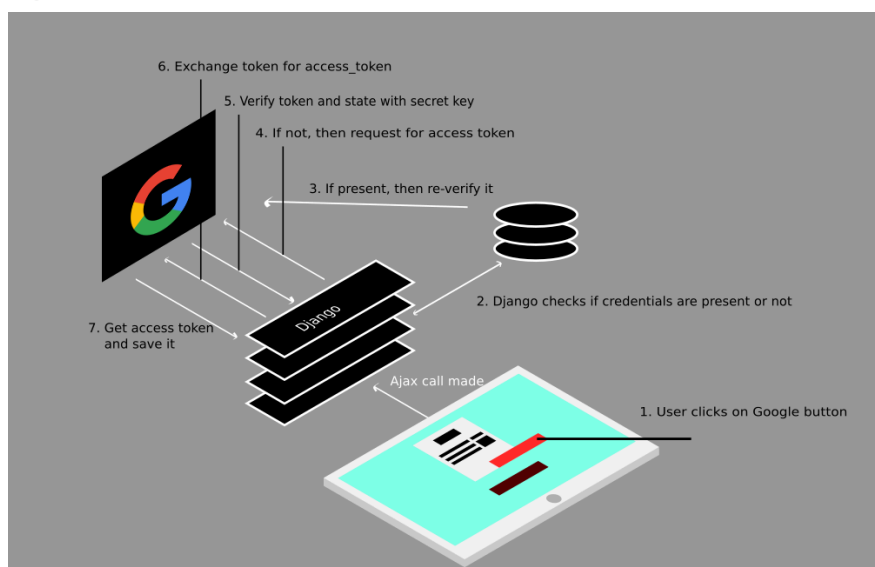




Python Django | Google authentication and Fetching mails from scratch

Last Updated : 18 Apr, 2023

Google Authentication and Fetching mails from scratch mean without using any module which has already set up this authentication process. We'll be using Google API python client and oauth2client which is provided by Google. Sometimes, it really hard to implement this Google authentication with these libraries as there was no proper documentation available. But after completing this read, things will be clear completely. Now Let's create Django 2.0 project and then implement Google authentication services and then extract mails. We are doing extract mail just to show how one can ask for permission after authenticating it.



Step #1: Creating Django Project The first step is to create virtual environment and then install the dependencies. So We'll be using venv:

```
mkdir google-login && cd google-login
```

```
python3.5 -m venv myenv  
source myenv/bin/activate
```

This command will create one folder myvenv through which we just activated virtual environment. Now type

```
pip freeze
```

Then you must see no dependencies installed in it. Now first thing first let's install Django:

```
pip install Django==2.0.7
```

That is Django version which we used but feel free to use any other version. Now next step is to create one project, let's name it gfglogin:

```
django-admin startproject gfglogin .
```

Since we are inside google-login directory that's why we want django project to be on that present directory only so for that you need to use '.' at the end for present directory indication. Then create one app to separate logic from main project, so create one app called gfgauth:

```
django-admin startapp gfgauth
```

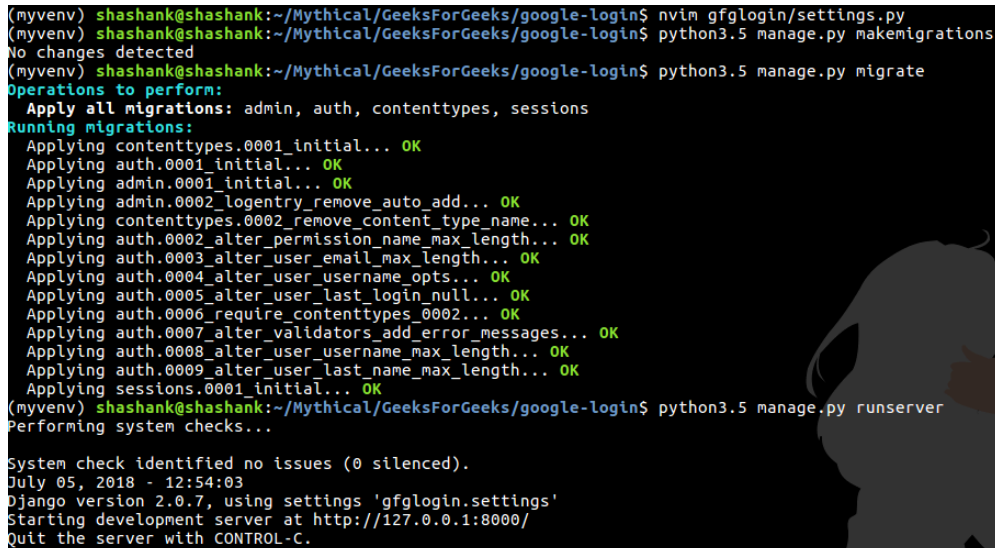
So overall terminal will look like:

```
shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ python3.5 -m venv myvenv
shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ source myvenv/bin/activate
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ pip install Django
Collecting Django
  Using cached https://files.pythonhosted.org/packages/ab/15/cfde97943f0db45e4f999c60b696fbb4df59e82bbccc686770f4e44c9094/Django-2.0.7-py3-none-any.whl
Collecting pytz (from Django)
  Using cached https://files.pythonhosted.org/packages/30/4e/27c34b62430286c6d59177a0842ed90dc789ce5d1ed740887653b898779a/pytz-2018.5-py2.py3-none-any.whl
Installing collected packages: pytz, Django
Successfully installed Django-2.0.7 pytz-2018.5
You are using pip version 8.1.1, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ django-admin startproject gfglogin .
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ django-admin startapp gfgauth
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$
```

Since we created one app. Add that app name into settings.py in INSTALLED_APP list. Now we have Django project up running, so let's migrate it first and then check if there is any error or not.

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

So after migrating, one should be able to run the server and see the starting page of Django on that particular url.



```
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ nvim gfglogin/settings.py
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ python3.5 manage.py makemigrations
No changes detected
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ python3.5 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
(myvenv) shashank@shashank:~/Mythical/GeeksForGeeks/google-login$ python3.5 manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
July 05, 2018 - 12:54:03
Django version 2.0.7, using settings 'gfglogin.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Step #2: Installing dependencies Since we have the project up running successfully, let's install basic requirements. First, we need googleapiclient, this is needed because we have to create one resource object which helps to interact with the API. So to be precise we will make use of 'build' method from it. To install:

```
pip install google-api-python-client==1.6.4
```

Now the second module is oauth2client, this will make sure of all the authentication, credential, flows and many more complex thing so it is important to use this.

```
pip install oauth2client==4.1.2
```

And at last, install jsonpickle, (just in case if it is not installed) because it will be used by oauth2client while making CredentialsField.

```
pip install jsonpickle==0.9.6
```

So these are the only dependencies which we need. Now let's step into the coding part and see how it works. **Step #3:** Creating models Use models to store the credentials which we get from an API, so there are only 2 main field which needed to take care of. First is id , which will be ForeignKey and second

is credential which will be equal to CredentialsField. This field needs to be imported from oauth2client. So our models.py will look like:

Python3

```
from django.contrib import admin
from django.contrib.auth.models import User
from django.db import models
from oauth2client.contrib.django_util.models import CredentialsField

class CredentialsModel(models.Model):
    id = models.ForeignKey(User, primary_key = True, on_delete = models.CASCADE)
    credential = CredentialsField()
    task = models.CharField(max_length = 80, null = True)
    updated_time = models.CharField(max_length = 80, null = True)

class CredentialsAdmin(admin.ModelAdmin):
    pass,
```

Currently task and updated_time are simply extra fields added, hence can be removed. So this credential will hold the credential data in the database.

Important guideline: When we import CredentialsField, then automatically __init__ method gets executed at back and if you'll notice the code in path /google-login/myenv/lib/python3.5/site-packages/oauth2client/contrib/django_util/__init__.py Line 233 They are importing urlresolvers so that they can make use of the reverse method from it. Now the problem is that this urlresolvers has been removed after Django 1.10 or Django 1.11, If you are working on Django 2.0 then it will give an error that urlresolvers cannot be found or not there. Now to overcome this problem we need to change 2 lines, first replace that import from django.core import urlresolvers from django.urls import reverse And then replace Line 411 urlresolvers.reverse(...) to reverse(...) Now you should be able to run it successfully. After creating these models:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

Step #4: Creating Views Right now we have only 3 main views to handle requests. First is to show the home page, status, Google button so that we can send requests for Authentication. Second view will get triggered when the google button is clicked, means an AJAX request. Third will be to handle the return request from google so that we can accept the access_token from it and save it in our database. First, let's do the Google authentication thing: So right now we need to specify the flow to the API that what permissions we need to ask, what is my secret key and redirect url. So to do that enter:

```
FLOW = flow_from_clientsecrets(
    settings.GOOGLE_OAUTH2_CLIENT_SECRETS_JSON,
    scope='https://www.googleapis.com/auth/gmail.readonly',
    redirect_uri='http://127.0.0.1:8000/oauth2callback',
    prompt='consent')
```

As you can notice settings.GOOGLE_OAUTH2_CLIENT_SECRETS_JSON, so go to settings.py file and type:

```
GOOGLE_OAUTH2_CLIENT_SECRETS_JSON = 'client_secrets.json'
```

This tells Django that where json file is present. We will later download this file. After specifying the flow, let's start logic. Whenever we need to see if someone is authorized or not we first check our database whether that user credentials is already present or not. If not then we make requests to API url and then get credentials.

Python3

```
def gmail_authenticate(request):
    storage = DjangoORMStorage(CredentialsModel, 'id', request.user, 'credential')
    credential = storage.get()

    if credential is None or credential.invalid:
        FLOW.params['state'] = xsrfutil.generate_token(settings.SECRET_KEY,
                                                         request.user)

        authorize_url = FLOW.step1_get_authorize_url()
        return HttpResponseRedirect(authorize_url)
    else:
```

```

http = httplib2.Http()
http = credential.authorize(http)
service = build('gmail', 'v1', http=http)
print('access_token = ', credential.access_token)
status = True

return render(request, 'index.html', {'status': status})

```

We use DjangoORMStorage (which is provided by oauth2client) so that we can store and retrieve credentials from Django datastore. So we need to pass 4 parameters for it. First is the model class which has CredentialsField inside it. Second is the unique id that has credentials means key name, third is the key value which has the credentials and last is the name of that CredentialsField which we specified in models.py. Then we get the value from storage and see if it is valid or not. If it is not valid then we create one user token and get one to authorize url, where we redirect the user to the Google login page. After redirecting user will fill up the form and once user is authorized by google then google will send data to *callback url* with access_token which we will do later. Now in case, if user credentials was already present then it will re-verify the credentials and give you back the access_token or sometimes the refreshed access_token in case if the previous one was expired. Now we need to handle the callback url, to do that:

Python3

```

def auth_return(request):
    get_state = bytes(request.GET.get('state'), 'utf8')
    if not xsrfutil.validate_token(settings.SECRET_KEY, get_state,
                                   request.user):
        return HttpResponseRedirect()

    credential = FLOW.step2_exchange(request.GET.get('code'))
    storage = DjangoORMStorage(CredentialsModel, 'id', request.user, 'credential')
    storage.put(credential)

    print("access_token: % s" % credential.access_token)
    return HttpResponseRedirect("/")

```

Now inside callback url when we get one response from google then we

[Python Basics](#) [Interview Questions](#) [Python Quiz](#) [Popular Packages](#) [Python Projects](#) [Practice Python](#) [AI Wit](#)

generated by generateToken. So what we do is that we validate the token with secret_key, the token which we generated and with the user who generated it. These things get verified by xsrfutil.validate_token method which make sure that the token is not too old with the time and it was generated at given particular time only. If these things doesn't work out well then it will give you error else you will go to the next step and share the code from callback response with google so that you can get the access_token. So this was 2 step verification and after successfully getting the credential we save it in Django datastore by using DjangoORMStorage because through that only we can fetch and store credential into CredentialsField. Once we will store it we can redirect the user to any particular page and that's how you can get access_token. Now let's create one home page which will tell whether the user is logged in or not.

Python3

```
def home(request):
    status = True

    if not request.user.is_authenticated:
        return HttpResponseRedirect('admin')

    storage = DjangoORMStorage(CredentialsModel, 'id', request.user, 'credential')
    credential = storage.get()

    try:
        access_token = credential.access_token
        resp, cont = Http().request("https://www.googleapis.com/auth/gmail.readon:
                                   headers ={'Host': 'www.googleapis.com',
                                   'Authorization': access_token})

    except:
        status = False
        print('Not Found')

    return render(request, 'index.html', {'status': status})
```

Right now we are assuming that the user is authenticated in Django, means that user is no more anonymous and has info saved in database. Now to have support for the anonymous user, we can remove the checks for credentials in database or create one temporary user. Coming back to home view, first we will check whether the user is authenticated or not, means that user is not anonymous, if yes then make him do log in else check for credentials first. If user has already logged in from Google then it will show Status as True else it will show False. Now coming to templates let's create one. First go to root folder and create one folder named as 'templates', then inside that create index.html:

HTML

```
{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <script src="{% static 'js/main.js' %}"></script>
    <title>Google Login</title>
</head>
<body>

<div>
    <div>
        {% if not status %}
            <a href="/gmailAuthenticate" onclick="gmailAuthenticate()" title="Goo{
        {% else %}
            <p>Your are verified</p>
        {% endif %}
    </div>

</div>
</body>
</html>
```

Now this page is very much simplified so no css or styling is there, just one simple link to check. Now you will notice *js file* also. so again go to root folder

and create one directory as static/js/ And inside js create one javascript file main.js:

javascript

```
function gmailAuthenticate(){
    $.ajax({
        type: "GET",
        url: "ajax/gmailAuthenticate",
        // data: '',
        success: function (data) {
            console.log('Done')
        }
    });
};
```

This *js file* was used to separate the logic from HTML file and also to make one AJAX call to Django. Now we have all the parts done for views. **Step #5:** Creating urls and basic settings In the main project urls means gfglogin/urls.py edit and put:

Python3

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('gfgauth.urls')),
]
```

Because we need to test gfgauth app working. Now inside gfgauth/urls.py type:

Python3

```
from django.conf.urls import url
from . import views
```

```
urlpatterns = [
    url(r'^gmailAuthenticate', views.gmail_authenticate, name = 'gmail_authenticate'),
    url(r'^oauth2callback', views.auth_return),
    url(r'^$', views.home, name = 'home'),
]
```

As you can see, gmailAuthenticate is for AJAX calls, oauth2callback is for callback url and the last one is home page url. Now before running there are few settings which we haven't talked about: In settings.py you need to edit:

1. In TEMPLATES list add 'templates' inside DIRS list.
2. At last of settings.py file add:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)

GOOGLE_OAUTH2_CLIENT_SECRETS_JSON = 'client_secrets.json'
```

So we just specified where templates and static files are present and most importantly where is google oauth2 client secret json file. Now we will download this file. **Step #6:** Generating Oauth2 Client secret file Head over to [Google Developer Console page](#) and create one project and name it anything you like. After creating it, head over to the project dashboard and click on the Navigation menu which is on the top left. Then click on API services and then credentials page. Click on create credentials (you might need to set product name before going ahead so do that first). Now select web application since we are using Django. After this specifies the name and then simply go to redirect URIs and over there type:

```
http://127.0.0.1:8000/oauth2callback
```

And then save it. You need not specify the Authorized Javascript origins so leave it blank for now. After saving it, you will be able to see all your credentials, just download the credentials, it will get saved in some random

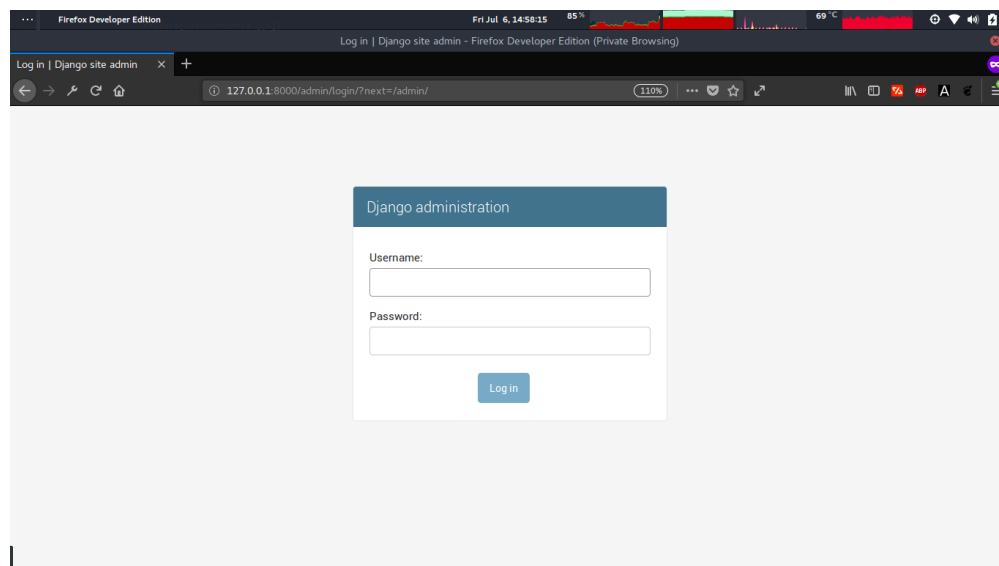
names so simply reformat the filename and type 'client_secrets' and make sure it is of json format. Then save it and paste it on Django project root folder (where manage.py is there). **Step #7:** Running it Now recheck if everything is correct or not. Make sure you have migrated it. Also before going ahead create one superuser so that you are no more anonymous:

```
python3.5 manage.py createsuperuser
```

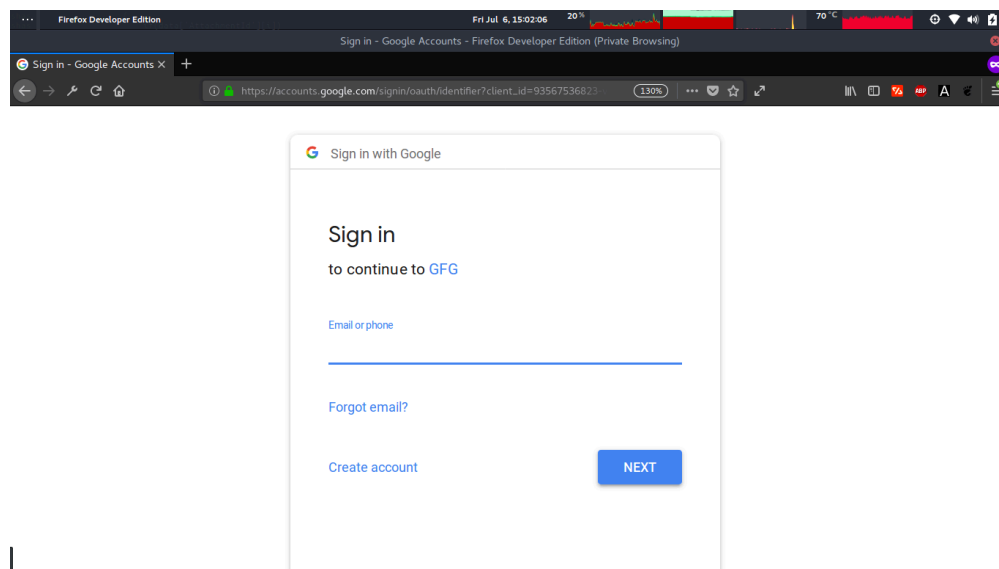
Type all the necessary details and then do:

```
python3.5 manage.py runserver
```

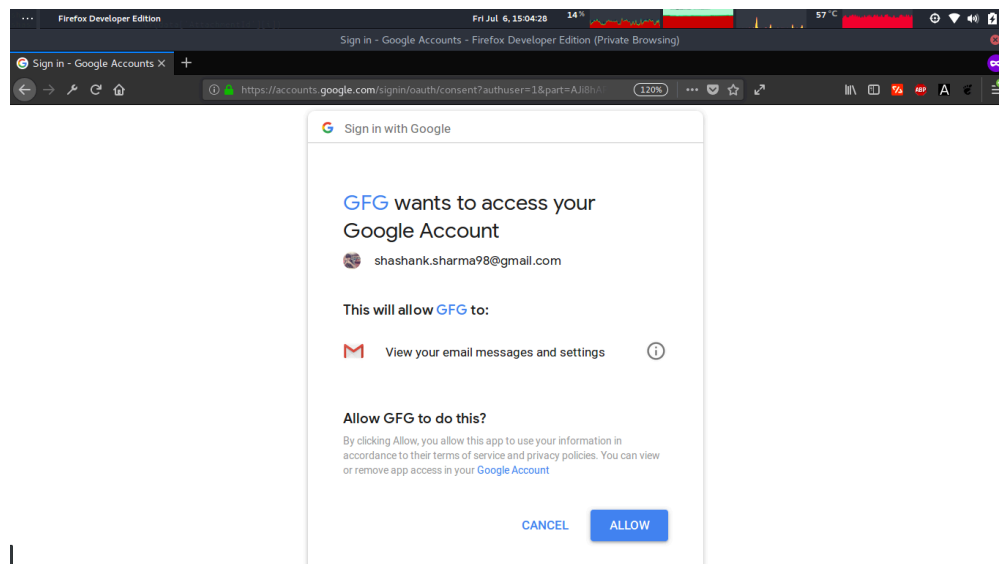
And go to <http://127.0.0.1:8000> You will see this:



Which is perfectly fine, now type your superuser credentials over here and then you will be able to see the admin dashboard. Just avoid that and again go to <http://127.0.0.1:8000> Now you should be able to see the Google link, now click on it and you will see:



Now as you can see it is saying Sign in to continue to GFG, here GFG is my Project name. So it is working fine. Now enter your credentials and after submitting you will see:



Since we are requesting for mails permission that's why it is asking the user to allow it. In case if it is showing error then in Google console you might need to activate Gmail API into your project. Now once you will allow it you will get the credentials and get saved inside your database. In case if the user clicks on Cancel then you will need to write a few more codes to handle such flow. Now in case if you allowed it then you will be able to see the access_token on your console/database. After getting access_token you can make use of this to fetch user email and all other stuff. See full code in this repository [here](https://www.geeksforgeeks.org/python-django-google-authentication-and-fetching-mails-from-scratch/).

Looking to dive into the world of programming or sharpen your Python skills? Our [Master Python: Complete Beginner to Advanced Course](#) is your ultimate guide to becoming proficient in Python. This course covers everything you need to build a solid foundation from fundamental programming concepts to advanced techniques. With **hands-on projects**, real-world examples, and expert guidance, you'll gain the confidence to tackle complex **coding challenges**. Whether you're starting from scratch or aiming to enhance your skills, this course is the perfect fit. Enroll now and master Python, the language of the future!

S shas...



26

Previous Article

Django Sign Up and login with confirmation Email | Python

Next Article

ToDo webapp using Django

Similar Reads

Fetching recently sent mails details sent via a Gmail account using Python

In this article, we will look at how to fetch a fixed number of recently sent e-mails via a Gmail account using Python. The libraries used in this implementation...

2 min read

Handling mails with EZGmail module in Python

EZGmail is a Python module that can be used to send and receive emails through Gmail. It works on top of the official Gmail API. Although EZGmail does not cove...

4 min read

E-Mails Notification Bot With Python

Email continues to be a widely used communication method, making it an effective platform for receiving updates and staying connected. However, manua...

3 min read

Fetching text from Wikipedia's Infobox in Python

An infobox is a template used to collect and present a subset of information about its subject. It can be described as a structured document containing a set ...

3 min read

Fetching and Displaying Data with HTMX and FastAPI

HTMX allows us to utilize current AJAX methods in HTML directly, while FastAPI is a strong backend framework for quick and efficient API development. This...

3 min read

Fetching top news using News API

News API is a simple JSON-based REST API for searching and retrieving news articles from all over the web. Using this, one can fetch the top stories running o...

5 min read

Build an Authentication System Using Django, React and Tailwind

In this article, we will guide you in building an Authentication system using React and Tailwind with the Django Framework that includes features like sign up, log...

15+ min read

Token Authentication in Django Channels and Websockets

Prerequisites: Django, WebSockets, Django channels, Token authentication The most popular Django topics right now are WebSockets and Django channels...

13 min read

Implement Token Authentication using Django REST Framework

Token authentication refers to exchanging username and password for a token that will be used in all subsequent requests so to identify the user on the server...

2 min read

JWT Authentication with Django REST Framework

JSON Web Token is an open standard for securely transferring data within parties using a JSON object. JWT is used for stateless authentication mechanisms for...

2 min read

Article Tags : [Python](#)

Practice Tags : [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



Company

[About Us](#)
[Legal](#)
[In Media](#)
[Contact Us](#)
[Advertise with us](#)
[GFG Corporate Solution](#)
[Placement Training Program](#)
[GeeksforGeeks Community](#)

DSA

[Data Structures](#)
[Algorithms](#)
[DSA for Beginners](#)
[Basic DSA Problems](#)
[DSA Roadmap](#)
[Top 100 DSA Interview Problems](#)
[DSA Roadmap by Sandeep Jain](#)
[All Cheat Sheets](#)

Languages

[Python](#)
[Java](#)
[C++](#)
[PHP](#)
[GoLang](#)
[SQL](#)
[R Language](#)
[Android Tutorial](#)
[Tutorials Archive](#)

Data Science & ML

[Data Science With Python](#)
[Data Science For Beginner](#)
[Machine Learning](#)
[ML Maths](#)
[Data Visualisation](#)
[Pandas](#)
[NumPy](#)
[NLP](#)
[Deep Learning](#)

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Interview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved