Django Introduction

Django is a Python-based web framework which allows us to quickly develop robust web application without much third party installations. It comes with many built-in features—like user authentication, an admin panel and form handling—that help you build complex sites without worrying about common web development tasks.

**Key Features of Django**

* **Rapid Development**: Build fully featured web applications quickly.
* **Built-In Admin Interface**: Manage your app’s data easily through an automatically generated admin panel.
* **Database Flexibility**: Easily switch between databases like [SQLite](https://www.geeksforgeeks.org/introduction-to-sqlite/), [MySQL](https://www.geeksforgeeks.org/what-is-mysql/) or [PostgreSQL](https://www.geeksforgeeks.org/postgresql-tutorial/).
* **Extensible**: Thousands of additional packages are available to extend Django’s capabilities.
* **Scalability**: Designed to grow with your application.

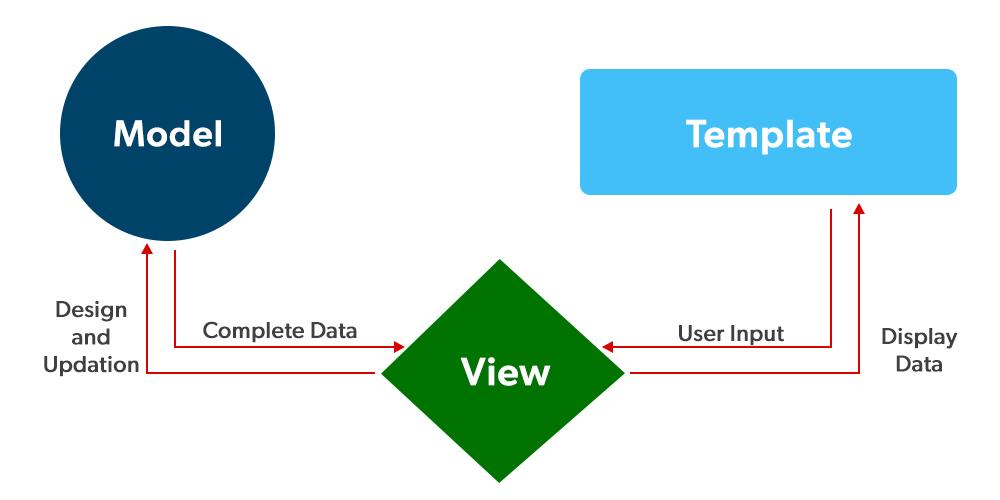
**Django Architecture: The MVT Pattern**

Django is based on MVT (Model-View-Template) architecture. MVT is a software design pattern for developing a web application. It's structure has the following three parts :

**1. Model:** Acts as the**data interface**. It defines the structure of your data and is usually backed by a database (e.g., MySQL, PostgreSQL).

**2. View:** A **Python function** or **class**that handles **web requests**. It interacts with the Model and renders a response, typically by passing data to a Template.

**3. Template:** Contains static HTML mixed with Django’s templating syntax. Templates are used to display dynamic data on the web page.

Django MVT

*To check more about Django's architecture, visit* [*Django Project MVT Structure*](https://www.geeksforgeeks.org/django-project-mvt-structure)

**Installing Django**

Follow these steps to set up Django on your system:

**1. Install Python 3:**

Download and install the latest Python 3 version from the official website.

**2. Install pip:**

Pip comes with recent versions of Python. Open the command prompt and run:

*pip --version*

**3. Set Up a Virtual Environment:**

This isolates your project’s dependencies.

*python -m virtualenv env\_site*

**4. Activate the Environment:**

Windows:

*cd env\_site\Scripts\activate*

Mac/Linux:

*source env\_site/bin/activate*

**5. Install Django:**

With your virtual environment active, run:

*pip install django*

**Create a Django Project and App**

**1. Create a Django Project**

Open the terminal and run:

*django-admin startproject projectName  
cd projectName*

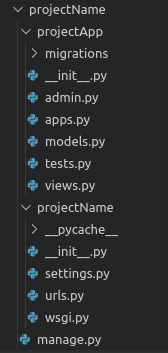
This creates a new folder named **projectName** containing your project settings and then changes the current working directory to it..

**2. Create a Django App**

Inside the project directory (where manage.py is located), run:

*python manage.py startapp projectApp*

This creates a new app named projectApp.

Now you can see your directory structure as under :

**3. Update INSTALLED\_APPS in Settings**

In projectName/settings.py, add your app to the INSTALLED\_APPS list:

*# Application definition*

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'projectApp'

]

We have finally created an app but to render it using urls we need to include the app in our main project so that urls redirected to that app can be rendered.

**4. Include App URLs in the Main Project**

Edit **projectName/urls.py**to include your app’s URLs:

**from** **django.contrib** **import** admin

**from** **django.urls** **import** path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include("projectApp.urls")), *# Routes all other URLs to projectApp*

]

Make sure you create a urls.py file inside **projectApp**to define your app’s routes.

**Using Django’s MVT Model**

Once your project is set up:

* **Models**: Define your data structure in **projectApp/models.py**.
* **Views**: Create functions or class-based views in projectApp/views.py to handle requests.
* **Templates**: Create HTML templates in a templates folder (either in the app or at the project level) and use Django’s templating language to render dynamic content.

**Python and Django**

**Introduction**

**Python** is a powerful, high-level programming language known for its simplicity, readability, and versatility. **Django** is a web development framework built **on top of Python**. The two are tightly connected — **Django cannot work without Python**.

**Nature of Relationship**

| **Aspect** | **Python** | **Django** |
| --- | --- | --- |
| Role | Programming Language | Web Framework |
| Purpose | General-purpose coding | Rapid web development |
| Dependency | Independent | Built using Python |
| Syntax | Provides basic syntax | Uses Python syntax and concepts |
| Execution | Can run standalone scripts | Needs Python to run its commands and apps |
|  |  |  |

**Why Django Uses Python**

1. **Ease of Use**: Python's clean syntax helps Django be readable and efficient.
2. **Rapid Development**: Python's simplicity aligns with Django’s goal of rapid web development.
3. **Large Ecosystem**: Django benefits from Python's massive standard library and third-party packages.
4. **OOP Support**: Django uses classes and objects (Python's OOP features) for models, views, and forms.

**Code Relationship Example**

Python code used in Django:

python

CopyEdit

# A Django Model (Python Class)

from django.db import models

class Student(models.Model):

name = models.CharField(max\_length=100)

age = models.IntegerField()

This is pure Python syntax, but written in Django's ORM structure.

**Running Django = Running Python**

Every Django command uses Python under the hood. Example:

bash

CopyEdit

python manage.py runserver

This uses Python’s interpreter to run Django’s internal logic.

**Learning Path**

To learn Django effectively, **Python basics are a prerequisite**:

1. Data types, functions, classes
2. File I/O, error handling
3. Virtual environments and pip (Python package manager)

**Real-Life Analogy**

Django is like a car (framework), but Python is the engine.  
You can’t drive the car unless the engine (Python) works.

**Djano Model View Template**

Django follows the **MVT (Model-View-Template)** architectural pattern, which is a variation of the traditional MVC (Model-View-Controller) design pattern used in web development. This pattern separates the application into three main components:

**1. Model**

Model acts as the data layer of your application. It defines the structure of your database and handles data-related logic. It typically represents your database tables and is responsible for querying, inserting, updating, and deleting data. Django models are usually backed by relational databases like MySQL, PostgreSQL, SQLite, etc.

*To check more, visit -* [*Django Models*](https://www.geeksforgeeks.org/django-models/)

**2. View**

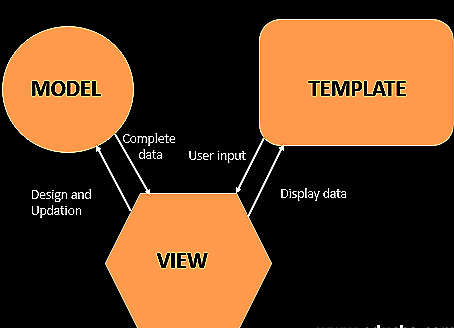
View handles the business logic and user interface rendering. It processes user requests, interacts with models to fetch data, and passes that data to templates for display. In Django, views are Python functions or classes that return HTTP responses.

*To check more, visit -* [*Django Views*](https://www.geeksforgeeks.org/views-in-django-python/)*.*

**3. Template**

Template is responsible for presenting the data to the user. It contains the static parts of the HTML and special template syntax (like Django Template Language) to dynamically insert data. Templates usually consist of HTML, CSS, and JavaScript.

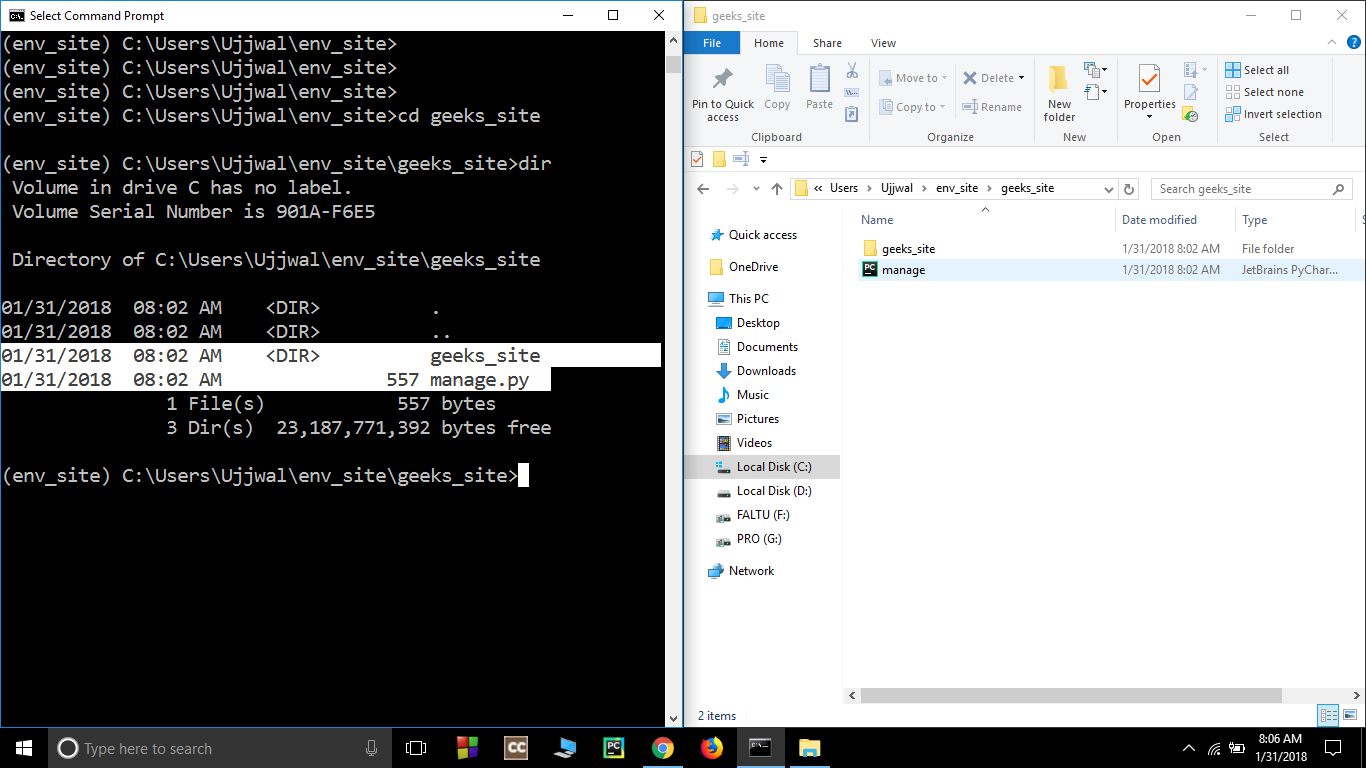
*To check more, visit -* [*Django Templates*](https://www.geeksforgeeks.org/django-templates/)



**Project Structure**

A Django Project when initialized contains basic files by default such as manage.py, view.py, etc. A simple project structure is enough to create a single-page application.

Here are the major files and their explanations. Inside the geeks\_site folder ( project folder ) there will be the following files:

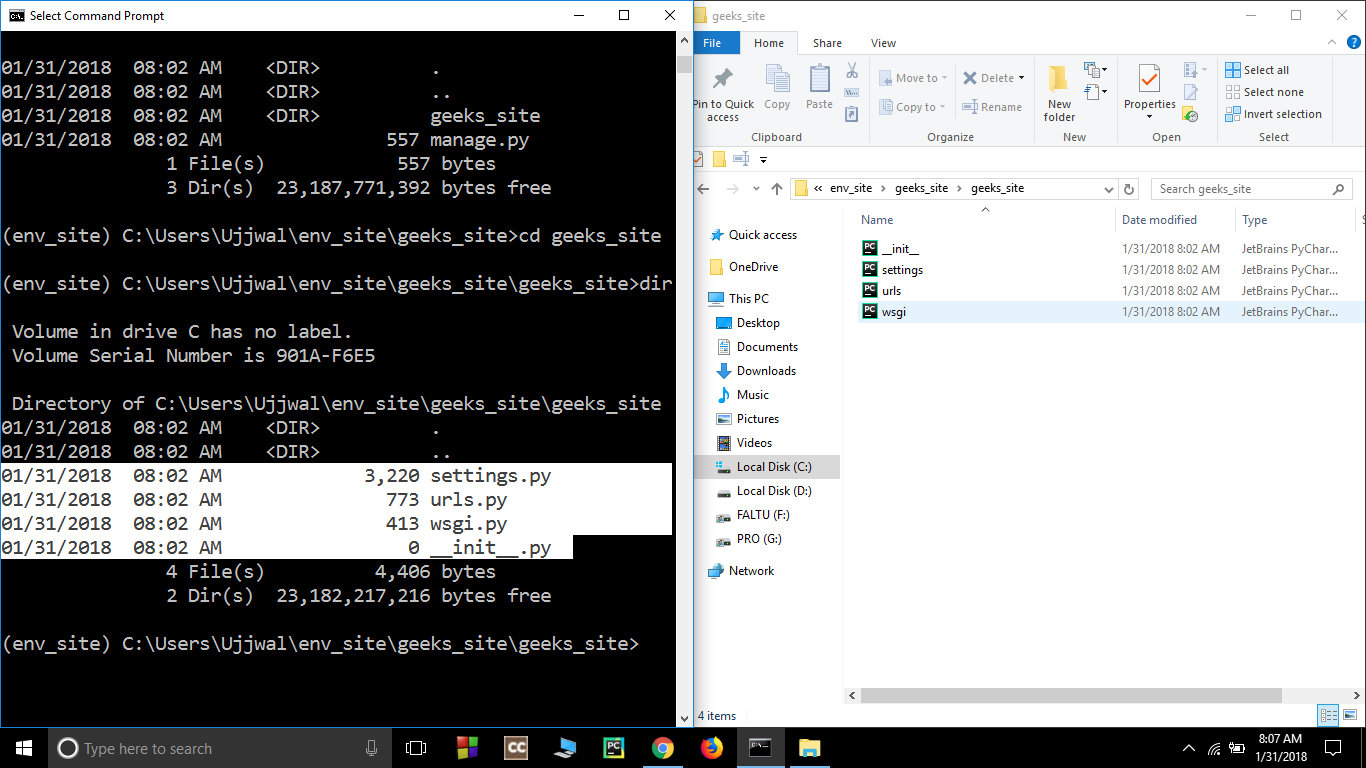


**Explanation of Key Files and Folders**

**1. manage.py:**This file is used to interact with your project via the command line(start the server, sync the database... etc). For getting the full list of commands that can be executed by manage.py type this code in the command window-

*$ python manage.py help*

**2. folder ( geeks\_site ):**This folder contains all the packages of your project. Initially, it contains four files -



* **\_init\_.py -**It is a python package. It is invoked when the package or a module in the package is imported. We usually use this to execute package initialization code, for example for the initialization of package-level data.
* **settings.py -**As the name indicates it contains all the website settings. In this file, we register any applications we create, the location of our static files, database configuration details, etc.
* **urls.py -**In this file, we store all links of the project and functions to call.
* **wsgi.py -**This file is used in deploying the project in WSGI. It is used to help your Django application communicate with the webserver.

**Installation of Django**

Installing and setting up Django is a straightforward process. Below are the step-by-step instructions to install Django and set up a new Django project on your system.

***Prerequisites****: Before installing Django, make sure you have* [*Python installed on your system*](https://www.geeksforgeeks.org/download-and-install-python-3-latest-version/)*.*

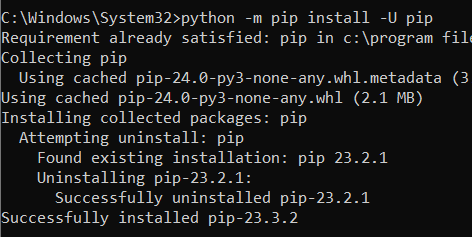
**How to Install Django?**

To Install Django in Linux and Mac is similar, here I am showing it in Windows for Linux and Mac just open the terminal in place of the command prompt and go through the following commands.

**Step 1: Install Pip**

Open the command prompt and enter the following command-

*python -m pip install -U pip*

Install Pip

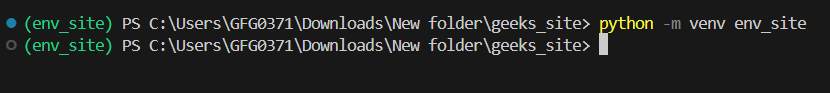
**Step 2: Set Virtual environment**

Setting up the virtual environment will allow you to edit the dependency which generally your system wouldn't allow. Follow these steps to set up a virtual environment-

**Step 3: Create virtual environment in Django**

We should first go the directory where we want to create the [virtual environment](https://www.geeksforgeeks.org/python-virtual-environment/) then we type the following command to create virtual environment in django.

*python -m venv env\_site*

Create Virtual Environment

then we need to activate virtual environment in django

**Step 4: Activate the virtual environment**

Run the activation script located in the bin directory within the virtual environment folder

* **For Windows:**

*.\env\_site\Scripts\activate.ps1*

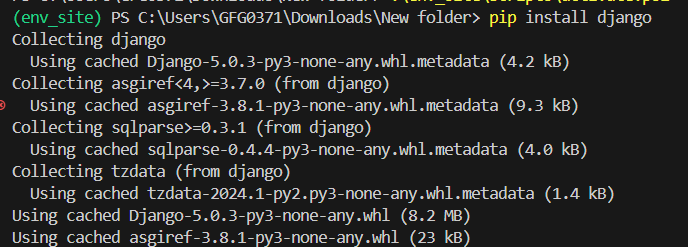
* **For MacOs/Linux:**

*source env\_site/bin/activate*

**Step 5: Install Django**

Install django by giving following command

*pip install django*

Install Django

**Django Setup**

Once Django is installed, we can start to create a new Django project.

**Step 1: Start a new Django Project**

Start a project by following command-

*django-admin startproject geeks\_site*

4Start a new Project Django

**Step 2: Navigate to the Project Directory**

Change directory to geeks\_site

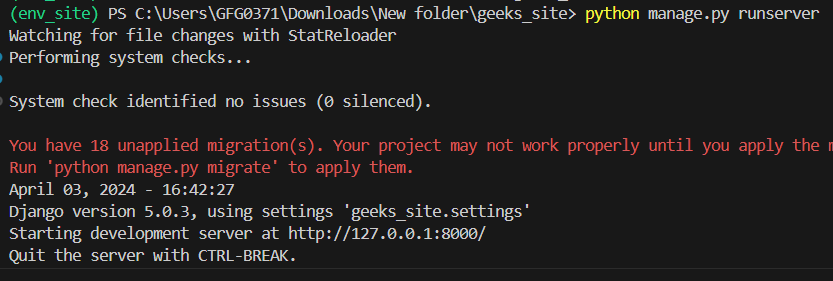
*cd geeks\_site*

4

**Step 3: Start the server**

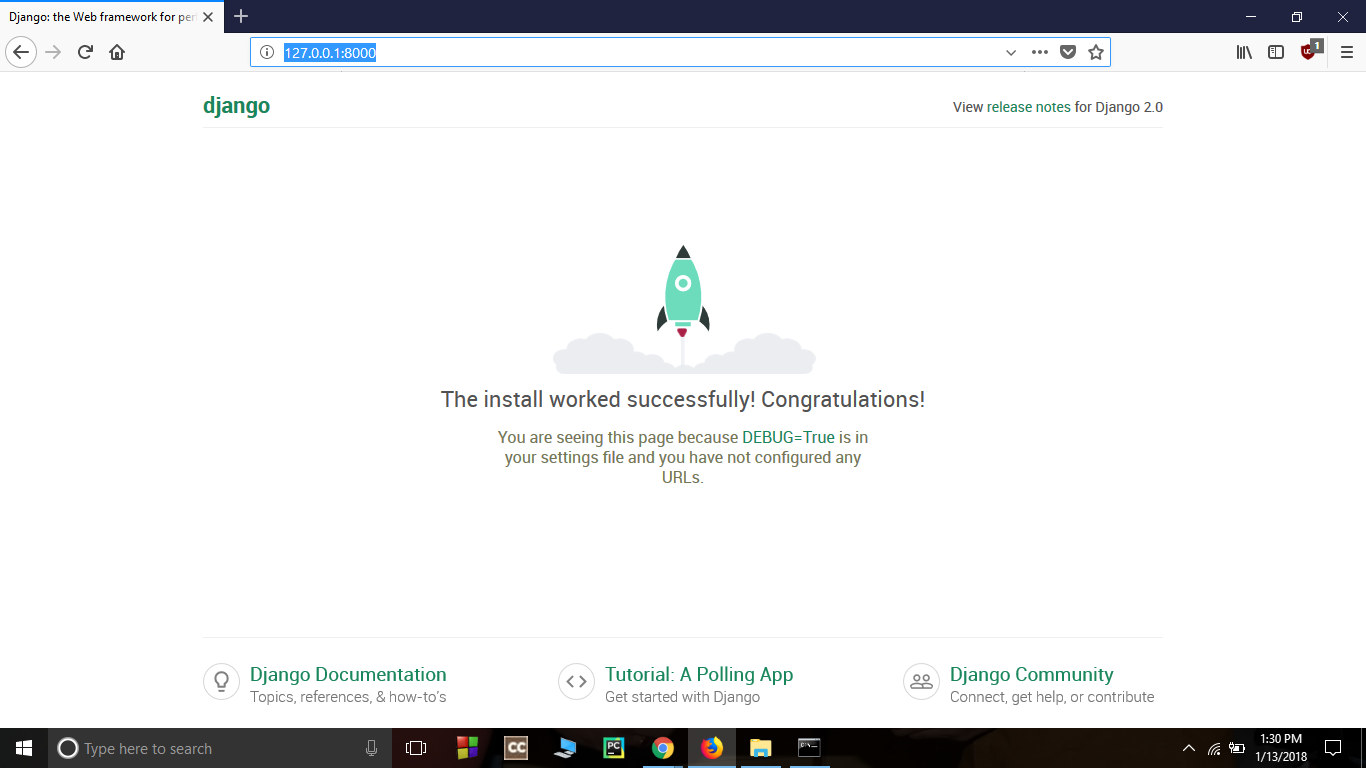
Start the server by typing following command in cmd-

*python manage.py runserver*

Start Django Server

**Step 4: Verify Server Status**

To check whether server is running or not go to web browser and enter **http://127.0.0.1:8000/** as URL.



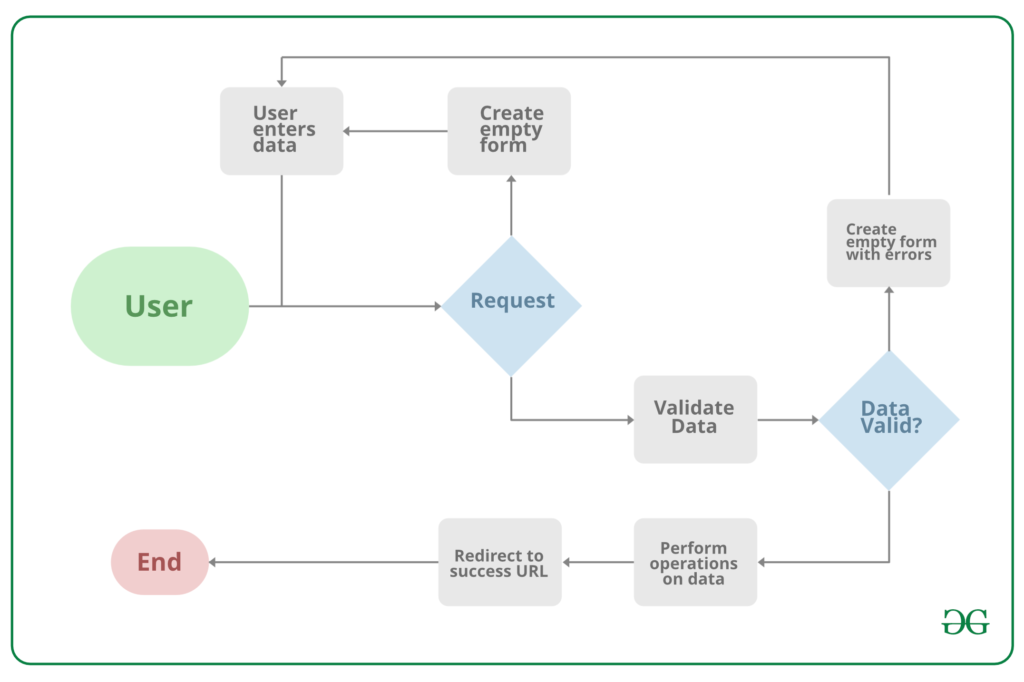
**Form Classes**

Django Forms are used to gather input from users, validate that input, and process it, often saving the data to the database. For example, when registering a user, a form collects information like name, email, and password.

Django automatically maps form fields to corresponding HTML input elements. It also handles:

* **Preparing**data for rendering
* **Creating**HTML forms
* **Processing** submitted form data securely

While you can build forms using raw **HTML**, Django Forms simplify form creation and validation, making your code cleaner and more secure.



***Note*** *that all types of work done by forms in Django can be done with advanced HTML stuff, but Django makes it easier and efficient especially the validation part. Once you get hold of forms in Django you will just forget about HTML forms.*

**Syntax:**

Django form fields are defined much like Django model fields:

*field\_name = forms.FieldType(\*\*options)*

Befor creating forms, ensure that a django project and app is already set up, refer to the following article to learn how to do it:

* [*How to Create a Basic Project using MVT in Django?*](https://www.geeksforgeeks.org/how-to-create-a-basic-project-using-mvt-in-django/)
* [*How to Create an App in Django ?*](https://www.geeksforgeeks.org/how-to-create-an-app-in-django/)

**Creating a Simple Django Form**

Creating a form in Django is completely similar to creating a model, one needs to specify what fields would exist in the form and of what type. For example, to input, a registration form one might need First Name (CharField), Roll Number (IntegerField), and so on.

**Syntax:**

*from django import forms  
  
class FormName(forms.Form):  
# each field would be mapped as an input field in HTML  
field\_name = forms.Field(\*\*options)*

To create a form, create a **forms.py**file in you app folder:

**from** **django** **import** forms

**class** **InputForm**(forms.Form):

first\_name = forms.CharField(max\_length=200)

last\_name = forms.CharField(max\_length=200)

roll\_number = forms.IntegerField(help\_text="Enter 6 digit roll number")

password = forms.CharField(widget=forms.PasswordInput())

**Render Django Forms**

Django form fields have several built-in methods to ease the work of the developer but sometimes one needs to implement things manually for customizing User Interface(UI). A form comes with 3 in-built methods that can be used to render Django form fields.

* [{{ form.as\_table }}](https://www.geeksforgeeks.org/form-as_table-render-django-forms-as-table/) will render them as table cells wrapped in <tr> tags
* [{{ form.as\_p }}](https://www.geeksforgeeks.org/form-as_p-render-django-forms-as-paragraph/) will render them wrapped in <p> tags
* [{{ form.as\_ul }}](https://www.geeksforgeeks.org/form-as_ul-render-django-forms-as-list/) will render them wrapped in <li> tags

To render this form into a view, move to views.py and create a home\_view as below.

**from** **django.shortcuts** **import** render

**from** **.forms** **import** InputForm

*# Create your views here.*

**def** home\_view(request):

context ={}

context['form']= InputForm()

**return** render(request, "home.html", context)

In view, one needs to just create an instance of the form class created above in forms.py. Now let's edit templates > **home.html**

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

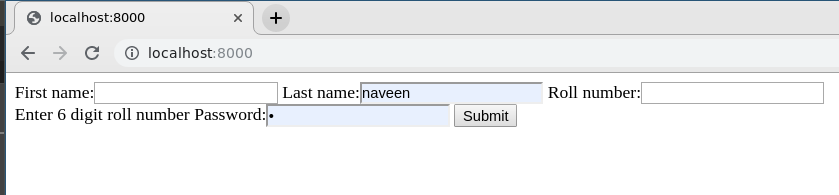
{% csrf\_token %}

{{form }}

<**input** type="submit" value=Submit">

</**form**>

Now, visit http://localhost:8000/



*To check how to use the data rendered by Django Forms visit* [*Render Django Form Fields*](https://www.geeksforgeeks.org/render-django-form-fields-manually/)

**Create Django Form from Models**

Django ModelForm is a class that is used to directly convert a model into a Django form. If you’re building a database-driven app, chances are you’ll have forms that map closely to Django models. Now when we have our project ready, create a model in geeks/models.py,

*# import the standard Django Model*

*# from built-in library*

**from** **django.db** **import** models

*# declare a new model with a name "GeeksModel"*

**class** **GeeksModel**(models.Model):

*# fields of the model*

title = models.CharField(max\_length = 200)

description = models.TextField()

last\_modified = models.DateTimeField(auto\_now\_add = **True**)

img = models.ImageField(upload\_to = "images/")

*# renames the instances of the model*

*# with their title name*

**def** \_\_str\_\_(self):

**return** self.title

To create a form directly for this model, dive into **geeks/forms.py**and Enter the following code:

*# import form class from django*

**from** **django** **import** forms

*# import GeeksModel from models.py*

**from** **.models** **import** GeeksModel

*# create a ModelForm*

**class** **GeeksForm**(forms.ModelForm):

*# specify the name of model to use*

**class** **Meta**:

model = GeeksModel

fields = "\_\_all\_\_"

Now visit **http://127.0.0.1:8000/**

***Read Next:*** [*Python | Form validation using django*](https://www.geeksforgeeks.org/python-form-validation-using-django/)

**Django Forms Data Types and Fields List**

The most important part of a form and the only required part is the list of fields it defines. Fields are specified by class attributes. Here is a list of all Form Field types used in Django

| **Name** | **Class** | **HTML Input** |
| --- | --- | --- |
| [BooleanField](https://www.geeksforgeeks.org/booleanfield-django-forms/) | class BooleanField(\*\*kwargs) | CheckboxInput |
| [CharField](https://www.geeksforgeeks.org/charfield-django-forms/) | class CharField(\*\*kwargs) | TextInput |
| [ChoiceField](https://www.geeksforgeeks.org/choicefield-django-forms/) | class ChoiceField(\*\*kwargs) | Select |
| [TypedChoiceField](https://www.geeksforgeeks.org/typedchoicefield-django-forms/) | class TypedChoiceField(\*\*kwargs) | Select |
| [DateField](https://www.geeksforgeeks.org/datefield-django-forms/) | class DateField(\*\*kwargs) | DateInput |
| [DateTimeField](https://www.geeksforgeeks.org/datetimefield-django-forms/) | class DateTimeField(\*\*kwargs) | DateTimeInput |
| [DecimalField](https://www.geeksforgeeks.org/decimalfield-django-forms/) | class DecimalField(\*\*kwargs) | NumberInput when Field.localize is False, else TextInput |
| [DurationField](https://www.geeksforgeeks.org/durationfield-django-forms/) | class DurationField(\*\*kwargs) | TextInput |
| [EmailField](https://www.geeksforgeeks.org/emailfield-django-forms/) | class EmailField(\*\*kwargs | EmailInput |
| [FileField](https://www.geeksforgeeks.org/filefield-django-forms/) | class FileField(\*\*kwargs) | ClearableFileInput |
| [FilePathField](https://www.geeksforgeeks.org/filepathfield-django-forms/) | class FilePathField(\*\*kwargs) | Select |
| [FloatField](https://www.geeksforgeeks.org/floatfield-django-forms/) | class FloatField(\*\*kwargs) | NumberInput when Field.localize is False, else TextInput |
| [ImageField](https://www.geeksforgeeks.org/imagefield-django-forms/) | class ImageField(\*\*kwargs) | ClearableFileInput |
| [IntegerField](https://www.geeksforgeeks.org/integerfield-django-forms/) | class IntegerField(\*\*kwargs) | NumberInput when Field.localize is False, else TextInput |
| [GenericIPAddressField](https://www.geeksforgeeks.org/genericipaddressfield-django-forms/) | class GenericIPAddressField(\*\*kwargs) | TextInput |
| [MultipleChoiceField](https://www.geeksforgeeks.org/multiplechoicefield-django-forms/) | class MultipleChoiceField(\*\*kwargs) | SelectMultiple |
| [TypedMultipleChoiceField](https://www.geeksforgeeks.org/typedmultiplechoicefield-django-forms/) | class TypedMultipleChoiceField(\*\*kwargs) | SelectMultiple |
| [NullBooleanField](https://www.geeksforgeeks.org/nullbooleanfield-django-forms/) | class NullBooleanField(\*\*kwargs) | NullBooleanSelect |
| [RegexField](https://www.geeksforgeeks.org/regexfield-django-forms/) | class RegexField(\*\*kwargs) | TextInput |
| [SlugField](https://www.geeksforgeeks.org/slugfield-django-models/) | class SlugField(\*\*kwargs) | TextInput |
| [TimeField](https://www.geeksforgeeks.org/timefield-django-forms/) | class TimeField(\*\*kwargs) | TimeInput |
| [URLField](https://www.geeksforgeeks.org/urlfield-django-forms/) | class URLField(\*\*kwargs) | URLInput |
| [UUIDField](https://www.geeksforgeeks.org/uuidfield-django-forms/) | class UUIDField(\*\*kwargs) | TextInput |

**Core Field Arguments**

Core Field arguments are the arguments given to each field for applying some constraint or imparting a particular characteristic to a particular Field. For example, adding an argument required = False to CharField will enable it to be left blank by the user. Each Field class constructor takes at least these arguments. Some Field classes take additional, field-specific arguments, but the following should always be accepted:

| **Field Options** | **Description** |
| --- | --- |
| [required](https://www.geeksforgeeks.org/required-django-form-field-validation/) | By default, each Field class assumes the value is required, so to make it not required you need to set required=False |
| [label](https://www.geeksforgeeks.org/label-django-form-field-validation/) | The label argument lets you specify the “human-friendly” label for this field. This is used when the Field is displayed in a Form. |
| [label\_suffix](https://www.geeksforgeeks.org/label-django-form-field-validation/) | The label\_suffix argument lets you override the form’s label\_suffix on a per-field basis. |
| [widget](https://www.geeksforgeeks.org/django-form-field-custom-widgets/) | The widget argument lets you specify a Widget class to use when rendering this Field. See Widgets for more information. |
| [help\_text](https://www.geeksforgeeks.org/help_text-django-form-field-validation/) | The help\_text argument lets you specify descriptive text for this Field. If you provide help\_text, it will be displayed next to the Field when the Field is rendered by one of the convenience Form methods. |
| [error\_messages](https://www.geeksforgeeks.org/error_messages-django-form-field-validation/) | The error\_messages argument lets you override the default messages that the field will raise. Pass in a dictionary with keys matching the error messages you want to override. |
| [validators](https://www.geeksforgeeks.org/django-form-field-custom-widgets/) | The validators argument lets you provide a list of validation functions for this field. |
| **localize** | The localize argument enables the localization of form data input, as well as the rendered output. |
| [disabled](https://www.geeksforgeeks.org/disabled-django-form-field-validation/)**.** | The disabled boolean argument, when set to True, disables a form field using the disabled HTML attribute so that it won’t be editable by users. |

Validation

Django works on an MVT pattern. So there is a need to create data models (or tables). For every table, a model class is created.   
Suppose there is a form that takes **Username**, **gender**, and **text** as input from the user, the task is to validate the data and save it.  
In django this can be done, as follows:

**from** **django.db** **import** models

*# model named Post*

**class** **Post**(models.Model):

Male = 'M'

FeMale = 'F'

GENDER\_CHOICES = (

(Male, 'Male'),

(FeMale, 'Female'),

)

*# define a username field with bound max length it can have*

username = models.CharField( max\_length = 20, blank = **False**,

null = **False**)

*# This is used to write a post*

text = models.TextField(blank = **False**, null = **False**)

*# Values for gender are restricted by giving choices*

gender = models.CharField(max\_length = 6, choices = GENDER\_CHOICES,

default = Male)

time = models.DateTimeField(auto\_now\_add = **True**)

After creating the data models, the changes need to be reflected in the database to do this run the following command:

python manage.py makemigrations

Doing this compiles the models and if it didn't find any errors then, it creates a file in the migration folder. Later run the command given below to finally reflect the changes saved onto the migration file onto the database.

python manage.py migrate

Now a form can be created. Suppose that the username length should not be less than 5 and post length should be greater than 10. Then we define the Class **PostForm** with the required validation rules as follows: 

**from** **django.forms** **import** ModelForm

**from** **django** **import** forms

**from** **formValidationApp.models** **import** \*

*# define the class of a form*

**class** **PostForm**(ModelForm):

**class** **Meta**:

*# write the name of models for which the form is made*

model = Post

*# Custom fields*

fields =["username", "gender", "text"]

*# this function will be used for the validation*

**def** clean(self):

*# data from the form is fetched using super function*

super(PostForm, self).clean()

*# extract the username and text field from the data*

username = self.cleaned\_data.get('username')

text = self.cleaned\_data.get('text')

*# conditions to be met for the username length*

**if** len(username) < 5:

self.\_errors['username'] = self.error\_class([

'Minimum 5 characters required'])

**if** len(text) <10:

self.\_errors['text'] = self.error\_class([

'Post Should Contain a minimum of 10 characters'])

*# return any errors if found*

**return** self.cleaned\_data

Till now, the data models and the Form class are defined. Now the focus will be on how these modules, defined above, are actually used.  
First, run on localhost through this command 

python manage.py runserver

Open **http://localhost:8000/** in the browser, then it's going to search in the **urls.py** file, looking for ' ' path  
**urls.py** file is as given below: 

**from** **django.contrib** **import** admin

**from** **django.urls** **import** path, include

**from** **django.conf.urls** **import** url

**from** **django.shortcuts** **import** HttpResponse

**from** **.** **import** views

urlpatterns = [

path('', views.home, name ='index'),

]

Basically, this associates the ' ' url with a function ***home*** which is defined in **views.py** file.  
**views.py file**: 

**from** **.models** **import** Post

**from** **.forms** **import** PostForm

**from** **.import** views

**from** **django.shortcuts** **import** HttpResponse, render, redirect

**def** home(request):

*# check if the request is post*

**if** request.method =='POST':

*# Pass the form data to the form class*

details = PostForm(request.POST)

*# In the 'form' class the clean function*

*# is defined, if all the data is correct*

*# as per the clean function, it returns true*

**if** details.is\_valid():

*# Temporarily make an object to be add some*

*# logic into the data if there is such a need*

*# before writing to the database*

post = details.save(commit = **False**)

*# Finally write the changes into database*

post.save()

*# redirect it to some another page indicating data*

*# was inserted successfully*

**return** HttpResponse("data submitted successfully")

**else**:

*# Redirect back to the same page if the data*

*# was invalid*

**return** render(request, "home.html", {'form':details})

**else**:

*# If the request is a GET request then,*

*# create an empty form object and*

*# render it into the page*

form = PostForm(**None**)

**return** render(request, 'home.html', {'form':form})

**home.html** template file 

{% load bootstrap3 %}

{% bootstrap\_messages %}

<!DOCTYPE html>

<**html** lang="en">

<**head** >

<**title**>Basic Form</**title**>

<**meta** charset="utf-8" />

<**meta** name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<**link** rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<**script** src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">

</**script**>

<**script** src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">

</**script**>

</**head**>

<**body** style="padding-top: 60px;background-color: #f5f7f8 !important;">

<**div** class="container">

<**div** class="row">

<**div** class="col-md-4 col-md-offset-4">

<**h2**>Form</**h2**>

<**form** action="" method="post"><**input** type='hidden'/>

{%csrf\_token %}

{% bootstrap\_form form %}

<!-This is the form variable which we are passing from the function

of home in views.py file. That's the beauty of Django we

don't need to write much codes in this it'll automatically pass

all the form details in here

->

<**div** class="form-group">

<**button** type="submit" class="btn btn-default ">

Submit

</**button**>

</**div**>

</**form**>

</**div**>

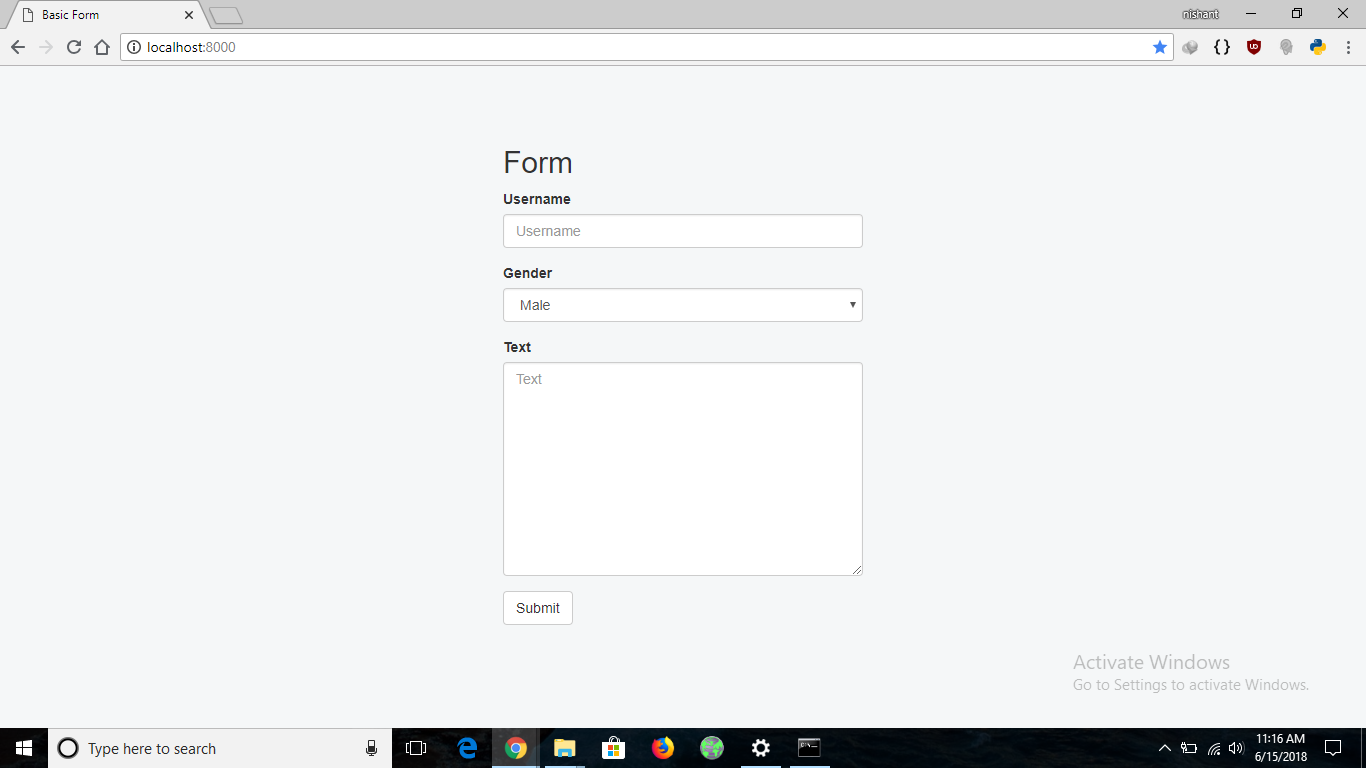
</**div**>

</**div**>

</**body**>

</**html**>

Opening **http://localhost:8000/** in the browser shows the following,



If a form with a username of length less than 5 is submitted, it gives an error at the time of submission and similarly for the Post Text filled as well. The following image shows how the form behaves on submitting invalid form data.

