the required files and folders.

Running the Django Project

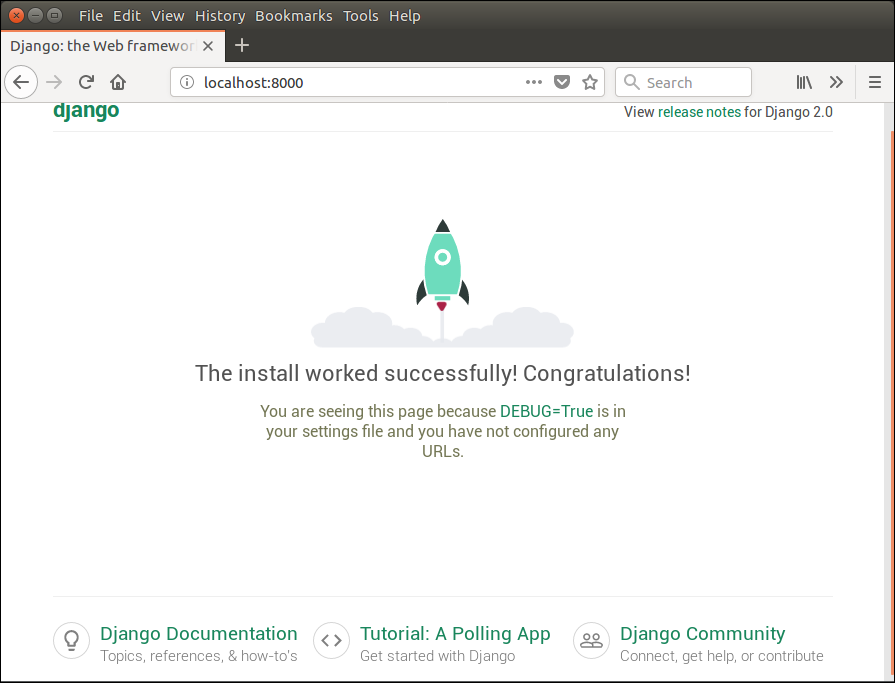
Django project has a built-in development server which is used to run application instantly without any external web server. It means we don't need of Apache or another web server to run the application in development mode.

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

1. $ python3 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.

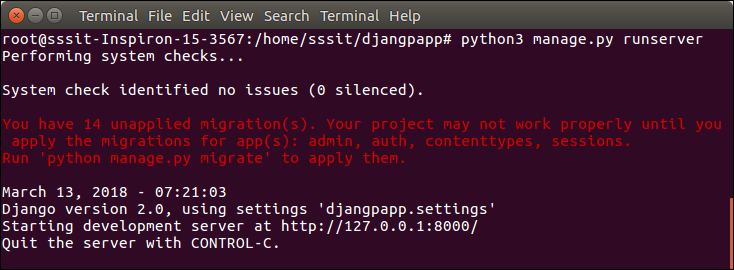


The application is running successfully. Now, we can customize it according to our requirement and can develop a customized web application.

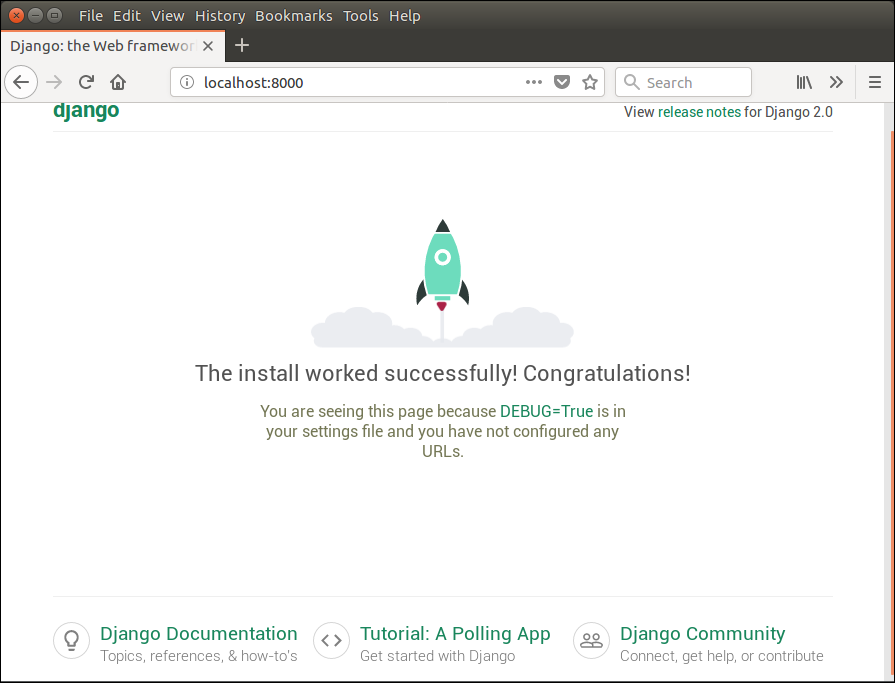
Django Configuration with Apache Web Server

Django uses its built-in development server to run the web application. To start this server, we can use **python manage.py runserver** command.

This command starts the server which runs on port 8000 and can be accessed at browser by entering *localhost:8000*. It shows a welcome page of the application.



And at browser, it can be accessed as below.

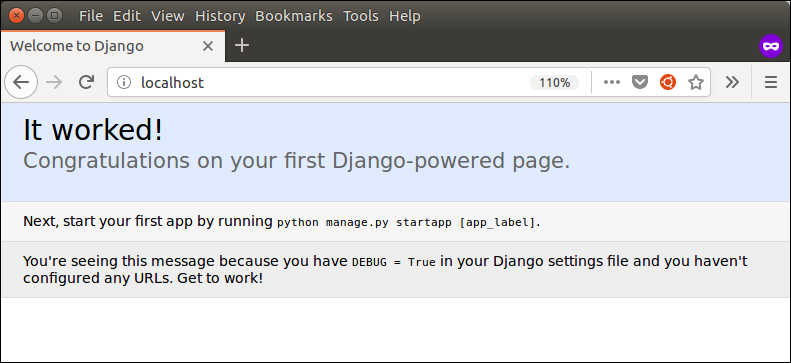


But if we want to run our application by using **apache server** rather than built-in development server, we need to configure **apache2.conf** file located at /**etc**/**apache** directory. Add the following code into this file.

**// apache2.conf**

1. WSGIScriptAlias / /var/www/html/django7/django7/wsgi.py
2. WSGIPythonPath /var/www/html/django7/
4. <Directory /var/www/html/django7>
5. <Files wsgi.py>
6. Require all granted
7. </Files>
8. </Directory>

After adding these lines, restart apache server by using the **service apache2 restart** command and then type **localhost** to the browser's address bar. This time, project will run on apache server rather than a built-in server. See, it shows the home page of the application.



Django Virtual Environment Setup

The virtual environment is an environment which is used by Django to execute an application. It is recommended to create and execute a Django application in a separate environment. Python provides a tool virtualenv to create an isolated Python environment. We will use this tool to create a virtual environment for our Django application.

To set up a virtual environment, use the following steps.

1. Install Package

First, install python3-venv package by using the following command.

$ apt-get install python3-venv

djanjo virtual environment setup install package

2. Create a Directory

$ mkdir djangoenv

After it, change directory to the newly created directory by using the cd djangoenv.

djanjo virtual environment setup Create a directory

3. Create Virtual Environment

$ python3 -m venv djangoenv

4. Activate Virtual Environment

After creating a virtual environment, activate it by using the following command.

$ source djangoenv/bin/activate

djanjo virtual environment setup activate virtual environment

Till here, the virtual environment has started. Now, we can use it to create Django application.

Install Django

Install Django in the virtual environment. To install Django, use the following command.

$ pip install django

djanjo virtual environment setup install django

Django has installed successfully. Now we can create a new project and build new applications in the separate environment.

Django Admin Interface

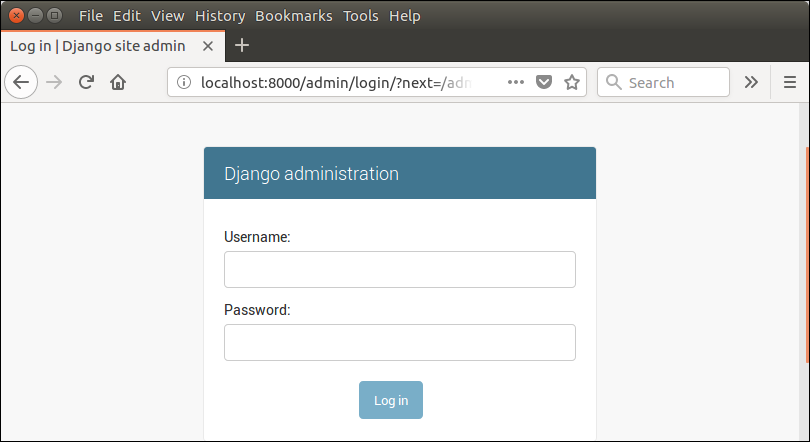
Django provides a built-in admin module which can be used to perform CRUD operations on the models. It reads metadata from the model to provide a quick interface where the user can manage the content of the application.

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.

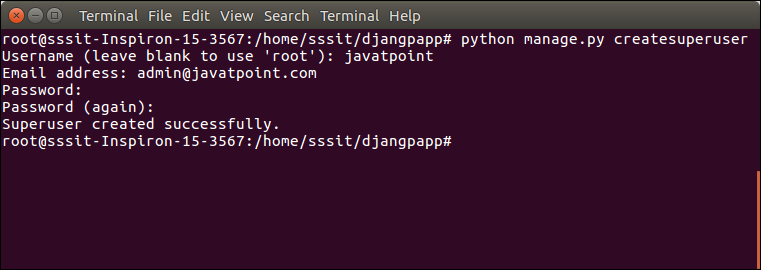
To access it at browser use '/**admin**/' at a local machine like **localhost:8000**/**admin**/ and it shows the following output:



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

Create an Admin User

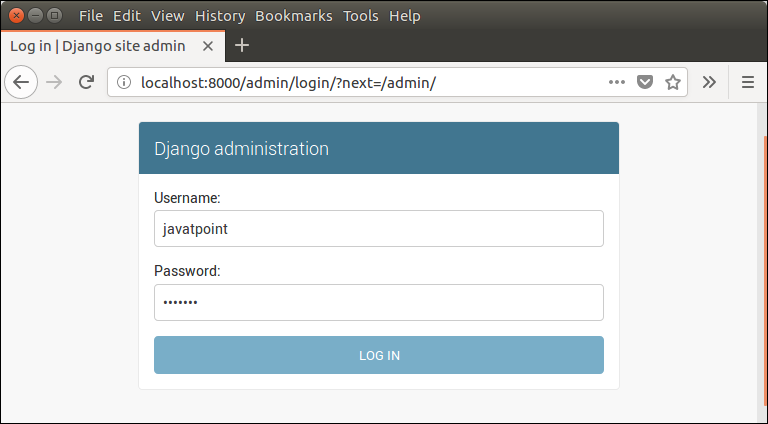
1. $ python3 managen.py createsuperuser



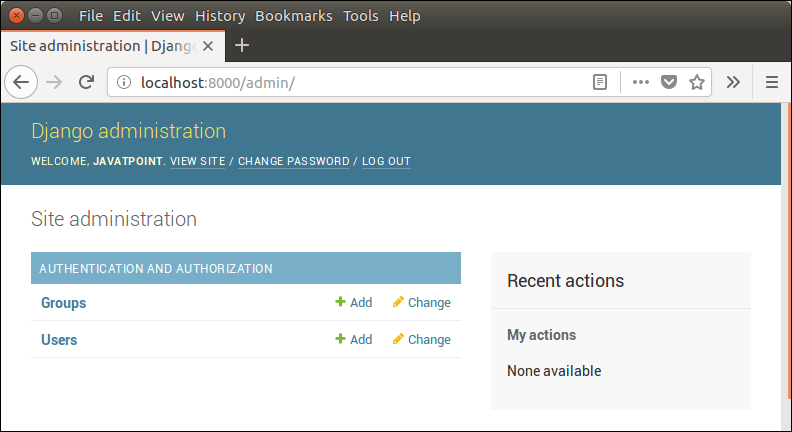
Now start development server and access admin login.

1. $ python3 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. The model registration process will be discussed in further chapters.

Django App

In the previous topics, 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.

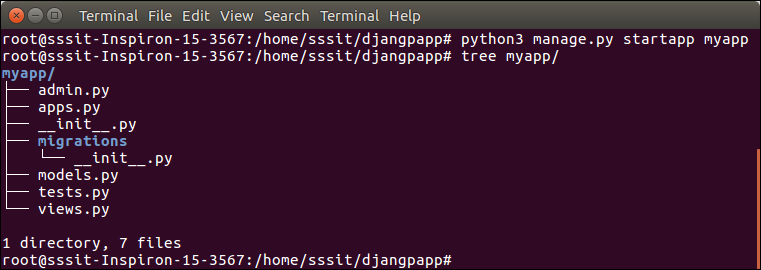
Creating an App

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

1. $ python3 manage.py startapp appname

Django App Example

1. $ python3 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**

1. from django.shortcuts **import** render
3. # Create your views here.
4. from django.http **import** HttpResponse
6. def hello(request):
7. **return** HttpResponse("<h2>Hello, Welcome to Django!</h2>")

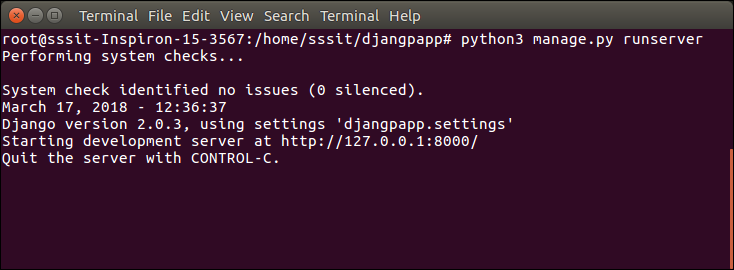
**// urls.py**

1. from django.contrib **import** admin
2. from django.urls **import** path
3. from myapp **import** views
5. urlpatterns = [
6. path('admin/', admin.site.urls),
7. path('hello/', views.hello),
8. ]

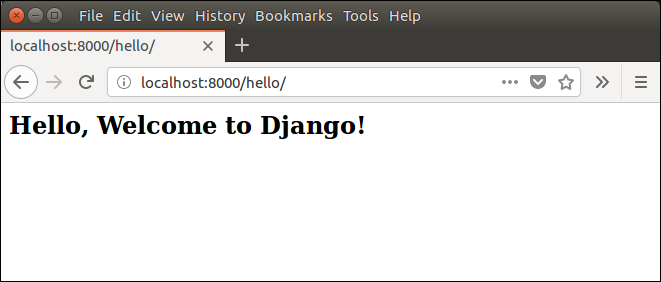
We have made changes in two files of the application. Now, let's run the it by using the following command. This command will start the server at port 8000.

Run the Application

1. $ python3 manage.py runserver



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



# 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 it?s 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**.

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**.

1. from django.db **import** models
3. **class** Employee(models.Model):
4. first\_name = models.CharField(max\_length=30)
5. 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.

1. CREATE TABLE appname\_employee (
2. "id" INT NOT NULL PRIMARY KEY,
3. "first\_name" varchar(30) NOT NULL,
4. "last\_name" varchar(30) NOT NULL
5. );

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,**

1. INSTALLED\_APPS = [
2. #...
3. 'appname',
4. #...
5. ]

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

1. first\_name = models.CharField(max\_length=50) # **for** creating varchar column
2. release\_date = models.DateField()                        # **for** creating date column
3. 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 mac\_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 allowed 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.

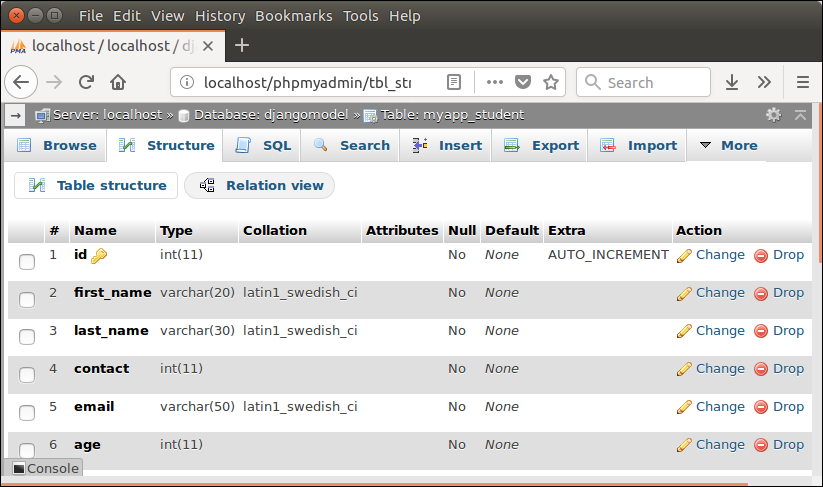
**//models.py**

1. **class** Student(models.Model):
2. first\_name = models.CharField(max\_length=20)
3. last\_name  = models.CharField(max\_length=30)
4. contact    = models.IntegerField()
5. email      = models.EmailField(max\_length=50)
6. age        = models.IntegerField()

After that apply migration by using the following command.

1. python3 manage.py makemigrations myapp

It will create a table **myapp\_student**. The table structure looks like the below.



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**

1. **import** datetime
2. # Create your views here.
3. from django.http **import** HttpResponse
4. def index(request):
5. now = datetime.datetime.now()
6. html = "<html><body><h3>Now time is %s.</h3></body></html>" % now
7. **return** HttpResponse(html)    # rendering the template in HttpResponse

Let's step through the code.

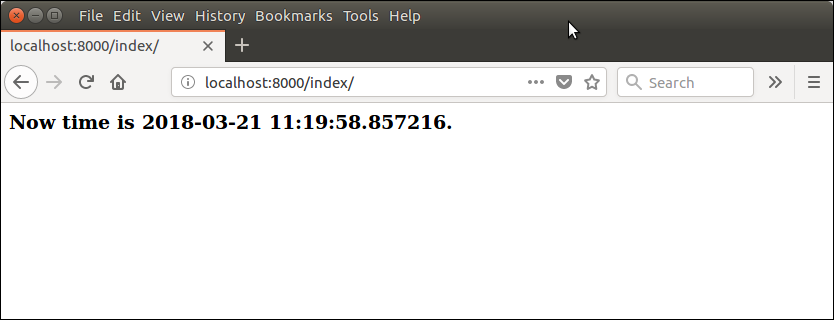
First, we will import DateTime library that provides a method to get current date and time and HttpResponse class.

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

1. path('index/', views.index),

**Output:**



Returning Errors

Django provides various built-in error classes that are the subclass of **HttpResponse** and use to show error message as HTTP response. Some classes are listed below.

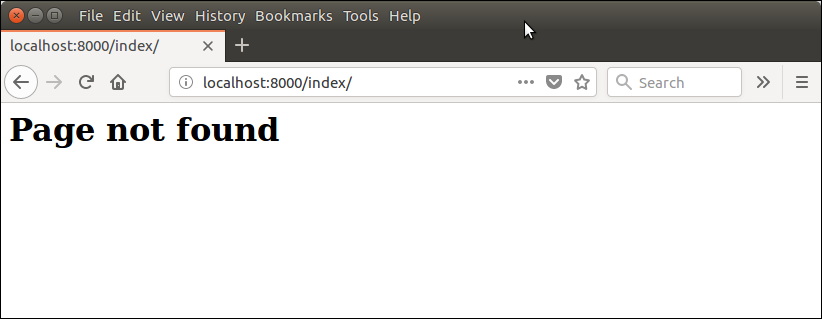
|  |  |
| --- | --- |
| **Class** | **Description** |
| class HttpResponseNotModified | It is used to designate that a page hasn't been modified since the user's last request (status code 304). |
| class HttpResponseBadRequest | It acts just like HttpResponse but uses a 400 status code. |
| class HttpResponseNotFound | It acts just like HttpResponse but uses a 404 status code. |
| class HttpResponseNotAllowed | It acts just like HttpResponse but uses a 410 status code. |
| HttpResponseServerError | It acts just like HttpResponse but uses a 500 status code. |

Django View Example

**// views.py**

1. from django.shortcuts **import** render
2. # Create your views here.
3. from django.http **import** HttpResponse, HttpResponseNotFound
4. def index(request):
5. a = 1
6. **if** a:
7. **return** HttpResponseNotFound('<h1>Page not found</h1>')
8. **else**:
9. **return** HttpResponse('<h1>Page was found</h1>') # rendering the template in HttpResponse

**Output:**



Django View HTTP Decorators

HTTP Decorators are used to restrict access to view based on the request method.

These decorators are listed in django.views.decorators.http and return a django.http.HttpResponseNotAllowed if the conditions are not met.

**Syntax**

require\_http\_methods(request\_method\_list)

Django Http Decorator Example

**//views.py**

1. from django.shortcuts **import** render
2. # Create your views here.
3. from django.http **import** HttpResponse, HttpResponseNotFound
4. from django.views.decorators.http **import** require\_http\_methods
5. @require\_http\_methods(["GET"])
6. def show(request):
7. **return** HttpResponse('<h1>This is Http GET request.</h1>')

This method will execute only if the request is an HTTP GET request.

**//urls.py**

1. from django.contrib **import** admin
2. from django.urls **import** path
3. from myapp **import** views
4. urlpatterns = [
5. path('admin/', admin.site.urls),
6. path('index/', views.index),
7. path('show/',  views.show),
8. ]

**Output:**

