|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| DJANGO  Python web framework |
| |  |  |  | | --- | --- | --- | | Abhijith M | 1/1/20 | Python | |

Table of Contents

[Django 3](#_Toc34658347)

[Install django 3](#_Toc34658348)

[Create new project 3](#_Toc34658349)

[website design Basics 4](#_Toc34658350)

[HTML 4](#_Toc34658351)

[CSS 4](#_Toc34658352)

[JS 4](#_Toc34658353)

[Templates 4](#_Toc34658354)

[Create a template 4](#_Toc34658355)

[Step 1: Create the Templates Folder. In this step we'll create the templates directory. 5](#_Toc34658356)

[Step 2: Create new html file. 5](#_Toc34658357)

[Step 3: Render your template to Django app. 5](#_Toc34658358)

[The final step is to start server 6](#_Toc34658359)

[Passing Context to Templates 6](#_Toc34658360)

[Create a base template 8](#_Toc34658361)

[Generic view 10](#_Toc34658362)

[Adding static files 11](#_Toc34658363)

[URL 12](#_Toc34658364)

[Render template shortcut 13](#_Toc34658365)

[Raising 403 Error 13](#_Toc34658366)

[Adding songs to db 14](#_Toc34658367)

[Design detail template 16](#_Toc34658368)

[Removing hardcoded URL 16](#_Toc34658369)

[Namespace and HTTP 404 shortcut 16](#_Toc34658370)

[Simple Form 17](#_Toc34658371)

[Bootstrap And static frames 19](#_Toc34658372)

[Model Form 21](#_Toc34658373)

[Rest API 25](#_Toc34658374)

[Authentication 27](#_Toc34658375)

[Database setup 31](#_Toc34658376)

[Models 31](#_Toc34658377)

[Fileds 31](#_Toc34658378)

[Code 32](#_Toc34658379)

[Django Database API 33](#_Toc34658380)

[Admin 35](#_Toc34658381)

[Writing another view 36](#_Toc34658382)

[Connecting to db 36](#_Toc34658383)

[Sort items in Db 36](#_Toc34658384)

[Custom user registration 36](#_Toc34658385)

[Form 40](#_Toc34658386)

[HTML Form 40](#_Toc34658387)

[Django Form 40](#_Toc34658388)

[Validation in email 41](#_Toc34658389)

[Login page 42](#_Toc34658390)

[Check authenticated 44](#_Toc34658391)

[Register Form 44](#_Toc34658392)

[Session 47](#_Toc34658393)

[Session key 48](#_Toc34658394)

[Stream video to webpage 49](#_Toc34658395)

[Video streaming using Javascript 50](#_Toc34658396)

[Upload Video 52](#_Toc34658397)

[Enabling https 55](#_Toc34658398)

[Method1 55](#_Toc34658399)

[method 2 56](#_Toc34658400)

[Scheduler 56](#_Toc34658401)

[celery 56](#_Toc34658402)

[Run faceanakytic -stunnel 56](#_Toc34658403)

[Reference 56](#_Toc34658404)

# Django

What is Django used for?

Django is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secures websites. A web application framework is a toolkit of all components need for application development.

Is Django backend or frontend?

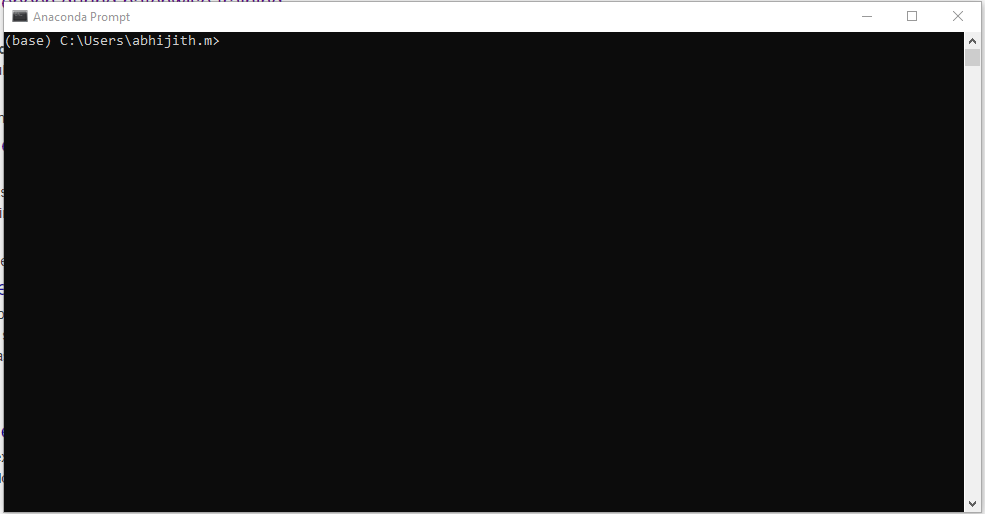
Django is a framework, not a language. Python is the language in which Django is written. Django is a collection of Python libs allowing you to quickly and efficiently create a quality Web application, and is suitable for both frontend and backend.

## Install django

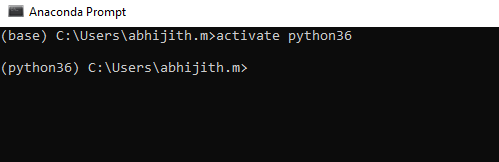
* Open anaconda prompt
* Create a new environment in anaconda prompt
* conda create -n python36 python=3.6
* This command will create a new environment in anaconda. Name of environment is python36
* then activate your newly created environment using following command.
* activate python36
* Now the environment is set. You can do all of Django projects in this environment.
* Simply an environment is a folder containing all necessary packages of your project
* You should need to install Django package in this environment.
* pip install django

## Create new project

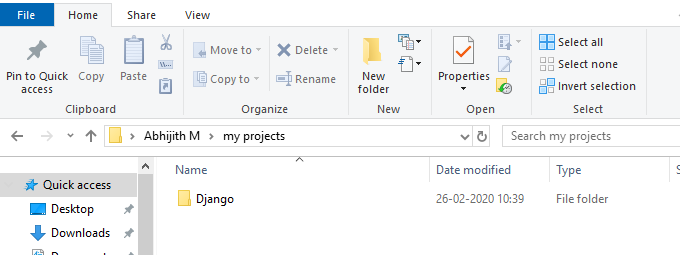
* First step is to activate environment in anaconda prompt
* Open anaconda prompt



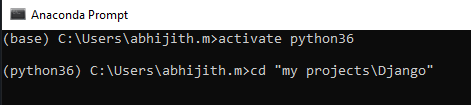
* activate python36



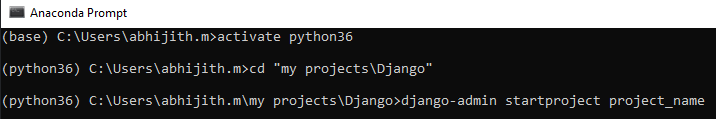
* Create a folder for Django projects in user directory

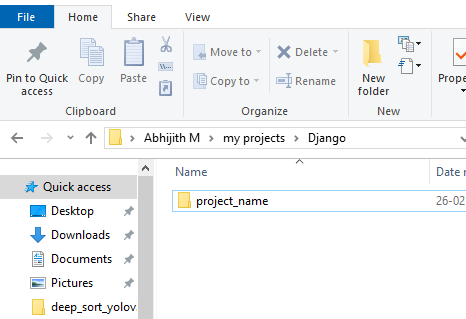


* Move to project directory
* To change directory in command prompt use the following code
* cd “path to your project directory”

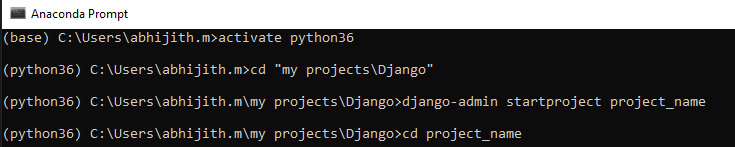


* *(python36) C:\Users\abhijith.m\Django>django-admin startproject project\_name*
* “django-admin startproject project\_name “command will create a new project with name project\_name. A new folder named “project\_name” will create to the current directory.

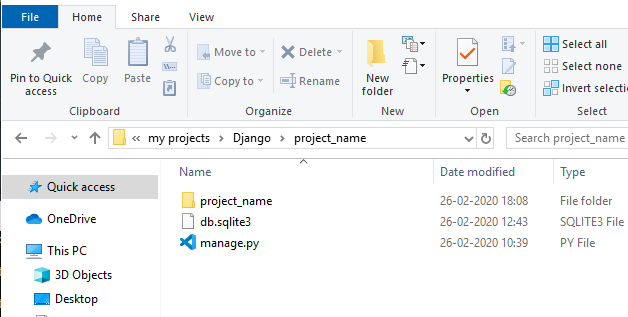




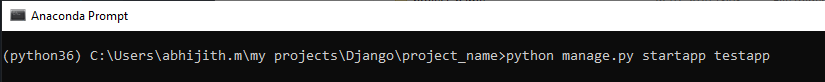
* *(python36) C:\Users\abhijith.m\Django>cd project\_name*
* Cd project\_name will change your current directory in prompt to project\_name

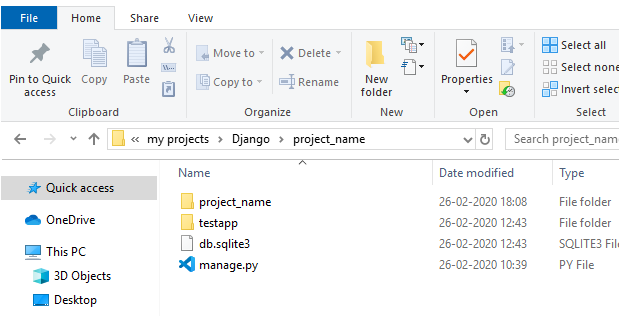


* When we first initialize the project these files are automatically created in project folder.

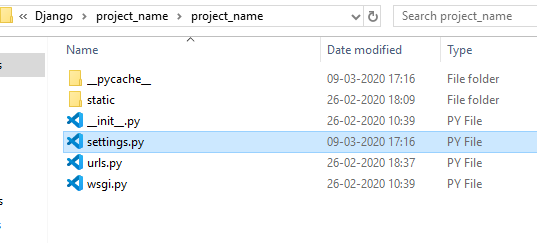


* *(python36) C:\Users\abhijith.m\Django\withoutrest>python manage.py startapp testapp*
* Manage.py is a Django controller file. “python manage.py startapp testapp” this command will create a new app with name testapp





* Open project folder - ../project\_name/project\_name/
* Open settings.py file



* Add your app name to INSTALLED\_APPS

# Application definition

INSTALLED\_APPS = [

    'count\_people.apps.CountPeopleConfig',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'testapp', # <-- your app name

]

* Now the project creation is almost completed. But the view is not created. For that we should need some website design knowledge.

# website design Basics

## HTML

## CSS

## JS

# Templates

What is Jinja template in Django?

**Template** Engines. **Template** engines take in tokenized strings and produce rendered strings with values in place of the tokens as output. **Templates** are typically used as an intermediate format written by developers to programmatically produce one or more desired output formats, commonly HTML, XML or PDF.

What is template engine in Django?

**Django's template engine** provides a powerful mini-language for defining the user-facing layer of your application, encouraging a clean separation of application and presentation logic. **Templates** can be maintained by anyone with an understanding of HTML; no knowledge of Python is required.

What is Jinja template in Django?

**Jinja** is a web **template** engine for the Python programming language and is licensed under a BSD License created by Armin Ronacher. It is similar to the **Django template** engine but provides Python-like expressions while ensuring that the **templates** are evaluated in a sandbox.

Does Django need HTML?

But when you ask about **Django** it **is** a web development framework, So you should learn **html**, css and javascript **is** very essential. ... But when you ask about **Django** it **is** a web development framework, So you should learn **html**, css and javascript **is** very essential.

What is a template used for?

A **template** is a form, mold, or pattern **used** as a guide to making something. ... A document in which the standard opening and closing parts are already filled in is a **template** that you can copy and then fill in the variable parts.

How do I add a template to Django project?

## Create a template

Step 1: Create the Templates Folder. In this step we'll create the templates directory.

Step 2: Create new html file.

Step 3: Render your template to Django app.

* After creating a new project do the following steps to create new template.

### Step 1: Create the Templates Folder. In this step we'll create the templates directory.

* 1. Open the project folder (../project\_name)
  2. open “testapp” folder
  3. In this folder you need to create new folder named “templates” (location:- ../project\_name/testapp/)

### Step 2: Create new html file.

2.1 Create a new “index.html” file in (../project\_name/testapp/templates)

- index.html file

<html>

    <head>

    </head>

    <body>

        <h1>Django template</h1>

    </body>

</html>

### Step 3: Render your template to Django app.

* To show this index page in website do the following steps,
  1. Create new view in views.py (location:- ../project\_name/testapp/views.py)

from django.shortcuts import render

# Create your views here.

def MyView(request):

    return render(request,template\_name="index.html", context={})

* 1. Open urls.py file (location:- ../project\_name/project\_name/urls.py)
* Import views to this file

from testapp import views

* Add url path to map you view
* urls.py file >

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('mytemplate/', views.MyView),

]

### The final step is to start server

* open anaconda prompt
* move to project folder
* (python36) C:\Users\abhijith.m > cd my projects\\Django\\project\_name
* This is my project location you can replace this location with your project path
* Then start running your server
* (python36) C:\Users\abhijith.m\my projects\Django\project\_name>python manage.py runserver

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

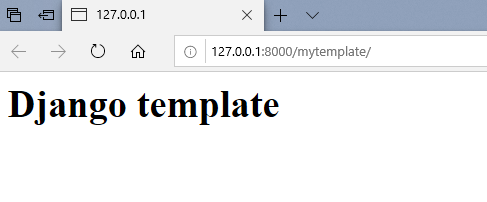
February 26, 2020 - 12:43:57

Django version 2.2.3, using settings 'project\_name.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.

* Open your web browser
* Search this url (<http://127.0.0.1:8000/mytemplate/>) in browser



## Passing Context to Templates

What is context in Django?

When you use a Django Template, it is compiled once (and only once) and stored for future use, as an optimization. A template can have variable names in double curly braces, such as {{ myvar1 }} and {{ myvar2 }}.

A Context is a dictionary with variable names as the key and their values as the value. Hence, if your context for the above template looks like: {myvar1: 101, myvar2: 102}, when you pass this context to the template render method, {{ myvar1 }} would be replaced with 101 and {{ myvar2 }} with 102 in your template. This is a simplistic example, but really a Context object is the context in which the template is being rendered.

What are views in Django?

Django views are a key component of applications built with the framework. At their simplest they are a Python function or class that takes a web request and return a web response. Views are used to do things like fetch objects from the database, modify those objects if needed, render forms, return HTML, and much more.

What is render in Python?

Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text. render() is the same as a call to render\_to\_response() with a context\_instance argument that forces the use of a RequestContext.

Why we use render in Django?

render() Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text. Django does not provide a shortcut function which returns a TemplateResponse because the constructor of TemplateResponse offers the same level of convenience as render() .

How to pass context to template?

* views.py >

from django.shortcuts import render

# Create your views here.

def MyView(request):

    myvar1 = "id1"

    myvar2 = "id2"

    context = {myvar1: 101, myvar2: 102}

    return render(request,template\_name="index.html", context=context)

* Here we passing two variables *myvar1* and *myvar2*. Value of *myvar1* is “id1” so in template page we use *id1* for accessing the value of *id1.* The {{ varname }} is the syntax used to read the value of Django variable.
* In index.html page update the code >

<html>

    <head>

    </head>

    <body>

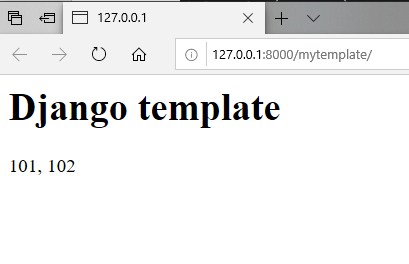
        <h1>Django template</h1>

        {{id1}}, {{id2}}

    </body>

</html>

* Open your browser and refresh the webpage (<http://127.0.0.1:8000/mytemplate/>)



## Create a base template

What is base template in Django?

A base template is the most basic template that you extend on every page of your website. You used the template tag {% block %} to make an area that will have HTML inserted in it. That HTML will come from another template that extends this template ( base. html ).

What is Template inheritance in Django?

Template inheritance. The most powerful – and thus the most complex – part of Django's template engine is template inheritance. Template inheritance allows you to build a base “skeleton” template that contains all the common elements of your site and defines blocks that child templates can override.

How can I create base template in Django?

1. Create a new base.html page in template folder

project\_name

└───testapp

└───templates

base.html

index.html

1. In base.html page add the following code.

<html>

    <head>

        <title> {% block title %} {% endblock %}</title>

    </head>

    <body>

        {% block index\_page %} {% endblock %}

    </body>

</html>

1. In index.html update the code >

{% extends 'base.html' %}

{% block title %}

Index

{% endblock %}

{% block index\_page %}

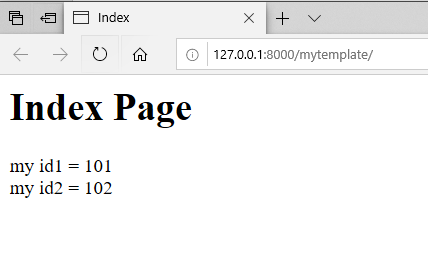
    <h1>Index Page</h1>

    my id1 = {{id1}} <br>

    my id2 = {{id2}}

{% endblock %}

* Refresh browser



* (music/templates/music) > create a new file ‘base.html’ inside music

<!DOCTYPE html>

<html>

<head>

<title>{% block title %} {% endblock %}</title>

</head>

<body>

<nav class="navbar navbar-inverse">…

</nav>

{% block body %}

{% endblock %}

</body>

</html>

* (music/templates/music/index.html) >

{% extends 'music/base.html' %}

{% block title %} Album title {% endblock %}

{% block body %}

{% if all\_albums %}…

{% endif %}

{% endblock %}

* (music/templates/music/details.html) > also like index.html

## Generic view

What is generic Django?

Django offers an easy way to set those simple views that is called generic views. Unlike classic views, generic views are classes not functions. Django offers a set of classes for generic views in Django. views. generic, and every generic view is one of those classes or a class that inherits from one of them.

What are class-based views?

Class-based views. A view is a callable which takes a request and returns a response. This can be more than just a function, and Django provides an example of some classes which can be used as views. These allow you to structure your views and reuse code by harnessing inheritance and mixins.

What is get and post method in Django?

Django's login form is returned using the POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response. GET , by contrast, bundles the submitted data into a string, and uses this to compose a URL.

What is as\_view in Django?

as\_view is the function(class method) which will connect my MyView class with its url. From django docs: classmethod as\_view(\*\*initkwargs) Returns a callable view that takes a request and returns a response: You just can't use class-based views like you could in normal function-based views.

How can I create a generic view in Django?

* Open views.py, delete *MyView* function and create new class-based view
* from django.views.generic import View – is a default function in Django for Generic view.

from django.shortcuts import render

from django.views.generic import View

# Create your views here.

class MyGenericView(View):

    def get(self, request, \*args, \*\*kwargs):

        myvar1 = "id1"

        myvar2 = "id2"

        context = {myvar1: 101, myvar2: 102}

        return render(request, template\_name="index.html", context=context)

* Now the function name is changed so we need to map this function in urls.py page
* In urls.py update path

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('mytemplate/', views.MyGenericView.as\_view()),

]

* Refresh browser. No changes will be found. But the code works fine.

## Adding static files

* Create a ‘static’ folder in parent directory

project\_name

└───static

└───testapp

└───templates

base.html

index.html

* Open settings.py (location:- ../project\_name/project\_name/settings.py)
* Add this code to the bottom of setting.py file

STATIC\_URL = '/static/'

# Add these new lines

STATICFILES\_DIRS = (

os.path.join(BASE\_DIR, 'static'),

)

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

* In static folder create folder css and js
* Inside this folder you can add your javascript and css files

project\_name

└───static

└───css

style.css

└───js

myscript.js

* {% load static %} will load static file link to html page

{% load static %}

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

* Complete code of base.html page

<html>

    <head>

        <title> {% block title %} {% endblock %}</title>

        {% load static %}

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

    </head>

    <body>

        {% block index\_page %} {% endblock %}

    </body>

</html>

* style.css (location:- ../project\_name/static/css/style.css)

body {

    background-color: antiquewhite;

}

* refresh your browser

## URL

What is url naming in Django?

Django offers a way to name urls so it's easy to reference them in view methods and templates. The most basic technique to name Django urls is to add the name attribute to url definitions in urls.py .

How do I name a url in Django?

* Add the name attribute to url definitions in urls.py .

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('mytemplate/', views.MyGenericView.as\_view(), name="mytemplate"

),

]

Using {% url ??? %} in django templates

* If you add a name attribute to the path, you can refer this name in html page.

<form action="{% url ‘logout\_view’ %}">

# Database

What is Database explain?

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Computer databases typically contain aggregations of data records or files, containing information about sales transactions or interactions with specific customers.

What is database example?

A Microsoft Excel spreadsheet or Microsoft Access are good examples of desktop database programs. These programs allow users to enter data, store it, protect it, and retrieve it when needed. Relational databases are the most common database systems. They include databases like SQL Server, Oracle Database, Sybase, Informix, and MySQL.

What is SQL and why it is used?

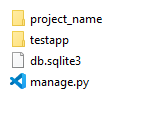
SQL is used to communicate with a database. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

What is called in SQL?

SQL is an abbreviation for structured query language, and pronounced either see-kwell or as separate letters. SQL is a standardized query language for requesting information from a database. The original version called SEQUEL (structured English query language) was designed by an IBM research center in 1974 and 1975.

What is DB(DataBase) sqlite3 in Django?

By default, when we made our first app and started the server you must have seen a new file in your project directory, named as ‘db.sqlite3’. The file is database file where all the data that you will be generating will be stored. It is a local file as Django is a server-side framework and it treats your computer as the host when you actually run the server in command line/terminal.



Does Django use SQL?

Yes and no. To actually use Django, you don't need to use SQL. In fact, Django makes it really really hard for you to use SQL directly with it. It comes with an ORM (Object Relational Mapper), which does most of the behind the scenes work for you.

What is ORM how it's important Django?

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

How connect Django to xampp?

1. Create a database in phpmyadmin.
2. In your settings.py edit the DATABASES DATABASES = { 'default': { 'ENGINE': 'django.db.backends.mysql', 'NAME': 'yourDbName', 'USER': 'root', 'PASSWORD': '', 'HOST': 'localhost', 'PORT': '3306', #my port is 3306 } }
3. Finally, in your terminal.

## How to create superuser in Django?

* Create admin

(python36) C:\Users\abhijith.m\Desktop\website>python manage.py createsuperuser

Username (leave blank to use 'abhijith.m'): admin

Email address: admin@example.com

Password:

Password (again):

Superuser created successfully.

* Open browser
* Go to: <http://127.0.0.1:8000/admin/>
* Enter username and password
* Open music/admin.py

from django.contrib import admin

from .models import Album

admin.site.register(Album)

* Again Go to: <http://127.0.0.1:8000/admin/>

## How does Django connect to MySQL database?

Step 1 — Create the Initial Django Project Skeleton.

Step 2 — Initialize MySQL Database.

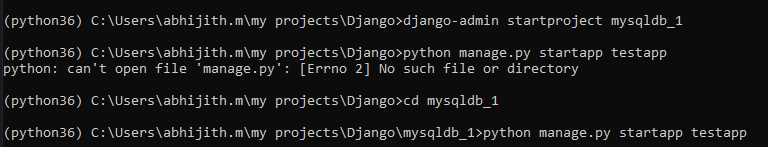
Step 3 — Create the Database.

Step 4 — Create superuser & Add database to admin view.

Step 5 — Test MySQL Connection to Application.

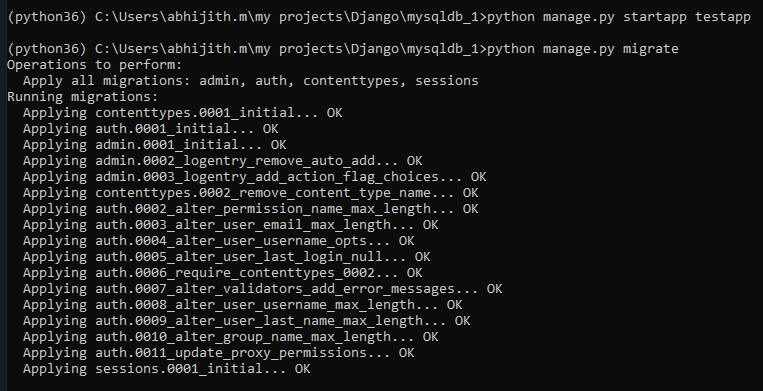
### Step 1 — Create the Initial Django Project Skeleton.

* Create a new project – project name = mysqldb\_1



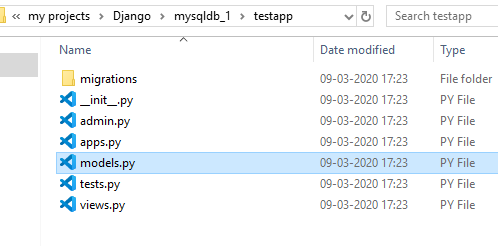
### Step 2 — Initialize MySQL Database.

* Python manage.py migrate
* This command will create all migrations for Django DB initialization.



### Step 3 — Create the Database.

* Open models.py in testapp folder



* Add the following code to create new table in database.

from django.db import models

# Create your models here.

class UserDB(models.Model):

    id = models.AutoField(primary\_key=True)

    name = models.CharField(max\_length=500)

    email = models.EmailField(max\_length=254)

### Step 4 — Create superuser & Add database to admin view.

* Create a superuser and register your model in admin view.

## Adding songs to db

* (music/admin.py)

from django.contrib import admin

from .models import Album, Song

admin.site.register(Album)

admin.site.register(Song)

* (music/models.py) >

class Song(models.Model):

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

file\_type = models.CharField(max\_length=10)

song\_title = models.CharField(max\_length=250)

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

return self.song\_title

* Open terminal

(python36) C:\Users\abhijith.m\Desktop\website>python manage.py shell

Python 3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 7.6.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from music.models import Album, Song

In [3]: album1 = Album.objects.get(pk=1)

In [4]: album1.artist

Out[4]: 'Taylor swift'

In [5]: song = Song()

In [6]: song.album = album1

In [7]: song.file\_type = 'mp3'

In [8]: song.song\_title = 'I hate my boyfriend'

In [9]: song.save()

* Open browser
* <http://127.0.0.1:8000/admin/music/song/>

In [10]: album1.song\_set.all()

Out[10]: <QuerySet [<Song: I hate my boyfriend>]>

In [11]: album1.song\_set.create(song\_title='I love bacon', file\_type='mp3')

Out[11]: <Song: I love bacon>

In [12]: album1.song\_set.create(song\_title='Bucky is lucky', file\_type='mp3')

Out[12]: <Song: Bucky is lucky>

In [13]: album1.song\_set.create(song\_title='Ice cream', file\_type='mp3')

Out[13]: <Song: Ice cream>

In [14]: song = album1.song\_set.create(song\_title='Ice cream', file\_type='mp3')

In [15]: song.song\_title

Out[15]: 'Ice cream'

In [16]: album1.song\_set.count()

Out[16]: 5

## Models

* Open music folder
* Open models.py

### Fileds

DateTimeField

from datetime import date

date\_time = models.DateTimeField(auto\_now\_add=True, blank=True)

or

date = models.DateTimeField(default=datetime.now, blank=True)

AutoField

id = models.AutoField(primary\_key=True)

ForeignKey

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

FileField

video = models.FileField()

BooleanField

train\_status = models.BooleanField(default=True)

Text field

field\_name = models.TextField()

Email field

email = models.EmailField(max\_length=254)

Recursive relation

next\_item\_id = models.ForeignKey('self', on\_delete=models.CASCADE, blank=True)

## Meta

### Combination of two field unique

class ItemsDB(models.Model):

    item\_id = models.CharField( max\_length=200)

    shop = models.ForeignKey(ShopsDB, on\_delete=models.CASCADE)

    item\_name = models.CharField(max\_length=200)

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

    image = models.CharField(max\_length=500)

    price = models.IntegerField()

    measurement\_unit = models.CharField(max\_length=50)

    description = models.TextField()

    validity = models.DateField()

    class Meta:

        unique\_together = ('item\_id','shop')

    def \_\_str\_\_(self):

        return self.item\_name

## Code

from django.db import models

# Create your models here.

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.CharField(max\_length=1000)

class Song(models.Model):

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

file\_type = models.CharField(max\_length=10)

song\_title = models.CharField(max\_length=250)

* Open website/settings.py
* Include following code inside INSTALLED\_APPS

'music.apps.MusicConfig',

* Code

INSTALLED\_APPS = [

'music.apps.MusicConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

* Open terminal
* Goto project directory

1. *python manage.py makemigrations music*
2. *python manage.py sqlmigrate music 0001*
3. *python manage.py migrate*
4. *python manage.py runserver (Restart server)*

## Django Database API

* Open cmd
* Go to project directory

1. python manage.py shell

(python36) C:\Users\abhijith.m\Desktop\website>python manage.py shell

Python 3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 7.6.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from music.models import Album, Song

In [2]: Album.objects.all()

Out[2]: <QuerySet []>

In [3]: a = Album(artist="Taylor swift", album\_title="Red", genre="Country", album\_logo="https://previews.123rf.com/ima

...: ges/queezzard/queezzard1601/queezzard160100009/50339428-equalizer-icon-can-be-used-as-logo-to-music-album-dj-se

...: t-concert-banner-vector-illustration-.jpg")

In [4]: a.save()

In [5]: a.artist

Out[5]: 'Taylor swift'

In [6]: a.album\_title

Out[6]: 'Red'

In [7]: a.id

Out[7]: 1

In [8]: a.pk

Out[8]: 1

In [9]: b = Album()

In [10]: b.artist = "Myth"

In [11]: b.album\_title = "High School"

In [12]: b.genre = "Punk"

In [13]: b.album\_logo = "https://www.kidsdiscover.com/wp-content/uploads/2014/06/Kid\_Smiling-e1401721146279.

...: jpg"

In [14]: b.save()

In [15]: b.artist

Out[15]: 'Myth'

1. When check all objects in album. It will display like this

In [16]: Album.objects.all()

Out[16]: <QuerySet [<Album: Album object (1)>, <Album: Album object (2)>]>

* Open music/models.py
* Include this function inside Artist function

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

return self.album\_title + ' - ' +self.artist

* Full code

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.CharField(max\_length=1000)

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

return self.album\_title + ' - ' +self.artist

1. *exit() # exit from shell*
2. *python manage.py shell*

In [1]: from music.models import Album, Song

In [2]: Album.objects.all()

Out[2]: <QuerySet [<Album: Red - Taylor swift>, <Album: High School - Myth>]>

Filter

In [3]: Album.objects.filter(id=1)

Out[3]: <QuerySet [<Album: Red - Taylor swift>]>

In [5]: Album.objects.filter(artist\_\_startswith='Taylor')

Out[5]: <QuerySet [<Album: Red - Taylor swift>]>

## Writing another view

* Open music/urls.py

### Connecting to db

* In music/views.py

*from .models import Album*

from .models import Album

def index(request):

#connecting to db

all\_albums = Album.objects.all()

html = ''

for album in all\_albums:

url = '/music/' + str(album.id) + '/'

html += '<a href="'+ url +'">' + album.album\_title + '</a><br>'

return HttpResponse(html)

## Sort items in Db

User.objects.all().order\_by('date\_joined') # For ascending

User.objects.all().order\_by('-date\_joined') # For descending; Not '-' sign in order\_by method

## Custom user registration

* (Models.py)>

from django.db import models

from django.contrib.auth.models import User

class UserProfile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

location = models.CharField(max\_length=30)

age = models.IntegerField()

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

return self.user.username

* (admin.py)>

from django.contrib import admin

from .models import UserProfile

admin.site.register(UserProfile)

* (Setings.py)> INSTALLED\_APPS

'testapp',]

* Create file (testapp/forms.py)>

from django import forms

from django.forms import ModelForm

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

from .models import UserProfile

class ExtendedUserCreationForm(UserCreationForm):

email = forms.EmailField(required=True)

first\_name = forms.CharField(max\_length=30)

last\_name = forms.CharField(max\_length=50)

class Meta:

model = User

fields = ('username', 'email', 'first\_name', 'last\_name', 'password1', 'password2')

def save(self, commit=True):

user = super().save(commit=True)

user.email = self.cleaned\_data['email']

user.first\_name = self.cleaned\_data['first\_name']

user.last\_name = self.cleaned\_data['last\_name']

if commit:

user.save()

return user

class UserProfileForm(ModelForm):

class Meta:

model = UserProfile

fields = ("location", "age")

* (views.py)>

from django.shortcuts import render, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib.auth import authenticate, login

from .forms import ExtendedUserCreationForm, UserProfileForm

def index(request):

if request.user.is\_authenticated:

username = request.user.username

else:

username = 'not logged in'

context = {'username' : username}

return render(request, 'example/index.html', context)

def register(request):

if request.method == 'POST':

form = ExtendedUserCreationForm(request.POST or None)

profile\_form = UserProfileForm(request.POST)

if form.is\_valid() and profile\_form.is\_valid():

user = form.save()

profile = profile\_form.save(commit=False)

profile.user = user

profile.save()

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

password = form.cleaned\_data.get('password1')

user = authenticate(username=username, password=password)

login(request, user)

return redirect('index')

else:

form = ExtendedUserCreationForm()

profile\_form = UserProfileForm()

context = {'form': form,

'profile\_form':profile\_form

}

return render(request, "auth/register.html", context)

* (testapp/templates/example/index.html)>

{{username}}

{% if user.is\_authenticated %}

Email: {{ user.email }}

Location: {{ user.userprofile.location }}

{% endif %}

* (testapp/templates/auth/register.html)>

<form method="POST" action="{% url 'register' %}">

{% csrf\_token %}

{{ form.as\_p }}

{{profile\_form.as\_p}}

<button type="submit" >Create new account</button>

</form>

* (urls.py)>

from testapp import views

urlpatterns = [

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

path('register/',views.register, name='register'),

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

]

* <http://127.0.0.1:8000/register/>
* <http://127.0.0.1:8000/admin/> (add user to superuser and staff permission)
* Logout
* <http://127.0.0.1:8000/admin/> login with new user id
* <http://127.0.0.1:8000/index/>

## Raising 403 Error

* (music/templates/music/) > Create a deatil.html file
* Code

{{album}}

* (music/views.py)>
* Code

from .models import Album

from django.shortcuts import render

from django.http import Http404

def detail(request, album\_id):

try:

album = Album.objects.get(pk=album\_id)

except Album.DoesnotExist:

raise Http404("Album does not exist")

return render(request, 'music/detail.html', {'album':album})

## Namespace and HTTP 404 shortcut

404

* (music/views.py)> only need to import the following library. Delete other libraries

from .models import Album

from django.shortcuts import render, get\_object\_or\_404

* Update

def detail(request, album\_id):

# album = Album.objects.get(pk=album\_id)

album = get\_object\_or\_404(Album, pk=album\_id)

return render(request, 'music/detail.html', {'album':album})

## Simple Login Form

* (music/models.py) Add the following line to song model

is\_favorite = models.BooleanField(default=False)

* Full code

class Song(models.Model):

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

file\_type = models.CharField(max\_length=10)

song\_title = models.CharField(max\_length=250)

is\_favorite = models.BooleanField(default=False)

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

return self.song\_title

* Open cmd
* *python manage.py makemigrations music*
* *python manage.py migrate*
* *python manage.py runserver (Restart server)*
* (music/urls.py)> include the path inside ‘urlpatterns’

path('<int:question\_id>/favorite/', views.favorite, name='favorite'),

* (music/templates/music/details.html) >

<img src="{{ album.album\_logo }}">

<h1>{{ album.album\_title }}</h1>

<h3> {{ album.artist }} </h3>

{% if error\_message %}

<p><strong>{{ error\_message }}</strong></p>

{% endif %}

<form action="{% url 'favorite' album.id %}" method="POST">

{% csrf\_token %}

{% for song in album.song\_set.all %}

<input type="radio" id="song{{ forloop.counter }}" name="song" value="{{ song.id }}">

<label for="song{{ forloop.counter }}">

{{ song.song\_title }}

{% if song.is\_favorite %}

<img href="http://icons.iconarchive.com/icons/icons-land/vista-elements/256/Favorites-icon.png"

width="15" />

{% endif %}

</label><br>

{% endfor %}

<input type="submit" value="Favorite" >

</form>

* (music/views.py)> import ‘Song’

from .models import Album, Song

* Create a new method
* (music/views.py) >

def favorite(request, album\_id):

album = get\_object\_or\_404(Album, pk=album\_id)

try:

selected\_song = album.song\_set.get(pk=request.POST['song'])

except (KeyError, Song.DoesNotExist):

return render(request, 'music/detail.html', {

'album':album,

'error\_message': "You did not select a valid song",

})

else:

selected\_song.is\_favorite = True

selected\_song.save()

return render(request, 'music/detail.html', {'album':album})

## Model Form

* (music/models.py)>

from django.core.urlresolvers import reverse

* Add inside Album

def get\_absolute\_url(self):

return reverse('music:detail', kwargs={'pk': self.pk})

* Album

# Create your models here.

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.CharField(max\_length=1000)

def get\_absolute\_url(self):

return reverse('music:detail', kwargs={'pk': self.pk})

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

return self.album\_title + ' - ' +self.artist

views.py

from django.http import Http404

from .models import Album, Song

from django.shortcuts import render

def index(request):

#connecting to db

all\_albums = Album.objects.all()

context = {

'all\_albums': all\_albums,

}

return render(request, 'music/index.html', context)

def favorite(request, pk):

try:

album = Album.objects.get(pk=pk)

except Album.DoesnotExist:

raise Http404("Album does not exist")

try:

selected\_song = album.song\_set.get(pk=request.POST['song'])

except (KeyError, Song.DoesNotExist):

return render(request, 'music/detail.html', {

'album':album,

'error\_message': "You did not select a valid song",

})

else:

selected\_song.is\_favorite = True

selected\_song.save()

return render(request, 'music/detail.html', {'album':album})

def detail(request, pk):

try:

album = Album.objects.get(pk=pk)

except Album.DoesnotExist:

raise Http404("Album does not exist")

return render(request, 'music/detail.html', {'album':album})

## Authentication

* Start new project
  1. *django-admin startproject authenticate*
  2. *cd authenticate*
  3. *python manage.py startapp user\_example*
  4. *python manage.py migrate*
  5. *python manage.py createsuperuser*

*Username (leave blank to use 'abhijith.m'): admin*

*Email address: admin@example.com*

*Password:*

*Password (again):*

*Superuser created successfully.*

* (authenticate/urls.py)> inside ‘urlpatterns’

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

path('', include('user\_example.urls')),

path('accounts/', include('django.contrib.auth.urls'))

* (authenticate/settings.py)> inside ‘INSTALLED\_APPS’

'user\_example',

* (user\_example/)> Create another file ‘urls.py’

from django.urls import path

from . import views

urlpatterns = [

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

]

* (user\_example/views.py)>

from django.shortcuts import render

def index(request):

return render(request, 'user\_example/index.html')

* (user\_example/templates/user\_example)> create another file ‘index.html’

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h1>This is my index page</h1>

</body>

</html>

* (user\_example/templates) create folder registration
* (user\_example/templates/registration) create file ‘login.html’

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

{% if form.errors %}

<p>There's something wrong with what you entered!</p>

{% endif %}

{% if next %}

<p>Hey, you can't access that page.</p>

{% endif %}

<form action="{% url 'login' %}" method="post">

{% csrf\_token %}

<p> Username: {{ form.username }} </p>

<p> Password: {{ form.password }} </p>

<input type="submit" value="login">

<input type="hidden" name= value="{{ next }}">

</form>

</body>

</html>

* (authenticate/settings.py)> add ‘LOGIN\_REDIRECT\_URL’ end of the file

STATIC\_URL = '/static/'

LOGIN\_REDIRECT\_URL = '/'

* In browser <http://127.0.0.1:8000/accounts/login/> login with (username:admin password:\*\*\*\*\*)
* (user\_example/views.py)> add

from django.contrib.auth.forms import UserCreationForm

def register(request):

form = UserCreationForm()

context = {'form' : form}

return render(request, 'registration/register.html', context)

* (userexample/templates/registration/)> create file ‘’registration.html’’

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Register</title>

</head>

<body>

<form method="POST" action="{% url 'register' %}"></form>

{% csrf\_token %}

{% if form.errors %}

<p>There are errors in the form!</p>

{% endif %}

{{ form }}

<input type="submit" value="Register">

</body>

</html>

* (user\_example/urls.py)> add path

path('register', views.register, name = 'register'),

* In browser <http://127.0.0.1:8000/register>
* (user\_example/views.py)> complete code

from django.shortcuts import render, redirect

def index(request):

return render(request, 'user\_example/index.html')

from django.contrib.auth import authenticate, login

from django.contrib.auth.forms import UserCreationForm

def register(request):

if request.method == 'POST':

form = UserCreationForm(request.POST)

if form.is\_valid():

form.save()

username = form.cleaned\_data['username']

password = form.cleaned\_data['password1']

user = authenticate(username= username, password=password)

login(request, user)

return redirect('index')

context = {'form' : form}

return render(request, 'registration/register.html', context)

* (user\_example/templates/user\_example/index.html)> update

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h1>This is my index page</h1>

{% if user.is\_authenticated %}

<h2>Your name is: {{ user.username }}</h2>

{% else %}

<h2>You are not logged in.</h2>

{% endif %}

</body>

</html>

# Form

## HTML Form

* (testapp/views.py)>

def login\_page(request):

return render(request, "auth/login.html", {})

* (testapp/templates/auth/login.html)>

<form method="POST" >

{% csrf\_token %}

<input type="text" placeholder="Name" name="fullname">

<button type="submit" >Submit</button>

</form>

* (urls.py)>

path('testapp/',views.login\_page),

* Update (views.py)>

def login\_page(request):

if request.method == "POST":

print(request.POST)

print(request.POST.get('fullname'))

return render(request, "auth/login.html", {})

* In browser <http://127.0.0.1:8000/testapp/>

## Django Form

* Create file (testapp/forms.py)

from django import forms

class ContactForm(forms.Form):

fullname = forms.Charfield()

email = forms.EmailField()

* In (views.py)>

from .forms import ContactForm

def login\_page(request):

contact\_form = ContactForm()

if request.method == "POST":

print(request.POST)

print(request.POST.get('fullname'))

context = {

'form':contact\_form,

}

return render(request, "auth/login.html", context)

* In (templates/auht/login.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* In browser <http://127.0.0.1:8000/testapp/>

## Validation in email

* (Testapp/forms.py)>

from django import forms

class ContactForm(forms.Form):

fullname = forms.CharField(

widget=forms.TextInput(

attrs={

"id":"form\_full\_name",

"placeholder":"Your full name"

}

)

)

#fullname = forms.CharField(widget=forms.TextInput(attrs={"class":"form-control","id":"form\_full\_name", "placeholder":"Your full name"}))

email = forms.EmailField()

def clean\_email(self):

email = self.cleaned\_data.get("email")

if not "gmail.com" in email:

raise forms.ValidationError("Email has to be gmail.com")

return email

* (testapp/views.py)>

def login\_page(request):

contact\_form = ContactForm(request.POST or None)

if contact\_form.is\_valid():

print(contact\_form.cleaned\_data)

# if request.method == "POST":

# print(request.POST)

# print(request.POST.get('fullname'))

context = {

'form':contact\_form,

}

return render(request, "auth/login.html", context)

* (testapp/templates/auth/login.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* In browser <http://127.0.0.1:8000/testapp/>
* Check email validation with and without ‘gmail.com’

## Login page

* (testapp/forms.py)>

class LoginForm(forms.Form):

username = forms.CharField()

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

* (testapp/views.py)>

from .forms import ContactForm, LoginForm

def login\_page(request):

login\_form = LoginForm(request.POST or None)

print(request.user.is\_authenticated)

if login\_form.is\_valid():

print(login\_form.cleaned\_data)

context = {

'form':login\_form,

}

return render(request, "auth/login.html", context)

* Create (templates/auth/login.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* In browser <http://127.0.0.1:8000/login/>
* In (views.py)>

from django.contrib.auth import authenticate, login

from django.shortcuts import render, redirect

def login\_page(request):

login\_form = LoginForm(request.POST or None)

context = {

'form':login\_form,

}

print("\*" \* 10)

print(request.user.is\_authenticated)

if login\_form.is\_valid():

print(login\_form.cleaned\_data)

username = login\_form.cleaned\_data.get("username")

password = login\_form.cleaned\_data.get("password")

user = authenticate(request, username=username, password=password)

print(user)

if user is not None:

login(request, user)

#context['form'] = LoginForm()

return redirect("/login")

else:

print("Error")

return render(request, "auth/login.html", context)

* In cmd
  + python manage.py createsuperuser
* in browser <http://127.0.0.1:8000/login/>

## Check authenticated

* in (urls.py)>

path('home/',views.home\_page),

* (views.py>

def home\_page(request):

# if not request.user.is\_authenticated():

# return Login

print(request.session.get("first\_name", "unknown"))

print(request.session.get("user", "unknown"))

context = {}

if request.user.is\_authenticated:

context["premium\_content"] = "User logged in"

return render(request, "testapp/home.html",context)

* (testapp/templates/testapp/home.html)>

<h1>Cart</h1>

{% if request.user.is\_authenticated%}

<h1>Premium</h1>

{{ premium\_content }}

{% endif %}

* In browser <http://127.0.0.1:8000/home/>

## Register Form

* (forms.py)>

class RegisterForm(forms.Form):

username = forms.CharField()

email = forms.EmailField()

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

password2 = forms.CharField(label='Confirm password' , widget=forms.PasswordInput())

* In (views.py)>

from .forms import ContactForm, LoginForm, RegisterForm

def register\_page(request):

register\_form = RegisterForm(request.POST or None)

context = {

'form':register\_form,

}

if register\_form.is\_valid():

print(register\_form.cleaned\_data)

return render(request, "auth/register.html", context)

* Create file (testapp/templates/auth/register.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* (urls.py)>

path('register/',views.register\_page),

* (forms.py)>

class RegisterForm(forms.Form):

username = forms.CharField()

email = forms.EmailField()

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

password2 = forms.CharField(label='Confirm password' , widget=forms.PasswordInput())

def clean(self):

data = self.cleaned\_data

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

password2 = self.cleaned\_data.get('password2')

if password2 != password:

raise forms.ValidationError("password must match")

return data

* (views.py)>

from django.contrib.auth import authenticate, login, get\_user\_model

User = get\_user\_model()

def register\_page(request):

register\_form = RegisterForm(request.POST or None)

context = {

'form':register\_form,

}

if register\_form.is\_valid():

print(register\_form.cleaned\_data)

username = register\_form.cleaned\_data.get("username")

email = register\_form.cleaned\_data.get("email")

password = register\_form.cleaned\_data.get("password")

new\_user = User.objects.create\_user(username, email, password)

print(new\_user)

return render(request, "auth/register.html", context)

* (forms.py)>

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

class RegisterForm(forms.Form):

username = forms.CharField()

email = forms.EmailField()

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

password2 = forms.CharField(label='Confirm password' , widget=forms.PasswordInput())

def clean\_username(self):

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

qs = User.objects.filter(username = username)

if qs.exists():

raise forms.ValidationError("Username is taken")

return username

def clean(self):

data = self.cleaned\_data

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

password2 = self.cleaned\_data.get('password2')

if password2 != password:

raise forms.ValidationError("password must match")

return data

# Session

1. django-admin startproject session
2. cd session
3. python manage.py startapp testapp

* (testapp) views.py >

from django.shortcuts import render

# Create your views here.

def cart\_home(request):

print(request.session)

print(dir(request.session))

return render(request, "testapp/home.html",{})

* Urls.py

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

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

path('testapp/',views.cart\_home),

]

* Settings.py

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'testapp',

]

* Create folder:- testapp/templates/home.html

<h1>Cart</h1>

## Session key

* Testapp/views.py

from django.shortcuts import render

# Create your views here.

def cart\_home(request):

# print(request.session)

# print(dir(request.session))

key = request.session.session\_key

print(key)

return render(request, "testapp/home.html",{})

* In cmd
  + python manage.py migrate
  + python manage.py runserver
* in (testapp/views.py)>

def cart\_home(request):

# print(request.session)

# print(dir(request.session))

# key = request.session.session\_key

# print(key)

request.session['first\_name'] = "Justin"

return render(request, "testapp/home.html",{})

def home\_page(request):

print(request.session.get("first\_name", "unknown"))

return render(request, "testapp/home.html",{})

* (urls.py)>

urlpatterns = [

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

path('testapp/',views.cart\_home),

path('testapp1/',views.home\_page),

]

* Refresh browser >
* <http://127.0.0.1:8000/testapp/>
* <http://127.0.0.1:8000/testapp1/>

# Stream video to webpage

* Views.py

from django.shortcuts import render

from django.views.decorators import gzip

from django.shortcuts import render

from django.http import HttpResponse,StreamingHttpResponse

import cv2

import time

class VideoCamera(object):

def \_\_init\_\_(self):

self.video = cv2.VideoCapture(0)

def \_\_del\_\_(self):

self.video.release()

def get\_frame(self):

ret,image = self.video.read()

ret,jpeg = cv2.imencode('.jpg',image)

return jpeg.tobytes()

def gen(camera):

while True:

frame = camera.get\_frame()

yield(b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@gzip.gzip\_page

def index(request):

try:

return StreamingHttpResponse(gen(VideoCamera()),content\_type="multipart/x-mixed-replace;boundary=frame")

except:

print("aborted")

* urls.py

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

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

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

]

# Video streaming using Javascript

* index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta content="stuff, to, help, search, engines, not" name="keywords">

<meta content="What this page is about." name="description">

<meta content="Display Webcam Stream" name="title">

<title>Display Webcam Stream</title>

<style>

#container {

margin: 0px auto;

width: 500px;

height: 375px;

border: 10px #333 solid;

}

#videoElement {

width: 500px;

height: 375px;

background-color: #666;

}

</style>

</head>

<body>

<div id="container">

<video autoplay="true" id="videoElement">

</video>

</div>

<script>

var video = document.querySelector("#videoElement");

if (navigator.mediaDevices.getUserMedia) {

navigator.mediaDevices.getUserMedia({

video: true

})

.then(function(stream) {

video.srcObject = stream;

// myJson = JSON.stringify(stream)

})

.catch(function(err0r) {

console.log("Something went wrong!");

});

}

console.log(video)

</script>

</body>

</html>

## Stream video Xframe option error

from django.http import HttpResponse

from django.views.decorators.clickjacking import xframe\_options\_exempt

@xframe\_options\_exempt

def ok\_to\_load\_in\_a\_frame(request):

    return HttpResponse("This page is safe to load in a frame on any site.")

* views.py

def index(request):

return render(request, 'testapp/index.html', {})

# Upload Video

* create ‘media’ folder in parent directory
* settings.py > bottom of page

MEDIA\_ROOT= os.path.join(BASE\_DIR, 'media/')

MEDIA\_URL= "/media/"

* model.py

from django.db import models

# Create your models here.

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.FileField()

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

return self.album\_title + " - " + self.artist

* admin.py

from .models import Album

admin.site.register(Album)

* setting.py

INSTALLED\_APPS = [

'upload.apps.UploadConfig',

* setting.py – bottom of page

STATIC\_URL = '/static/'

MEDIA\_ROOT= os.path.join(BASE\_DIR, 'media/')

MEDIA\_URL= "/media/"

* urls.py

from django.contrib import admin

from django.urls import path

from django.conf import settings

from django.conf.urls.static import static

from upload import views

urlpatterns = [

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

path('index/', views.IndexCBV.as\_view(), name='index'),

]

if settings.DEBUG:

urlpatterns += static(settings.STATIC\_URL, document\_root = settings.STATIC\_ROOT)

urlpatterns += static(settings.MEDIA\_URL, document\_root = settings.MEDIA\_ROOT)

* forms.py >

from django import forms

from .models import Album

class AlbumForm(forms.ModelForm):

class Meta:

model = Album

fields = ['artist', 'album\_title', 'genre', 'album\_logo']

* view.py

from django.shortcuts import render

from django.views.generic import View

from .forms import AlbumForm

from .models import Album as AlbumDB

# Create your views here.

class IndexCBV(View):

def get(self, request, \*args, \*\*kwargs):

uploadForm = AlbumForm()

all\_albums = AlbumDB.objects.all()

context = {

'form' : uploadForm,

'all\_albums' : all\_albums,

}

return render(request, "index.html", context)

def post(self, request, \*args, \*\*kwargs):

print(request.POST, request.FILES)

artist = request.POST['artist']

album\_title = request.POST['album\_title']

genre = request.POST['genre']

album\_logo = request.FILES['album\_logo']

formdata = {

'artist' : artist,

'album\_title' : album\_title,

'genre' : genre,

'album\_logo' : album\_logo,

}

form = AlbumForm(request.POST, request.FILES)

if form.is\_valid():

form.save(commit=True)

result = 'Resource created successfully'

if form.errors:

result = form.errors

context = {

'result' : result

}

return render(request, "index.html", context )

* (templates/index.html)

{% for album in all\_albums %}

<a href="#" >

<img src="{{ album.album\_logo.url }}" >

</a>

{{ album.album\_title }}

{% endfor %}

<form method="POST" enctype="multipart/form-data">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

{% if result %}

Result: {{result}}

{% endif %}

* Forms.py

from django import forms

from .models import Album

class AlbumForm(forms.ModelForm):

class Meta:

model = Album

fields = ['artist', 'album\_title', 'genre', 'album\_logo']

# Enabling https

## Method1

sudo apt-get update

sudo apt-get install software-properties-common

sudo add-apt-repository universe

sudo add-apt-repository ppa:certbot/certbot

sudo apt-get update

sudo apt-get install python-certbot-apache

sudo certbot --apache

sudo certbot renew --dry-run

## method 2

<https://gist.github.com/claudiosanches/7012524>

* stunnel4 dev\_https & python3 manage.py runserver 0.0.0.0:8444& HTTPS=1 python3 manage.py runserver 0.0.0.0:8001

<https://ai.zerone-consulting.com:8444/login/>

# Scheduler

## celery

* sudo apt-get install rabbitmq-server
* pip install celery
* pip install celery[redis]

# Run faceanakytic -stunnel

stunnel4 stunnel/dev\_https & sudo python3 manage.py runserver 0.0.0.0:8443& HTTPS=1 sudo python3 manage.py runserver 0.0.0.0:8001

# Hosting your site in a network or public domain – 0.0.0.0:80

# Reference

1. <https://docs.djangoproject.com/en/2.2/>