Introduction

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source. It is based on MVT (Model View Template) design pattern.

Advantages

Ridiculously fast.

Django was designed to help developers take applications from concept to completion as quickly as possible.

Fully loaded.

Django includes dozens of extras you can use to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks — right out of the box.

Reassuringly secure.

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

Exceedingly scalable.

Some of the busiest sites on the planet use Django's ability to quickly and flexibly scale to meet the heaviest traffic demands.

Incredibly versatile.

Companies, organizations and governments have used Django to build all sorts of things — from content management systems to social networks to scientific computing platforms.

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

- Instagram
- Mozilla
- Disqus

- Pinterest
- Bitbucket
- The Washington Times

Django Installation

Django requires **pip** to start installation. Pip is a package manager system which is used to install and manage packages written in python.

To check python version that is installed in your system, open a command prompt. Go to start ->run->cmd and type the following:

python - -version

To check the version of pip installed type the following in command prompt:

pip - -version

pip is generally preinstalled in the newer versions of Python.

To install django type the following in command prompt:

pip install django

Creating a project

To create a project use the command:

django-admin startproject <projectname>

Let's look at what startproject created:

```
<projectname>/
    manage.py
    <projectname>/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

To navigate to the project folder:

```
cd <projectname>
```

To start the development server

```
python manage.py runserver
```

To check if the server is running:

Open any browser and type the url:

```
localhost:8000
```

It should open up the home page of the Django server

Creating the app

```
python manage.py startapp <appname>
```

That'll create a directory polls, which is laid out like this:

```
<appname>/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

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.

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 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. Django template engine is used to separate the design from the python code and allows us to build dynamic web pages.

To create a view function and render a template

- Go to views.py add the following code:

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

- Go to djangoapp folder and create a folder called templates within it

- Write the following code in index.html

```
<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <title>Index</title>

        {% load static %}


</head>

<body>

<h2>Welcome to Django app!!!</h2>

<h3>my name is {{name}}</h3>

<img src="{% static '/Desert.jpeg' %}"  alt="sorry"  width="300px"
height="700px">

</body>

</html>
```

- The above code uses an image.
- For that go to your app folder djangoapp and create a folder called static within it

- Within the static folder create a sub folder called images:

  Keep all your images within the images folder
- To render templates do the following:

- Go to your project folder djangop and then again djangop folder:

- Go to settings.py file

- Add the name of your app 'djangoapp' as given below:

```
INSTALLED_APPS = [
   'djangoapp',
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
   'django.contrib.sessions',
   'django.contrib.messages',
   'django.contrib.staticfiles',
]
```

- Also make the following changes in settings.py in TEMPLATES section:

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

- Go to urls.py and add the following:

```
from djangoapp import views

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

- Start the server with the command:

```
python manage.py runserver
```

- Run the file with the url:

```
localhost:8000/index
```

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.

Example:

```
from django.db import models

# Create your models here.

class Student(models.Model):

    fname=models.CharField(max_length=30)

    lname=models.CharField(max_length=30)

    email=models.EmailField()

    dob=models.DateField()

    age=models.IntegerField()
```

This model will create a table into the database. 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.

Django Database 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.

- makemigrations : It is used to create a migration file that contains code for the tabled schema of a model.
- migrate : It creates table according to the schema defined in the migration file.
- sqlmigrate : It is used to show a raw SQL query of the applied migration.
- showmigrations : It lists out all the migrations and their status.

To create a migration for this model, use the following command. It will create a migration file inside the migration folder.

```
python manage.py makemigrations
```

This migration file contains the code in which a Migration class is created that contains the name and fields of student table.

After creating a migration, migrate it so that it reflects the database permanently. The migrate command is given below.

```
python manage.py migrate
```

Django Admin Interface

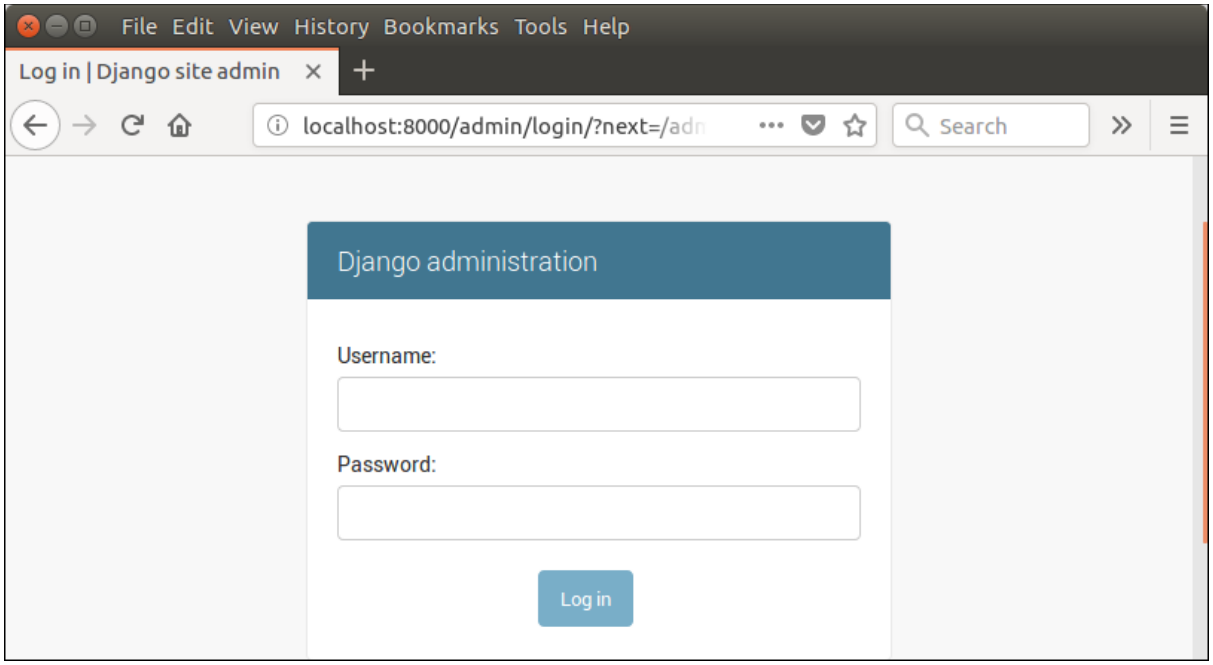
Django provides a built-in admin module which can be used to perform CRUD operations on the models. It reads metadata from the model to provide a quick interface where the user can manage the content of the application.

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

python3 manage.py createsuperuser

After creating the superuser, start development server and access admin login.

python3 manage.py runserver

Provide created username and password and login.

After login successfully, it shows the admin interface.

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

Register the Student model with admin

Open admin.py file within the app folder:

```
from django.contrib import admin

from djangoapp.models import Student

# Register your models here.

admin.site.register(Student)
```

DJANGO MODEL FORMS

It is a class which is used to create an HTML form by using the Model. It is an efficient way to create a form without writing HTML code.

Django automatically does it for us to reduce the application development time. For example, suppose we have a model containing various fields, we don't need to repeat the fields in the form file.

For this reason, Django provides a helper class which allows us to create a Form class from a Django model.

Create a file called form.py and save it within your app folder:

Example:

form.py

```
from djangoapp.models import Student
from django import forms
class StudentForm(forms.ModelForm):
    class Meta:
        model=Student
        fields="__all__"
```

DJANGO FORMS

Django provides a Form class which is used to create HTML forms. It describes a form and how it works and appears.

It is similar to the **ModelForm** class that creates a form by using the Model, but it does not require the Model.

Each field of the form class map to the HTML form **<input>** element and each one is a class itself, it manages form data and performs validation while submitting the form.

Example:

```
class CustomerForm(forms.Form):
    cid=forms.IntegerField()
    cfname=forms.CharField(label='Enter first name',max_length=30)
    clname=forms.CharField(label='Enter last name',max_length=30)
```

DJANGO CRUD OPERATIONS ON DATABASE

CREATE A STUDENT RECORD

We will using the Student model and the Student ModelForm created earlier.

Go to views.py add the following code:

```
def createstudent(request):
    if request.method=="POST":
        form=StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect("/showstudents")
    else:
        form=StudentForm()
        return render(request,"createstudent.html",{'sform':form})
```

Within templates folder of your app create a template called createstudent.html

```
<!DOCTYPE html>
<html>
<head>
<title>CREATE STUDENT FORM</title>
</head>
<body>
<form method="post">
{% csrf_token  %}
<table border="1px">
```

```
{{sform.as_table}}
</table>
<button type="submit">SAVE</button>
</form>
<br><br>
<a href="/showstudents/">Back to List</a>


</body>
</html>
```

- Go to urls.py and add the following:

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

- Start the server with the command:

```
python manage.py runserver
```

- Run the file with the url:

```
localhost:8000/ createstudent/
```

SHOW ALL STUDENT RECORDS

Go to views.py add the following code:

```
def showstudents(request):
    students=Student.objects.all()
    return render(request,"showstudents.html",{'students':students})
```

- Create a template called showstudents.html within templates folder:

```
<!DOCTYPE html>
<html>
<head>
<title>STUDENTS LIST</title>
</head>
<body>
<table border="1px" align="center" cellpadding="10px" cellspacing="0px">
<tr>
<th>FIRST NAME</th>
<th>LAST NAME</th>
<th>EMAIL</th>
<th>DATE OF BIRTH</th>
<th>AGE</th>
<th colspan="2">ACTION</th>
</tr>
{% for student in students %}
<tr>
<td>{{student.fname}}</td>
<td>{{student.lname}}</td>
<td>{{student.email}}</td>
<td>{{student.dob}}</td>
```

```
<td>{{student.age}}</td>
<td><a href="/edit/{{student.id}}">Edit</a></td>
<td><a href="/delete/{{student.id}}">Delete</a></td>
</tr>
{% endfor %}
</table>
<br>
<center><a href="/createstudent/">Create New</a></center>
</body>
</html>
```

- Go to urls.py and add the following:

path(‘showstudents /', views. showstudents)

- Start the server with the command:

python manage.py runserver

- Run the file with the url:

localhost:8000/ showstudents /

DELETE A STUDENT RECORD

In views.py:

```
def delete(request,id):
    student=Student.objects.get(id=id)
    student.delete()
    return redirect("/showstudents")
```

- In urls.py add the URL mapping and then run the file with the url:

localhost:8000/ showstudents /

UPDATE A STUDENT RECORD

In views.py:

```
def edit(request,id):
    student=Student.objects.get(id=id)
    if request.method=="POST":
        form=StudentForm(request.POST,instance=student)
        if form.is_valid():
            form.save()
            return redirect("/showstudents")
    else:
        form=StudentForm()
```

```
        return
render(request,"editstudent.html",{'sform':form,'student':student})
```

- Create a template called editstudent.html within templates folder:

```
<!DOCTYPE html>
<html>
<head>
<title>EDIT STUDENT FORM</title>
</head>
<body>
<form method="post" action="/edit/{{student.id}}/">
{% csrf_token  %}
<table border="1px">
<tr>
<th>FIRST NAME</th>
<td><input type="text" name="fname" id="id_fname" value="{{student.fname}}"></td>
</tr>

<tr>
<th>LAST NAME</th>
<td><input type="text" name="lname" id="id_lname" value="{{student.lname}}"></td>
</tr>

<tr>
<th>EMAIL</th>
<td><input type="text" name="email" id="id_email" value="{{student.email}}"></td>
```

```
</tr>


<tr>
<th>DATE OF BIRTH</th>
<td><input type="date" name="dob" id="id_dob" value="{{student.dob}}"></td>
</tr>


<tr>
<th>AGE</th>
<td><input type="number" name="age" id="id_age" value="{{student.age}}"></td>
</tr>


</table>
<button type="submit">UPDATE</button>
<br><br>
<a href="/showstudents/">Back to List</a>


</body>
</html>
```

- In urls.py add the URL mapping and then run the file with the url: localhost:8000/ showstudents /

DJANGO SESSION

A session is a mechanism to store information on the server side during the interaction with the web application.

In Django, by default session stores in the database and also allows file-based and cache based sessions. It is implemented via a piece of middleware and can be enabled by using the following code.

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

```
def set_session(request):
    request.session["name"]="Ajay"#creates a session variable called name
    request.session["city"]="Kolkata"#creates a session variable called city
    return HttpResponse("Session is set")
```

```
def get_session(request):
    name=request.session["name"]#fetches the session value
    city=request.session["city"]#fetches the session value
    return HttpResponse("Name:" + name + "City:"+city)
```

DJANGO COOKIE

A cookie is a small piece of information which is stored in the client browser. It is used to store user's data in a file permanently (or for the specified time).

Cookie has its expiry date and time and removes automatically when gets expire. Django provides built-in methods to set and fetch cookie.

The **set_cookie()** method is used to set a cookie and **get()** method is used to get the cookie.

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

In views.py:

```
def setcookie(request):
    response=HttpResponse("Cookie set")
    response.set_cookie("mycookie","abcd")
    #mycookie is cookie name,abcd is cookie value
    return response

def getcookie(request):
    cookievalue=request.COOKIES["mycookie"]#fetches cookie value
    return HttpResponse("Cookie value is "+cookievalue)
```

DJANGO PDF GENERATION

To generate PDF, we will install **ReportLab** Python PDF library that creates customized dynamic PDF.

pip install reportlab

In views.py:

```
from reportlab.pdfgen import canvas
from django.http import HttpResponse


def getpdf(request):
   response=HttpResponse(content_type='application/pdf')
```

```
    response['Content-Disposition']='attachment;filename="sample.pdf"'
    c=canvas.Canvas(response)
    c.setFont("Times-Roman",42)
    c.drawString(100,700,"Hello user!")
    c.showPage()
    c.save()
    return response
```

DJANGO CSV

Django uses Python's built-in CSV library to create Dynamic CSV (Comma Separated Values) file. We can use this library in our project's view file.

```
import csv
def getcsv(request):
    response=HttpResponse(content_type='text/csv')
    response['Content-Disposition']='attachment;filename="student.csv"'
    writer=csv.writer(response)
    writer.writerow(['Id','Name','Marks'])
    writer.writerow(['100','priya','90'])
    writer.writerow(['101','puja','92'])
    return response
```

DJANGO AUTHENTICATION

Overview

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks

Steps for login/registration:

Step 1:

Go to URLS .py and add the following import

from django.contrib.auth.views import LoginView

Add the following url to the list of urls:

path('login/',LoginView.as_view())

Step 2:

Go to the templates folder within your app folder:

Create a folder called registration inside it.

Inside the registration folder create a template file called login.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <title>Login</title>
</head>
<body>
<form method="POST" action="/check/">
      {% csrf_token %}
      {{ form.as_p }}
      <button type="submit">Sign In</button>
</form>

<br>
<a href="/customreg/">New User?Click here to Sign Up</a>
</body>
</html>
```

Step 3

Run the file with the url:

localhost:8000/login

Step 4

To create the registration section:

Go to models.py and add the following import

from django.contrib.auth.models import User

Create the following model class:

```
class UserData(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    bio = models.TextField(max_length=500, blank=True)
    gender = models.CharField(max_length=30)
    dob = models.DateField()
    location=models.CharField(max_length=30)
```

Step 5

Stop the server if running by using Ctrl +C in command prompt

Then perform database migration:

python manage.py makemigrations

python manage.py migrate

Step 6

Register the model in admin.py

Add the following import to admin.py

from djangoapp.models import UserData

Then add the following :

admin.site.register(UserData)

Step 7

Go to form.py

Add the following lines to import:

```
from djangoapp.models import UserData
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
```

Create the following forms:

```
class UserForm(UserCreationForm):
    class Meta:
        model = User
        fields = ('username','first_name', 'last_name',
'email','password1','password2')
```

```
class RegForm(forms.ModelForm):
   dob=forms.DateField(label='Date of Birth')
   choices=[('male','Male'),
       ('female','Female')]

   gender = forms.ChoiceField(choices=choices, widget=forms.RadioSelect)

   class Meta:
      model=UserData
      fields=('bio','gender','dob','location')
```

Step 8

Go to views.py

Add the following function:

```
def customreg(request):
   if request.method=="POST":
      user=UserForm(request.POST)
      form=RegForm(request.POST)
      if user.is_valid() and form.is_valid():
         profile = form.save(commit=False)
         profile.user = request.user
         user.save()
```

```
            profile.save()
            return redirect("/login/")
    else:
        user=UserForm()
        form=RegForm()
    return
render(request,"registration/customreg.html",{'form':form,'user':user})
```

Step 9

Go to registration folder within templates folder and create the customreg.html template:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
</head>
<body>
<form method="POST">
        {% csrf_token %}
                                {{ user.as_p }}
        {{ form.as_p }}
```

```
    <button type="submit">Sign Up</button>
</form>
<br>
<a href="/login/">Click here to login</a>
</body>
</html>
```

Step 10

Add the URL mapping in urls.py

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

Step 11

Start the server with the command:

python manage.py runserver

Run the file with the url:

localhost:8000/login

Click on the new user link, it will open up the sign up form

On successful sign up it will redirect to login page

To check login:

Step 1:

Go to views.py and add the following import:

```
from django.contrib.auth import authenticate,login,logout
from django.contrib.auth.decorators import login_required
```

Then add the following function:

```
def check(request):
    username=request.POST['username']
    password=request.POST['password']
    user=authenticate(request,username=username,password=password)
    if user is not None:
        login(request,user)#logs in the user
        return redirect("/home")
    else:
        return redirect("/login")
```

Step 2

In views.py add the next function:

```
@login_required
def home(request):
   username=request.user.username
   return render(request,"home.html",{'username':username})
```

Step 3

Go to templates folder and create the template home.html

```
<!DOCTYPE html>
<html>
<head>
<title>Home Page</title>
</head>
<body>
<h1>Hello {{username}}</h1>
<a href="/logoutview/">Logout</a>
</body>
</html>
```

Step 4

Go to views.py and add the following function:

```
def logoutview(request):
    logout(request)#logsout the current user
    return redirect("/login")
```

Step 5

Add the following url mapping:

```
path('check/',views.check),
path('home/',views.home),
path('logoutview/',views.logoutview)
```

Step 6

Start the server with the command:

python manage.py runserver

Run the file with the url:

localhost:8000/login

DJANGO AND AJAX

AJAX essentially is a combination of technologies that are integrated together to reduce the number of page loads. Instead of reloading the full page, only part of the page or the data in the page is reloaded.

There are many scenarios where you may want to use AJAX requests in your web application. It is a great resource that enables web applications to be faster and more dynamic.

Views.py

```
from django.http import JsonResponse
def create_student(request):
   if request.method == 'POST':
      form=StudentForm11(request.POST)

      fname = request.POST.get('fname')
      lname = request.POST.get('lname')
      age = request.POST.get('age')
      email = request.POST.get('email')
      response_data = {}

      student = Student(fname=fname,lname=lname,age=age,email=email)
      student.save()

      response_data['result'] = 'Create student successful!'
```

```
        response_data['fname'] = student.fname
        response_data['lname'] = student.lname
        response_data['age'] = student.age
        response_data['email'] = student.email


        return JsonResponse(response_data)
    else:
        form=StudentForm11()
        return render(request,"student.html",{'form':form})
```

student.html

```
<!DOCTYPE html>
<html>
<head>
<title>STUDENT FORM</title>
{% load static %}
<script src="{% static 'js/jquery-3.4.1.js' %}"></script>
</head>
<body>
<form method="post" id="student-form">
{% csrf_token %}
{{form.as_p}}
<ul id="student">
</ul>
 <input type="submit" value="ADD STUDENT">
</form>
```

```
<script>
$('#student-form').on('submit', function(event){
   event.preventDefault();
   console.log("form submitted!")
   createstudent();
});
// AJAX for posting
function createstudent() {


   console.log("create student is working!")
   $.ajax({
      url : "/createstudent/", // the endpoint
      type : "POST", // http method
      data : { 'fname': $('#fname').val(),
                                          'lname': $('#lname').val(),
                                          'age': $('#age').val(),
                                          'email': $('#email').val(),
                                          'csrfmiddlewaretoken':$(
"input[name='csrfmiddlewaretoken']" ).val()


                                 },// data sent with the post request


      // handle a successful response
      success : function(json) {
         $('#fname').val('');
                                       $('#lname').val('');
```

```
                                        $('#age').val('');
                                        $('#email').val(''); // remove the value from the
input



                            $("#student").append("<li><strong>"+json.fname+"</stro
ng> - <em> "+json.lname+"</em> - <span> "+json.age+"</span>");
                                        console.log("success");
},
   });
};
</script>
</body>
</html>
```

Urls.py

```
path('createstudent/',views.create_student)
```