Email + Social Logins in Django – Step by Step Guide

Last Updated : 24 Nov, 2022

This article revolves around a Django Project. It includes Email + Social Login integration in any Django project. We have used React as front end to demonstrate the working of the project. You can use your own front end to int

Requirements –

1. We need to register a user we need to pass username, email , first_name , password (because of our model) .
2. We need to log in that user and authenticate user.
3. User should be able to login with multiple Social Platforms.
4. We need to make API for all the requirements.

How will authentication happen:

Before starting, we need to know that we will send authenticated requests by sending a **Bearer** token in Authorization headers of the request .This token will tell the server ***which user*** *sent the request*. For that we will do these:

1. We will send the username and password to a endpoint to get the token and this will serve as our login
2. In return we will get access token and refresh token, we will set the authorization header of all requests to this access token
3. However access token will expire after a short time, then we will send the refresh token to the endpoint to get a new access token
4. Thus we will repeat step 2 and 3 internally without letting the user know when the access token expire

Before starting , let us go through some prerequisites:

- Basic knowledge of Django – Django Tutorial
- Create a django project (inside a virtual environment preferably) – Create a Demo Project in Django

- Postman installed in your system (you can download it from here or use any alternative of postman)

#Step 1: Creating Custom User Model in Django

After you have created a demo project. Create a app **Accounts**.

```
python manage.py startapp accounts
```

Then you can see that a new folder is created with accounts name, now lets add it to the **INSTALLED_APPS** in settings.py. So it should look something like this:

```
INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
   'django.contrib.sessions',
   'django.contrib.messages',
   'django.contrib.staticfiles',
    # add this
   'accounts',
]
```

Inside the **models.py** lets create the model **Account** and its **Manager** & also import these:

```
from django.db import models
from django.contrib.auth.models import
AbstractBaseUser,PermissionsMixin,BaseUserManager
from django.utils.translation import gettext_lazy as _

class Account(AbstractBaseUser,PermissionsMixin):
   email=models.EmailField(unique=True)
   username= models.CharField(_('User Name'),max_length=150)
   first_name = models.CharField(_('First Name'),max_length=150)
   last_name = models.CharField(_('last Name'),max_length=150)
   is_staff=models.BooleanField(default=False)
   is_active=models.BooleanField(default=True)
```

```
    objects=CustomAccountManager()

    USERNAME_FIELD='email'
    REQUIRED_FIELDS=['username','first_name']

    def __str__(self):
        return self.email
```

- here we are using **email** as default **USERNAME_FIELD** and we want username and first_name as required fields (you can also have any other fields **but username field should be there**
- **is_active** should be **True** by default and **is_staff** should be **False** by default
- for creating objects(users) we need a Custom manager (*which we are creating below*)
- write "**username**" instead of user_name or any other style because facebook or google login returns username
- *using gettextlazy is optional*

```
class CustomAccountManager(BaseUserManager):
    def
create_user(self,email,username,first_name,password,**other_fields):
        if not email:
            raise ValueError(_('Please provide an email address'))
        email=self.normalize_email(email)

user=self.model(email=email,username=username,first_name=first_name,**ot
her_fields)
        user.set_password(password)
        user.save()
        return user

    def
create_superuser(self,email,username,first_name,password,**other_fields)
:
```



```
        other_fields.setdefault('is_superuser',True)
        other_fields.setdefault('is_active',True)
        if other_fields.get('is_staff') is not True:
                raise ValueError(_('Please assign is_staff=True for
```

```
superuser'))
        if other_fields.get('is_superuser') is not True:
                raise ValueError(_('Please assign is_superuser=True for
superuser'))
        return
self.create_user(email,username,first_name,password,**other_fields)
```

Here create_user will create regular users while create_superuser is create super users (admin).

create_user

- for creating users we need to pass email, username, first_name and password as well as any other fields
- if no email then raise error otherwise normalize email (Read here)
- then create a model Account object with email, username, and other fields
- then set password, **set_password** actually sets the password as a **hashed** password in the model's object so **no one can see the actual password**
- then save the user object and return it

create_superuser

- is_staff, is_superuser, is_active should be set to True by default
- if not set to True or is passed False then raise errors else create_user with this values as well as other fields

Now add this in *settings.py* to use our custom user model :

```
AUTH_USER_MODEL='accounts.Account'
```

Then we need to migrate this model to database.

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

#Step 2: Making REST API Endpoints for Authentication

For that we need to install some libraries first which I will explain below why we need them:

```
pip install djangorestframework
pip install django-cors-headers
pip install drf_social_oauth2
```

- **djangorestframework** is for REST API Endpoints
- **django-cors**-headers is required so that our React app can communicate with the django server
- **drf_social_oauth2** – this is the main library which enables us oauth2 token based authentication for email password as well as google and facebook

Now inside the settings.py we should add these to INSTALLED_APPS for our app to work as expected:

```
INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
   'django.contrib.sessions',
   'django.contrib.messages',
   'django.contrib.staticfiles',
   # add these
   'rest_framework',
   'oauth2_provider',
   'social_django',
   'drf_social_oauth2',
    'corsheaders',
   # LOCAL
   'accounts',
]
```

Let's also add these configurations to settings.py which I'll explain below:

```
AUTHENTICATION_BACKENDS = (
  'drf_social_oauth2.backends.DjangoOAuth2',
  'django.contrib.auth.backends.ModelBackend',
)
REST_FRAMEWORK = {
   'DEFAULT_AUTHENTICATION_CLASSES': (
       'oauth2_provider.contrib.rest_framework.OAuth2Authentication',
```

```
        'drf_social_oauth2.authentication.SocialAuthentication',
    )
}
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://127.0.0.1:3000"
]
```

- **AUTHENTICATION_BACKENDS** – so that we can authenticate with either OAuth2**(token based)** or basic authentication(**without token**)
- **REST_FRAMEWORK** endpoint requests can be authenticated using tokens only
- **CORS_ALLOWED_ORIGINS** will be our frontend's address (here it's react website's address)

Alright now let's add some configuration to the MIDDLEWARE and TEMPLATES :

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
     # add these
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
]

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        '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',
                    # add these
                 'social_django.context_processors.backends',
                'social_django.context_processors.login_redirect',
            ],
        },
    },
]
```

Those are required by social logins and cors

Let's add some URL to django project's urls.py as those are required by both rest framework and oauth

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

urlpatterns = [
    path('admin/', admin.site.urls),
     # add these
    path('api-auth/', include('rest_framework.urls')),
    path('api-auth/', include('drf_social_oauth2.urls',namespace='drf')),
]
```

After adding this and that as required let's start coding again.

For Registering / Signing Up a user we need to write a serializer and view . Note that after signing up , that user needs to login by himself , that won't happen automatically. However I have covered that as well! So a user can register and our backend will login that user!!

Let's create the serializer, I have already created a serializers.py in accounts folder , so now write this code:

```
from rest_framework import serializers
from .models import Account

class RegistrationSerializer(serializers.ModelSerializer):
    class Meta:
        model=Account
        fields=('email','username','password','first_name')
```

```
        extra_kwargs={'password':{'write_only':True}}

    def create(self,validated_data):
        password=validated_data.pop('password',None)
        instance=self.Meta.model(**validated_data)
        if password is not None:
            instance.set_password(password)
        instance.save()
        return instance
```

Inside the RegistrationSerializer :

- class Meta has what model , fields will be serialized and password cannot be read it will only be write only (for forms)
- create method will create new instance of Account and will fill that with validated data which we pass from our view
- we need that password and if that password is not None we can set hashed password so we will pop it out of the validated data
- then we will save the instance and return it

Now we will write the view for creating users:

```
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework import status,generics
from rest_framework.response import Response
from .serializers import RegistrationSerializer
from rest_framework import permissions
from .models import Account

class CreateAccount(APIView):
   permission_classes=[permissions.AllowAny]

   def post(self,request):
       reg_serializer=RegistrationSerializer(data=request.data)
       if reg_serializer.is_valid():
           new_user=reg_serializer.save()
           if new_user:
              return Response(status=status.HTTP_201_CREATED)
```

```
        return
Response(reg_serializer.errors,status=status.HTTP_400_BAD_REQUEST)
```

- we import the necessary , allow anyone to access this view and only post requests to this view is allowed
- when post request happens we will initialize RegistrationSerializer with request.data and if the data is valid then save it else return error
- ***Note : we will revisit this view to login the user after saving the serialized data later on**

also create a urls.py in accounts directory and write this

```
from django.urls import path
from .views import CreateAccount

app_name = 'users'

urlpatterns = [
   path('create/', CreateAccount.as_view(), name="create_user"),]
```

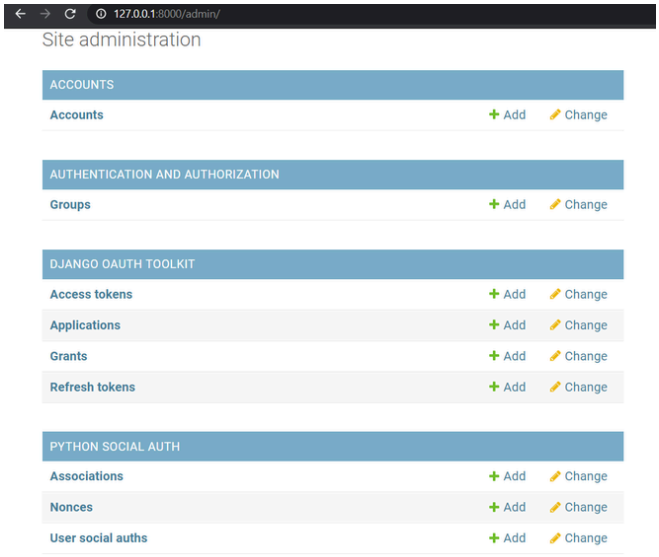
finally add this url to project's urls.py

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

urlpatterns = [
   path('admin/', admin.site.urls),
   path('api-auth/', include('rest_framework.urls')),
   path('api-auth/', include('drf_social_oauth2.urls',namespace='drf')),
   #add this
   path('api-auth/', include('accounts.urls'))
]
```

Now for login we **don't need to write any extra code** as login is nothing but getting access token from server and all these is already taken care of by drf-social-oauth2. We don't even need to write any endpoints for that !

We need to create a **Application** and get **client_id** and **client_secret** to get access tokens, So let's visit admin running on http://127.0.0.1:8000/admin/ and you should see something like this:



Admin

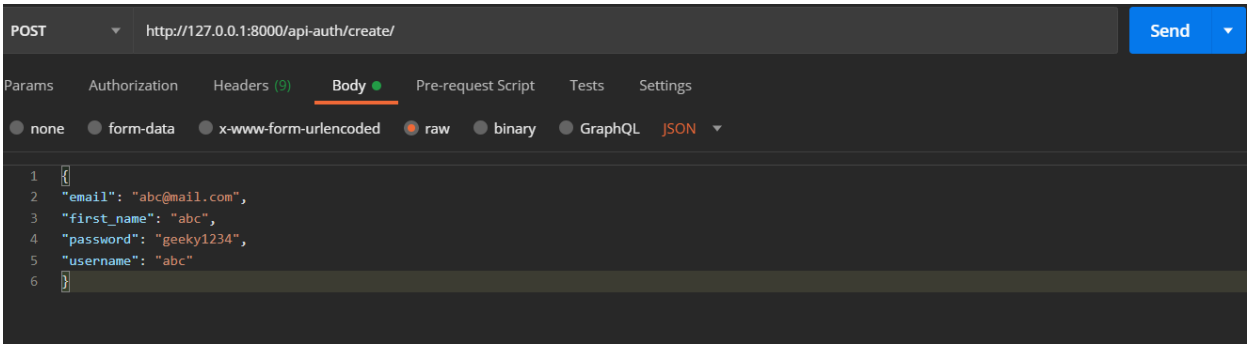
Here you can see Applications, press the add button of that and create a new application. **Don't touch the Client id and Client secret**, select user as your admin **superuser**, and Client type will be **confidential**, Authorization grant type will be **Resource owner password-based** , then save it.

That's it we are ready to check our application now so let's create a superuser and pass the required values and then run server inside virtual environment

```
python manage.py runserver
```

#Step 3 : Creating & Logging in user with email password

Now open Postman and write this URL and in the body write this and send the request!



Postman post request to create user

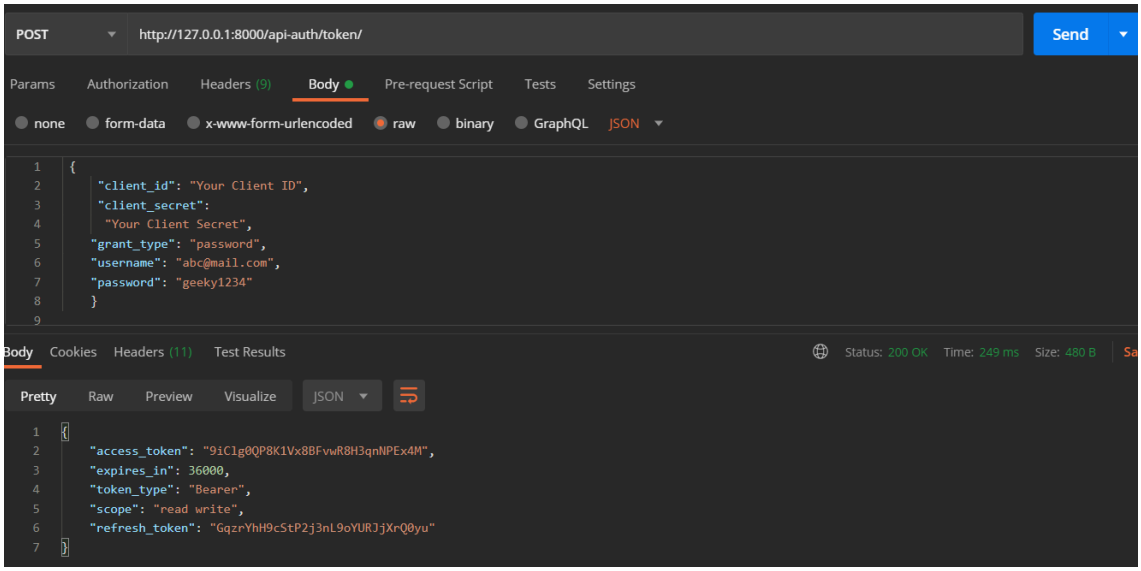
You should get a blank response like this

Status 201 created

Thus the **user is created** so it is working fine!

For sending request for token we will pass **client_id, client_secret**, **email** as username, **password** and **grant_type=password** to let server know this is for **getting fresh** new access_token and refresh token

Now we can send requests for token, so back in postman send this request to http://127.0.0.1:8000/api-auth/token :



Request returns Token

Note that we pass email to username field in request as passing username is mandatory

- It should return **access token and refresh token** as well as **status of 200 Ok**
- This access_token can be set on **Authorization headers** of request to send Authenticated requests (we will do that with react)

Remember I said we will revisit that CreateAccount view to login the user after saving the serialized data , Lets do that

```
import requests # add this

class CreateAccount(APIView):
    permission_classes=[permissions.AllowAny]

    def post(self,request):
        reg_serializer=RegistrationSerializer(data=request.data)
```

```
        if reg_serializer.is_valid():
            new_user=reg_serializer.save()
            if new_user:
               #add these
                r=requests.post('http://127.0.0.1:8000/api-auth/token',
data = {
                    'username':new_user.email,
                    'password':request.data['password'],
                    'client_id':'Your Client ID',
                    'client_secret':'Your Client Secret',
                    'grant_type':'password'
                })
                return Response(r.json(),status=status.HTTP_201_CREATED)
        return
Response(reg_serializer.errors,status=status.HTTP_400_BAD_REQUEST)
```

So after saving the new user we will send a post request to get the token and send it back as Response. This will automatically login the user after signup!

Google and Facebook authentications need a frontend for complete demonstrations so I will show you that with react , however any frontend can do if you know what to do ! (I'll do that after the necessary steps)

#Step 4 : Authenticated request demonstration preparation

We will create a different serializer which will return info about users and current user through two different views, one will be authenticated request and another will be a non-authenticated request.

so the serializer will look like this :

```
class UsersSerializer(serializers.ModelSerializer):
   class Meta:
        model=Account
        fields=('email','username','first_name')
```

We will write this two views :

- **AllUsers** returns all users and any one can view data
- **CurrentUser** which returns only the current user and only authenticated requests allowed

```
from rest_framework import status,generics

class AllUsers(generics.ListAPIView):
   permission_classes=[permissions.AllowAny]
   queryset=Account.objects.all()
   serializer_class=UsersSerializer

class CurrentUser(APIView):
   permission_classes = (permissions.IsAuthenticated,)
   def get(self, request):
       serializer = UsersSerializer(self.request.user)
       return Response(serializer.data)
```

urls.py will look like this :

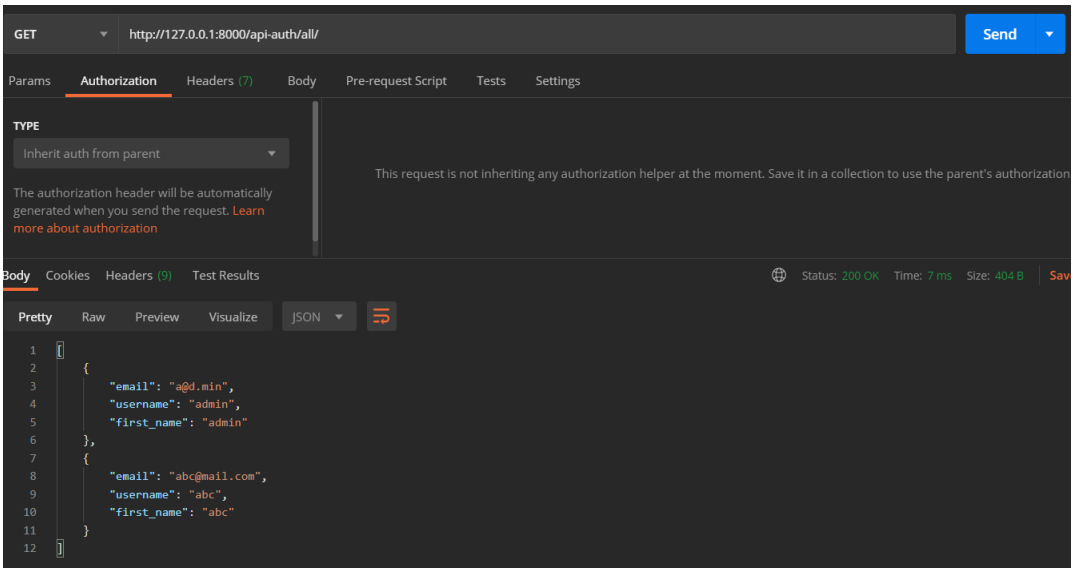
```
from django.urls import path
from .views import CreateAccount,AllUsers,CurrentUser

app_name = 'users'

urlpatterns = [
   path('create/', CreateAccount.as_view(), name="create_user"),
   path('all/', AllUsers.as_view(), name="all"),
   path('currentUser/', CurrentUser.as_view(), name="current"),
   ]
```

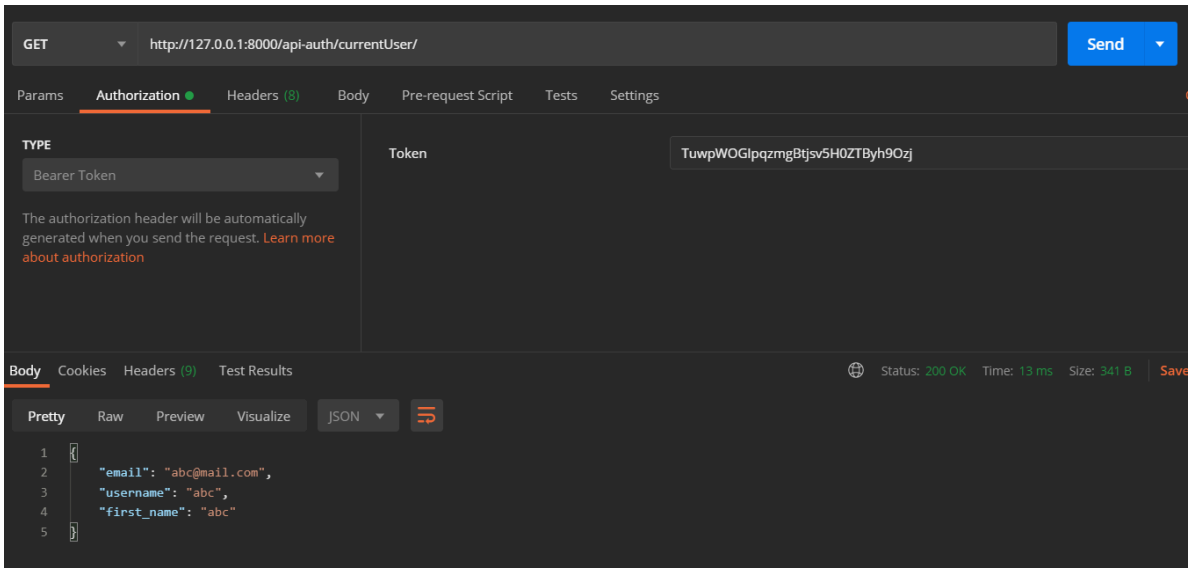
#Step 5 : Authenticated request demonstration

So let's send non-authenticated request first and it returns this response (**Note that Authorization header has nothing in it**)

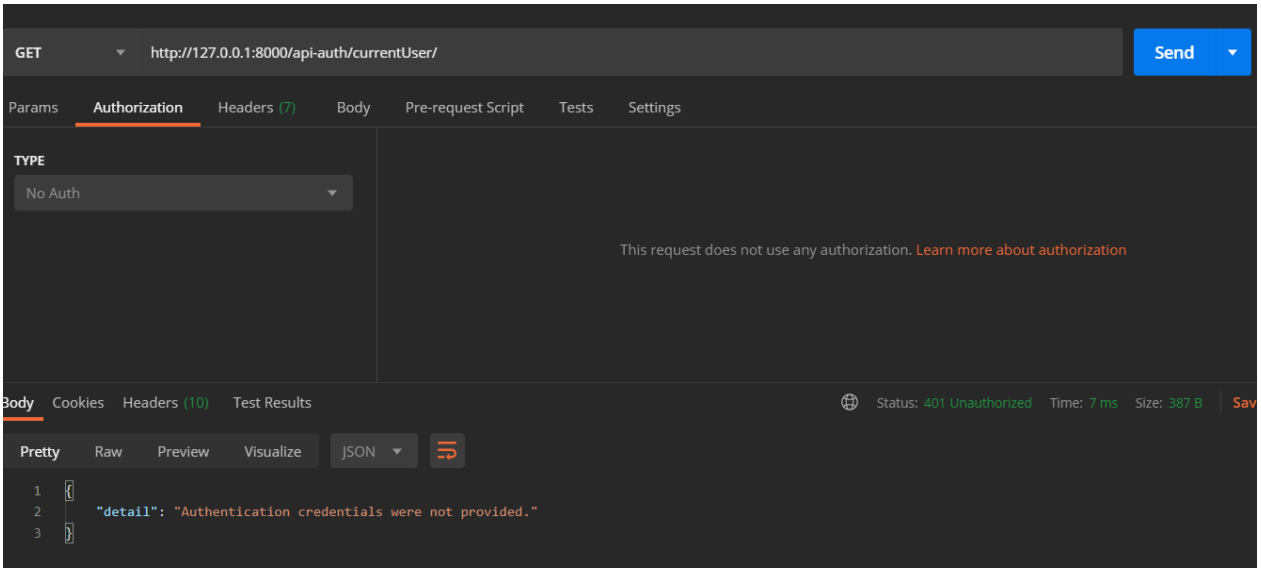


Now let's get the access token we previously **received on step 3**

- set the Authorization type to **Bearer** token from dropdown
- send a **authenticated request** to http://127.0.0.1:8000/api-auth/currentUser/



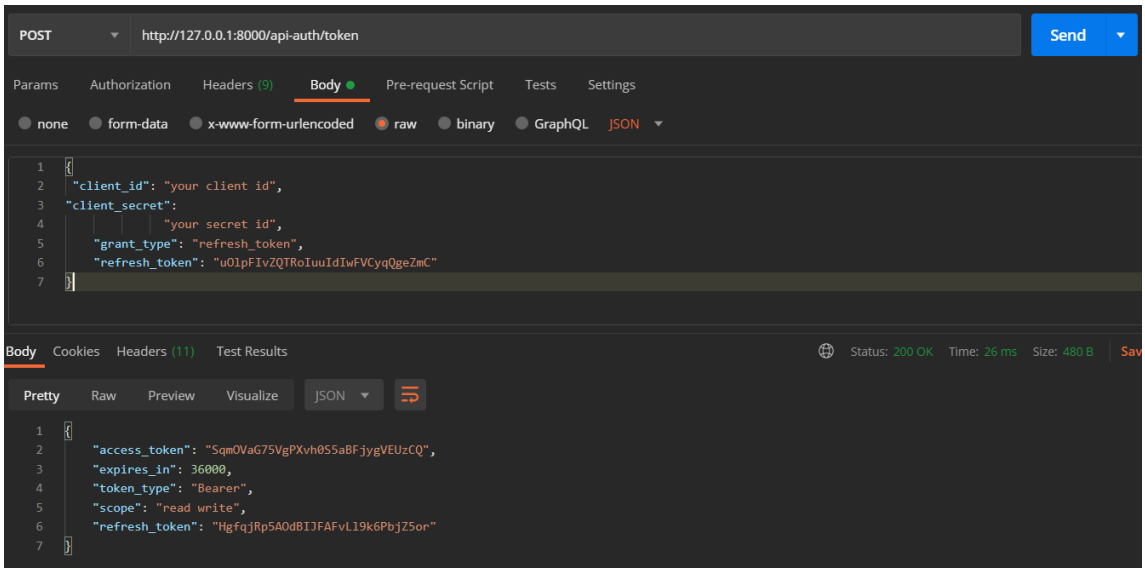
It will return the current user now if you have by chance sent the request without Authorization header it would have returned like this



Now what if the access token expires what to do then ??

For that we need to send refresh token to the same endpoint to get new access and refresh token

Let me show you how ! So we send the request to /token with **client_id, client_secret, grant_type = refresh_token** (so that server understands request has a refresh token which then converts **old** access and refresh token **to new** access and refresh token) and **refresh_token** will be the token we received in third step



It returns new tokens as a response you can use this new access token as **Bearer token** to Authenticate requests

#Step 6 : Facebook and Google Login

Add these in settings.py and to get required **keys for facebook and google** you need to visit here for fb and here for google and perform the necessary steps there.

```
AUTHENTICATION_BACKENDS = (
   'social_core.backends.google.GoogleOAuth2',
   'social_core.backends.facebook.FacebookAppOAuth2',
   'social_core.backends.facebook.FacebookOAuth2',

  'drf_social_oauth2.backends.DjangoOAuth2',
  'django.contrib.auth.backends.ModelBackend',
)
# Facebook configuration
SOCIAL_AUTH_FACEBOOK_KEY = 'your facebook key'
SOCIAL_AUTH_FACEBOOK_SECRET = 'your facebook secret'

# Define SOCIAL_AUTH_FACEBOOK_SCOPE to get extra permissions from
Facebook.
# Email is not sent by default, to get it, you must request the email
permission.
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email']
SOCIAL_AUTH_FACEBOOK_PROFILE_EXTRA_PARAMS = {
   'fields': 'id, name, email'
}
SOCIAL_AUTH_USER_FIELDS=['email','first_name','username','password']

SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = "your google oauth2 key"
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = "your google oauth2 secret"

# Define SOCIAL_AUTH_GOOGLE_OAUTH2_SCOPE to get extra permissions from
Google.
SOCIAL_AUTH_GOOGLE_OAUTH2_SCOPE = [
   'https://www.googleapis.com/auth/userinfo.email',
   'https://www.googleapis.com/auth/userinfo.profile',
]
```

#Step 7 : Google and Facebook Login using React JS

Before using this part of tutorial, checkout React JS Tutorial. After you have made a react App, perform the following steps –

Its very easy to authenticate once you have the keys , now we need to *install axios , react-facebook-login , react-google-login*

Then create a login component where we render these **fb and google login button** which will take **required keys** and every user when tries to login with these buttons, will return an access token as well as info about user

```
import ReactFacebookLogin from "react-facebook-login";
import ReactGoogleLogin from "react-google-login";
import { facebookLogin, googleLogin } from "../axios";# I'll create this
later

export default function LogIn() {

  function responseFb(response) {
    console.log(response);
    facebookLogin(response.accessToken);
  }
  function responseGoogle(response) {
    console.log(response);
    googleLogin(response.accessToken);
  }

  return (
    <>
    <ReactFacebookLogin
           appId="Your App Id"
           fields="name,email"
           callback={responseFb}
         />
         <ReactGoogleLogin
           clientId="your google client id"
           buttonText="Login"
           onSuccess={responseGoogle}
           onFailure={responseGoogle}
           cookiePolicy={"single_host_origin"}
         />
    </>
```

```
  );
}
```

However **our server has no clue** that we have logged in a user as this access token is returned from google , facebook and **not from our server**.

So how to let our server know ? Send this token to our server or set this token as our Authorized Bearer token ? No in that case we cannot create a account object with our Accounts model or access any account object as current user thus our user will not be technically registered or connected.

So we need to **convert this token** to get **access token, refresh token from our server**, (registering our user) and for that I **created a axios.js file** and there I put this code:

```
export function facebookLogin(accessToken) {
 axios
   .post(`http://127.0.0.1:8000/api-auth/convert-token`, {
     token: accessToken,
     backend: "facebook",
     grant_type: "convert_token",
     client_id: "your client id",
     client_secret:"your client secret ",
   })
   .then((res) => {
    // Save somewhere these access and refresh tokens
     console.log(res.data);
   });
}
export function googleLogin(accessToken) {
 axios
   .post(`http://127.0.0.1:8000/api-auth/convert-token`, {
     token: accessToken,
     backend: "google-oauth2",
     grant_type: "convert_token",
     client_id: "your client id",
     client_secret: "your client secret",
   })
   .then((res) => {
    // Save somewhere these access and refresh tokens
```

```
      console.log(res.data);
    });
}
```

what we are doing here is:

1. make a post request to **http://127.0.0.1:8000/api-auth/convert-token** to convert the token
2. backend will be the backend from where you got the access token to convert
3. **grant_type** will be **convert_token** to let server know we want this token to be converted to our server's access token
4. client_id and client_secret will be like previous steps

So how to refresh a token if access tokens expire ?? We want that to **happen automatically** without letting the user bothered.

- For that we will use **axios interceptors** which will return responses but if there is error it will handle those error
- if error status is **401 (Unauthorized request**) then if there is no refresh token in local storage then it will ask the user to login , and if refresh token exists then it will send that to http://127.0.0.1:8000/api-auth/token to **get new access and refresh tokens**
- Please note I am using local storage to store access tokens as an example but you should a secure method probably **Web Cookies (Secure, HttpOnly, Same Site)**
- **window needs to be reloaded** to let react fetch the website authenticates with latest access token

```
axiosInstance.interceptors.response.use(
 (response) => {
   return response;
 },
 async function (error) {
   const originalRequest = error;
   console.log(originalRequest);

   if (typeof error.response === "undefined") {
     alert("a server error happNeD, we will fix it shortly");
```

```
      return Promise.reject(error);
    }

    if (
      error.response.status === 401 &&
      !localStorage.getItem("refresh_token")
    ) {
      window.location.href = "/login/";
      return Promise.reject(error);
    }

    if (
      error.response.status === 401 &&
      error.response.statusText === "Unauthorized" &&
      localStorage.getItem("refresh_token") !== undefined
    ) {
      const refreshToken = localStorage.getItem("refresh_token");

      return axios
        .post("http://127.0.0.1:8000/api-auth/token", {
          client_id: "Your client id ",
          client_secret:
            "Your client secret",
          grant_type: "refresh_token",
          refresh_token: refreshToken,
        })
        .then((response) => {
          localStorage.setItem("access_token",
response.data.access_token);
          localStorage.setItem("refresh_token",
response.data.refresh_token);
          window.location.reload();
          axiosInstance.defaults.headers["Authorization"] =
            "Bearer " + response.data.access_token;
        })
        .catch((err) => console.log(err));
    }
  }
);
```

This way we can authenticate a user from Django and React using social logins.

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!

A abhik...

7

Previous Article

Render HTML Forms (GET & POST) in Django

Next Article

Similar Reads

Django settings file - step by step Explanation

Once we create the Django project, it comes with a predefined Directory structure having the following files with each file having its own uses. Let's take an...

3 min read

Step-by-Step Guide to Using RANSAC in OpenCV using Python

RANSAC (Random Sample Consensus) is an iterative method to estimate the parameters of a mathematical model from a set of observed data that contains...

4 min read

Setting Up Docker for Python Projects: A Step-by-Step Guide

Docker provides a standardized environment to develop, test and deploy applications in an isolated container ensuring that your code works seamlessly...

5 min read

Django Sign Up and login with confirmation Email | Python

Django by default provides an authentication system configuration. User objects are the core of the authentication system. Today we will implement Django's...

7 min read

How to Send Email with Django

Django, a high-level Python web framework, provides built-in functionality to send emails effortlessly. Whether you're notifying users about account...

4 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

How to customize Django forms using Django Widget Tweaks ?

Django forms are a great feature to create usable forms with just few lines of code. But Django doesn't easily let us edit the form for good designs. Here, we...

3 min read

Styling Django Forms with django-crispy-forms

Django by default doesn't provide any Django form styling method due to which it takes a lot of effort and precious time to beautifully style a form. django-crispy...

1 min read

Integrating Django with Reactjs using Django REST Framework

In this article, we will learn the process of communicating between the Django Backend and React js frontend using the Django REST Framework. For the sake...

15+ min read

How to Fix Django "ImportError: Cannot Import Name 'six' from...

After Django 3.0, django.utils.six was removed. The "ImportError: Cannot Import Name 'six' from 'django.utils'" error occurs because of the dependency issue. Thi...

3 min read

Article Tags : Python Technical Scripter Python Django Technical Scripter 2020

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

GET IT ON Google Play Download on the App Store

Company
About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation

Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Pandas
NumPy
NLP
Deep Learning

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial

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

Computer Science

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

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

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

Inteview Preparation

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

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

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

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved