Building Blog CMS (Content Management System) with Django

Last Updated : 07 Jun, 2023

Django is a python based web application framework that is helpful for building a variety of web applications. Django also includes an extensible Django-Admin interface, Default SQLIte3 database, which is also extensible to PostgreSQL, MySQL databases, and a few other components to build efficient web apps.

Install and Setup Django

Create a directory for our blog, install and activate the virtual environment. Then install Django using the below commands

```
# creating directory for our project
mkdir gfgblog && cd gfgblog

# installing virtual environment
pip install virtualenv
python3 -m venv env

# activating virtual environment
source env/bin/activate

# installing django
pip install django
```

As now we have installed Django we will now create a Django project which will set up a basic Django app

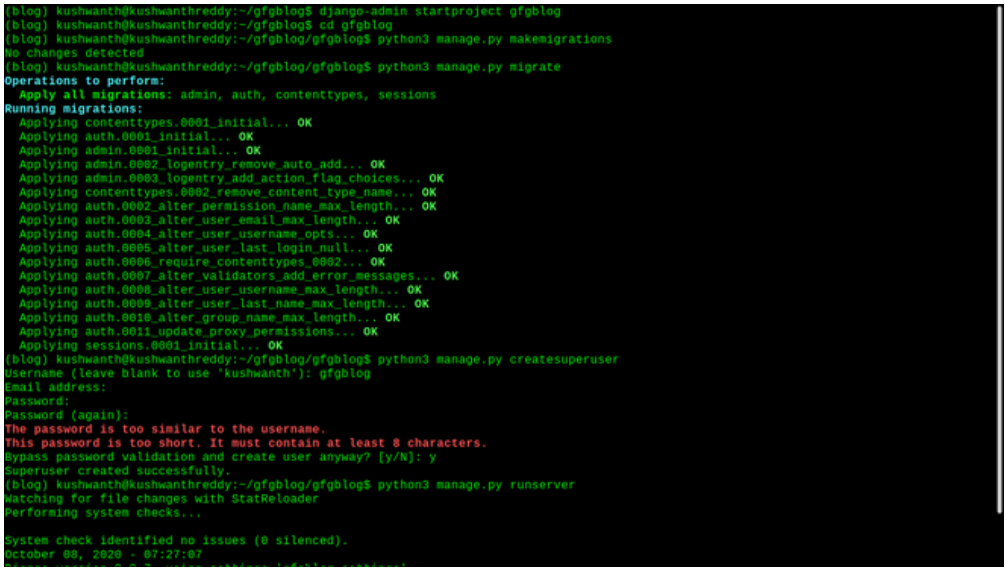
```
django-admin startproject gfgblog
cd gfgblog
```

In our gfgblog Django app, we will have these files

- **__init__.py** – empty file
- **urls.py** – for routing our Django project
- **settings.py** – has all settings for our Django project
- **asgi.py**, wsgi – helpful while deploying our app

We are in the Django web app directory. Now we will make some migrations for our database which will be SQLite3 which will set up some default tables to run our app in our database. Then we will create a superuser for our app.

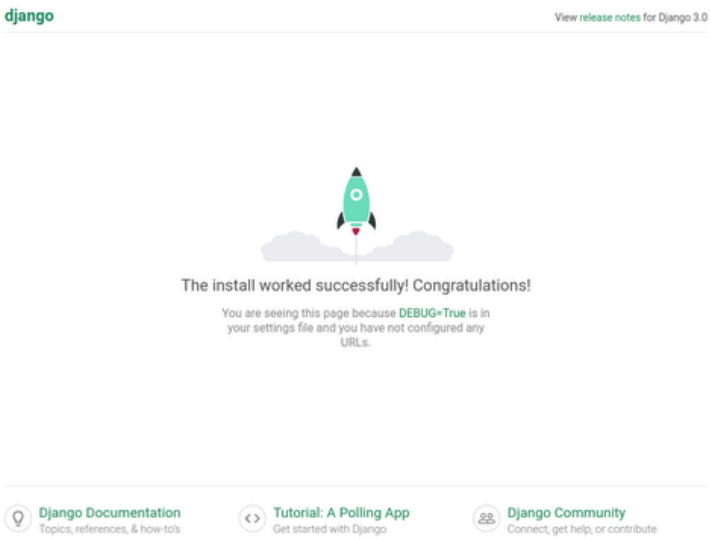
```
# migrating tables
python3 manage.py makemigrations
python3 manage.py migrate

# create and enter the details for superuser
python3 manage.py createsuperuser
```



creating, migrating django-app

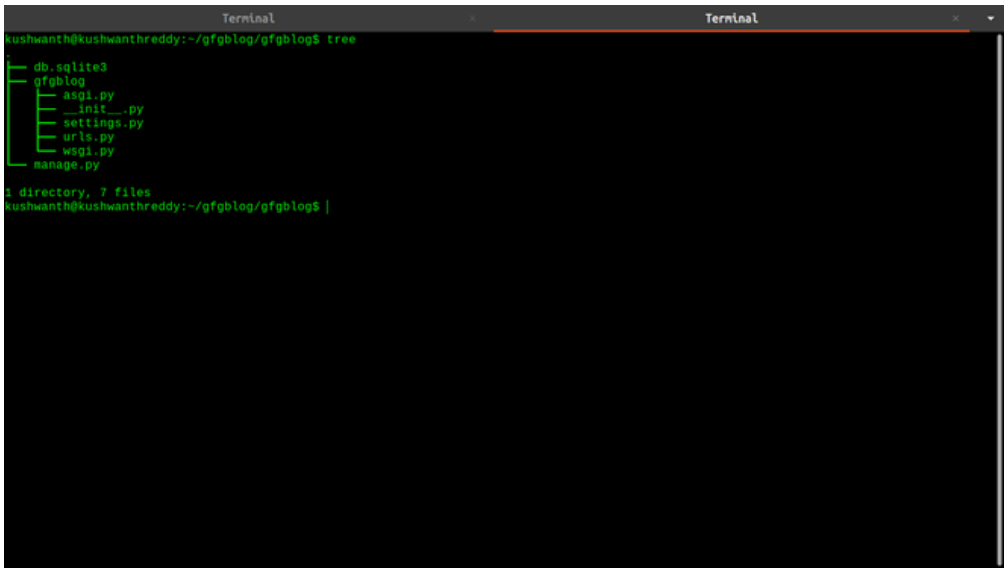
Now run the Django development server and open port 8000 in your localhost

```
# running python development server
python3 manage.py runserver
```



default django page

Now stop the server and go into gfgblog directory in our gfgblog Django app directory and open urls.py file using your code editor. You can look at the tree of our gfgblog directory from the below picture



django app directory tree structure

Create a Blog App in Django –

Now we will create our actual blog app and database for it. Go to the gfgblog project directory. You can see our SQLite3 database, gfgblog Django app. now in this directory create a new app named blog. The below command will create a new app for us.

```
# creating an app named blog
python3 manage.py startapp blog
```

The new app directory has 5 default files

- **__init__.py** – an empty file
- **admin.py** – for managing admin interface
- **apps.py** – for managing app configuration
- **models.py** – for managing database models of the app
- **tests.py** – for testing the app
- **views.py** – for managing behavior and logic of the app

As we have created the app now we have to tell the django app that there is a new app in our project. For that go to settings.py in gfgblog directory and open settings.py file. Go to installed apps section of the settings file and add the name of the app we created; in our case its blog

Python3

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

Creating Models for Blog CMS –

Now we will create Django models for the app. You can learn more about Django Models from this tutorial – Django Models

To create Django Models open in your editor and add the below code

Python3

```
# importing django models and users
from django.db import models
from django.contrib.auth.models import User

STATUS = (
    (0,"Draft"),
    (1,"Publish"),
    (2, "Delete")
)

# creating an django model class
class posts(models.Model):
    # title field using charfield constraint with unique constraint
    title = models.CharField(max_length=200, unique=True)
    # slug field auto populated using title with unique constraint
    slug = models.SlugField(max_length=200, unique=True)
    # author field populated using users database
    author = models.ForeignKey(User, on_delete= models.CASCADE)
    # and date time fields automatically populated using system time
    updated_on = models.DateTimeField(auto_now= True)
    created_on = models.DateTimeField()
    # content field to store our post
    content = models.TextField()
    # meta description for SEO benefits
    metades = models.CharField(max_length=300, default="new post")
    # status of post
    status = models.IntegerField(choices=STATUS, default=0)

    # meta for the class
    class Meta:
        ordering = ['-created_on']
    # used while managing models from terminal
    def __str__(self):
        return self.title
```

Save the file and make migration so that the fields will be created in our database using the below commands

```
# creates migrations for the blog app
python3 manage.py makemigrations blog
```

```
# migrates the blog app
python3 manage.py migrate blog
```

So far we are good with creating models and setting our database for storing our posts, but we should present them to users. Luckily Django comes with its own templating language with which we can build dynamic HTML pages to serve our users. We can also add styling using CSS and JavaScript. You can learn more about Django templates in this tutorial – Django Templates

Creating Templates for Blog CMS –

To create templates first create a template directory in the blog app (You can also create it outside the blog app but in this way, it would be easy to manage). To manage templates in an easy way for each app in the project we have to do some changes in the settings. Go to the project settings file and replace template settings with the code below


```
TEMPLATES_DIR = os.path.join(BASE_DIR,'templates')

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATES_DIR],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Now we will create templates for our blog. As Django can build pages dynamically we will build our templates according to it. Now we will create a

base template that serves as a base for all the pages in our blog.

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- will be replaced with meta content -->
  {% block metatags %}{% endblock %}
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
  <style>
html, body {
  height: 100%;
}
  body {
    font-family: sans-serif;
    font-size: 18px;
    background-color: #fdfdfd;
display: flex;
  flex-direction: column;
  }
  #nofooter {
flex: 1 0 auto;
}
darkbtn {
color: #66ff99; }
 .dark-mode {
    background-color: black;
    color: white;
  }
  #foot {
    flex-shrink: 0;
    font-family: sans-serif;
background-color: #BDFFD3; color: #00308F;
    bottom: 0px;
    width: 100%;
}
.card-group,.card,.card-footer,.card-body {
border: none;
}
</style>
</head>
<body><div id="bodydiv"><div id="nofooter">
  <nav class="navbar navbar-expand-lg navbar-light bg-light shadow" id="topnav">
    <div class="container-fluid">
      <a class="navbar-brand" href="{% url 'home' %}">
```

```
        <img src="/static/icons/logo.png" width="30" height="30" class="d-inline-bl
        GeeksForGeeks</a>
        <div class="nav justify-content-end">
          <div class="d-inline"><a class="nav-link text-black font-weight-bold" hre
          </div></div>
</nav>
   <!-- will be replaced with post content -->
{% block content %}
{% endblock content %}
</div>
<footer class="footer" id="foot">
<div class="container"><h5 style="text-align:center;">Enjoy the Blog and Stay Upd
<nav class="navbar navbar-expand-lg">
<div class="container-fluid">




<p>Copyright © GeeksForGeeks</p>




<div class="nav justify-content-center" id="navfooter">
<span><a class="nav-link text-black font-weight-bold" href="{% url 'about' %}">Ab
<span><a class="nav-link text-black font-weight-bold" href="{% url 'privacy' %}">
<span><a class="nav-link text-black font-weight-bold" href="{% url 'tos' %}">Term
<span><a class="nav-link text-black font-weight-bold" href="#">Feed</a></span>
</div>
</div>
  </nav>
    </div>
</footer></div>
</body>
</html>
```

This will be our base template, You can block like {% block name %}{% endblock %} which will be replaced with the content assigned to them, Now we will create a home template that would be the homepage for our blog which will have the latest posts.

HTML

```
{% extends "base.html" %}
{% block metatags %}
<title>Home | GeeksForGeeks</title>
<meta name="description" content="A destination for Learning">
<meta property="og:title" content="GeeksForGeeks">
<meta property="og:site_name" content="GeeksForGeeks">
<meta property="og:url" content="https://GeeksForGeeks.org">
<meta property="og:description" content="A destination for Learning">
<meta property="og:type" content="website">
{% endblock %}
{% block content %}
<style type="text/css">
    body {
       font-family:  'Raleway', sans-serif;}
    .head_text {
        color: white;
    }
.h1, h4 {
font-family: 'Raleway', sans-serif;
}
#mainhome {
text-align: center; }
</style>
<header class="jumbotron" style="background-color: #BDFFD3; color: #00308F;">
    <div class="container">
      <p class="h1"> Welcome to <strong>GeeksForGeeks</strong></p>




              </div>
</header>
<div class="container">
    <div class="row">
        <div class="col-md-8 mt-3 left">
            {% for post in posts_list %}
            <div class="shadow-none card mb-4" id="newsfeed">
                <div class="card-body">
                    <h3 class="card-title">{{ post.title }}</h3>
                    <p class="card-text text-muted h6"><span>{{ post.author.first_




                    <p class="card-text">{{post.metades }}</p>
```

```
                    <span><a href="{% url 'post_detail' post.slug  %}" class="btn
                </div>
            </div>
            {% endfor %}
        </div>
</div>
</div>
{% if is_paginated %}
  <nav aria-label="Page navigation container"></nav>
  <ul class="pagination justify-content-center">
    {% if page_obj.has_previous %}
    <li><a href="?page={{ page_obj.previous_page_number }}" class="btn btn-outlin
    {% endif %}
    {% if page_obj.has_next %}
    <li><a href="?page={{ page_obj.next_page_number }}" class="btn btn-outline-in

    {% endif %}
  </ul>
{% endif %}
{%endblock%}
```

We have created our homepage with pagination. Posts will be displayed using the for a loop. Now we will add our last template to display posts individually.

HTML

```
{% extends 'base.html' %}
{% block metatags %}
<title>{{ object.title }}</title>
<meta name="description" content="{{ object.metades}}" />
<meta property="og:title" content="{{ object.title }}">
<meta property="og:site_name" content="GeeksForGeeks">
<meta property="og:url" content="{% url 'post_detail' object.slug %}">
<meta property="og:description" content="{{ object.metades }}">
<meta property="og:type" content="article">
{% endblock %}
{% block content %}
<style type="text/css">
#container img {
border-radius: 29px;
width: 100%;
```

```
height: 360px;
opacity: 0.7;
align-content: center;
}
#container img {
opacity: 1.0; }
a {text-align: center; text-decoration: none;}
</style>
<script type="application/ld+json">
{
  "@context": "https://schema.org/",
  "@type": "Post",
  "headline": "{{ object.title }}",
  "description": "{{ object.metades }}",
  "mainEntityOfPage": {
    "@type": "WebPage",
    "@id": "{% url 'post_detail' object.slug %}"
},
  "author": {
    "@type": "Person",
    "name": "{{ post.author.first_name }} {{ post.author.last_name "
  },
  "publisher": {
    "@type": "Organization",
    "name": "GeeksForGeeks",
  },
  "datePublished": "{{ news.created_on }}",
  "dateModified": "{{ news.created_on }}",
  "mentions": "{{ object.source }}"
}
</script>
<div class="container">
  <div class="row">
<div class="col-md-6 left">
        <h1 class="card-title">{% block title %} {{ object.title }} {% endblock t
        <p class=" text-muted">{{ object.author.first_name }} {{ object.author.la




        <p class="card-text">{{ object.content | safe }}</p>



      </div>
       </div>
```

```
        </div>
{% endblock content %}
```

Both homepage and post page will be built by extending the base template and replacing blocks with the data stored in our database. Both also have a Facebook open graphs for better sharing and social network and the post page has a structured schema for better SEO additionally. We will build AMP template incoming tutorial.

Creating Views for Blog CMS –

Now open views.py file in the blog directory. This file has the logic to run the blog app. We will be using class-based views for the blog app. Class Based Generic Views are an advanced set of Built-in views which are used for the implementation of selective view strategies such as Create, Retrieve, Update, Delete.

Python3

```
# importing models and libraries
from django.shortcuts import render
from .models import posts
from django.views import generic
from django.views.decorators.http import require_GET
from django.http import HttpResponse

# class based views for posts
class postslist(generic.ListView):
    queryset = posts.objects.filter(status=1).order_by('-created_on')
    template_name = 'home.html'
    paginate_by = 4

# class based view for each post
class postdetail(generic.DetailView):
    model = posts
    template_name = "post.html"
```

Creating Routes for Blog CMS –

For Routing our blog app just go into the blog app directory and create a file **urls.py** to route our app. see the code below

Python3

```
# importing django routing libraries
from . import views
from django.urls import path, include
from .views import *
from .feeds import blogFeed

urlpatterns = [
    # home page
    path('', views.postslist.as_view(), name='posts'),
    # route for posts
    path('<slug:slug>/', views.postdetail.as_view(), name='post_detail'),
]
```

Go to urls.py file in gfgblog directory and route our blog app. See the code below for reference

Python3

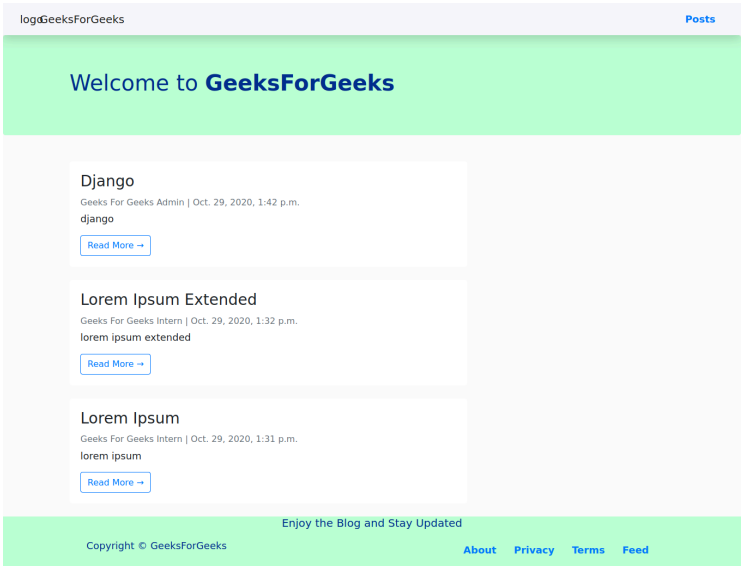
```
from django.contrib import admin
from django.urls import path, include, re_path
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # urls handling admin route
    path('admin/', admin.site.urls),
    # urls handling blog routes
    path('', include('blog.urls')),
]
```

Done, Let's Now run the server using

```
Python manage.py runserver
```

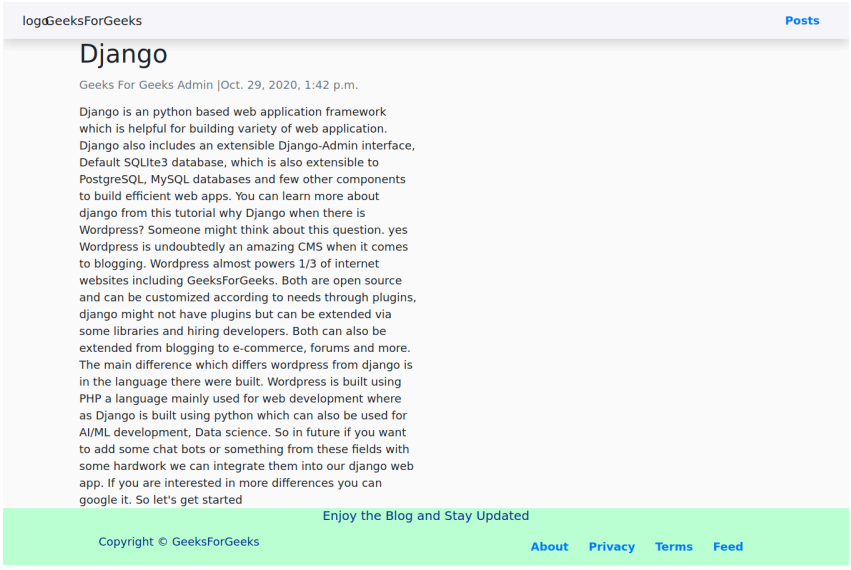
Blog CMS Screenshots –

HomePage –



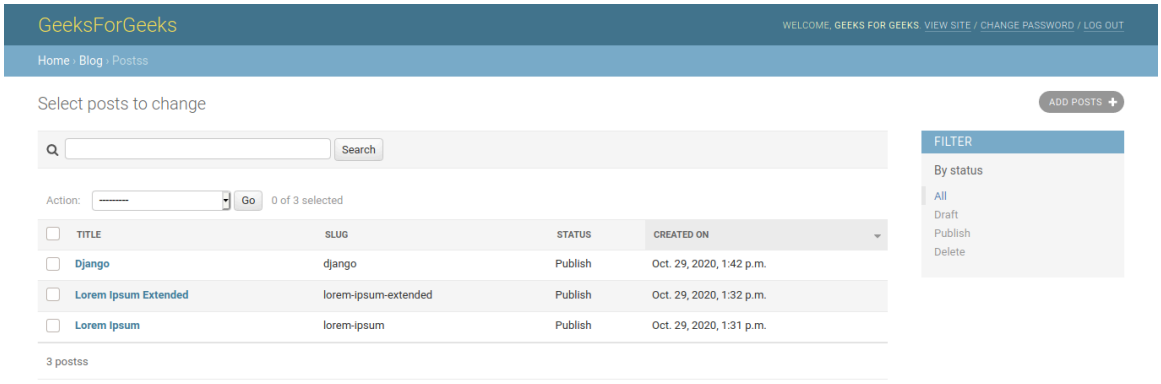
homepage with latest's posts

A sample post page –



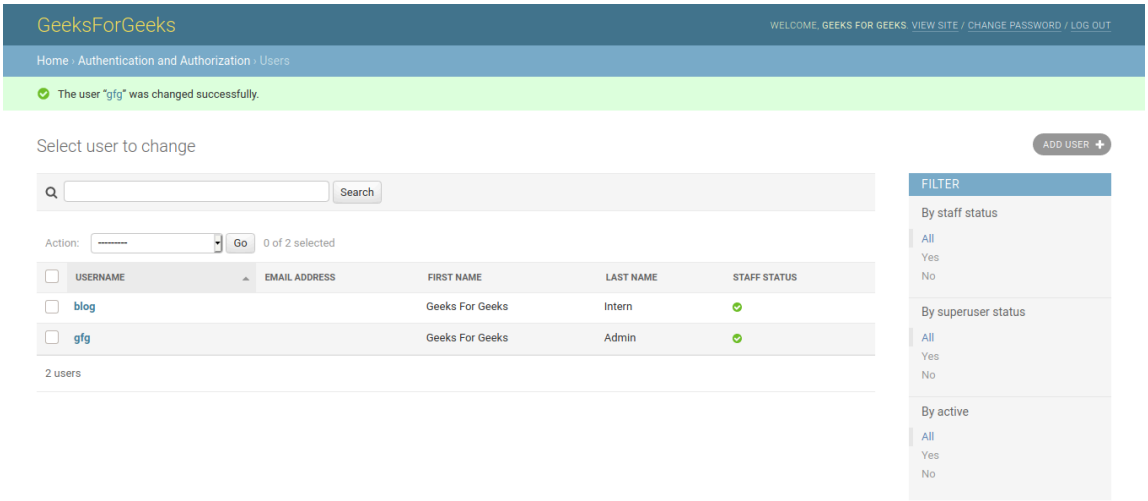
sample post page

Posts management with Django Admin



posts management

User management via Django Admin



User management

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our **Mastering Django Framework - Beginner to Advanced Course** on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

Next Article

Create Task Management System using Django

Similar Reads

Blog Post Recommendation using Django

In this article, we will guide you through the creation of a blog post recommendation system using Django. Our article covers the integration of a...

10 min read

7 Mistakes You Should Avoid While Building a Django Application

Django...We all know the popularity of this Python framework. Django has become the first choice of developers to build their web applications. It is a free...

6 min read

Building Web App with Django and FastAPI

Django is a powerful and popular web framework for building robust web applications with Python. It comes with a lot of built-in features, including an...

4 min read

Building APIs using FastAPI with Django

Combining FastAPI and Django can leverage the strengths of both frameworks: FastAPI's high performance for building APIs and Django's powerful ORM and...

3 min read

Building an Issue Tracker with Django

In this article, we will guide you through creating an Issue Tracker using Django. A Django issue tracker is a web-based application designed to help software...

6 min read

College Management System using Django - Python Project

In this article, we are going to build College Management System using Django and will be using dbsqlite database. In the times of covid, when education has...

15+ min read

Event Management System Using Python Django

In today's fast-paced world, businesses, schools, and groups need to organize events well. An Event Management System (EMS) makes planning and managin...

13 min read

Create Task Management System using Django

Task management systems are essential tools for individuals and organizations to organize, track, and prioritize tasks efficiently. In this article, we'll explore how to...

15+ min read

Build Blog website using Flask

In this article, we'll explore how to build a dynamic blog website using Flask, a lightweight and versatile Python web framework. Flask provides developers wit...

15+ min read

Adding Tags Using Django-Taggit in Django Project

Django-Taggit is a Django application which is used to add tags to blogs, articles etc. It makes very easy for us to make adding the tags functionality to our djang...

2 min read

Article Tags : Python Django-Projects python Python Django

Practice Tags : python python

Corporate & Communications Address:- A-143, 9th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305) | Registered Address:- K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305