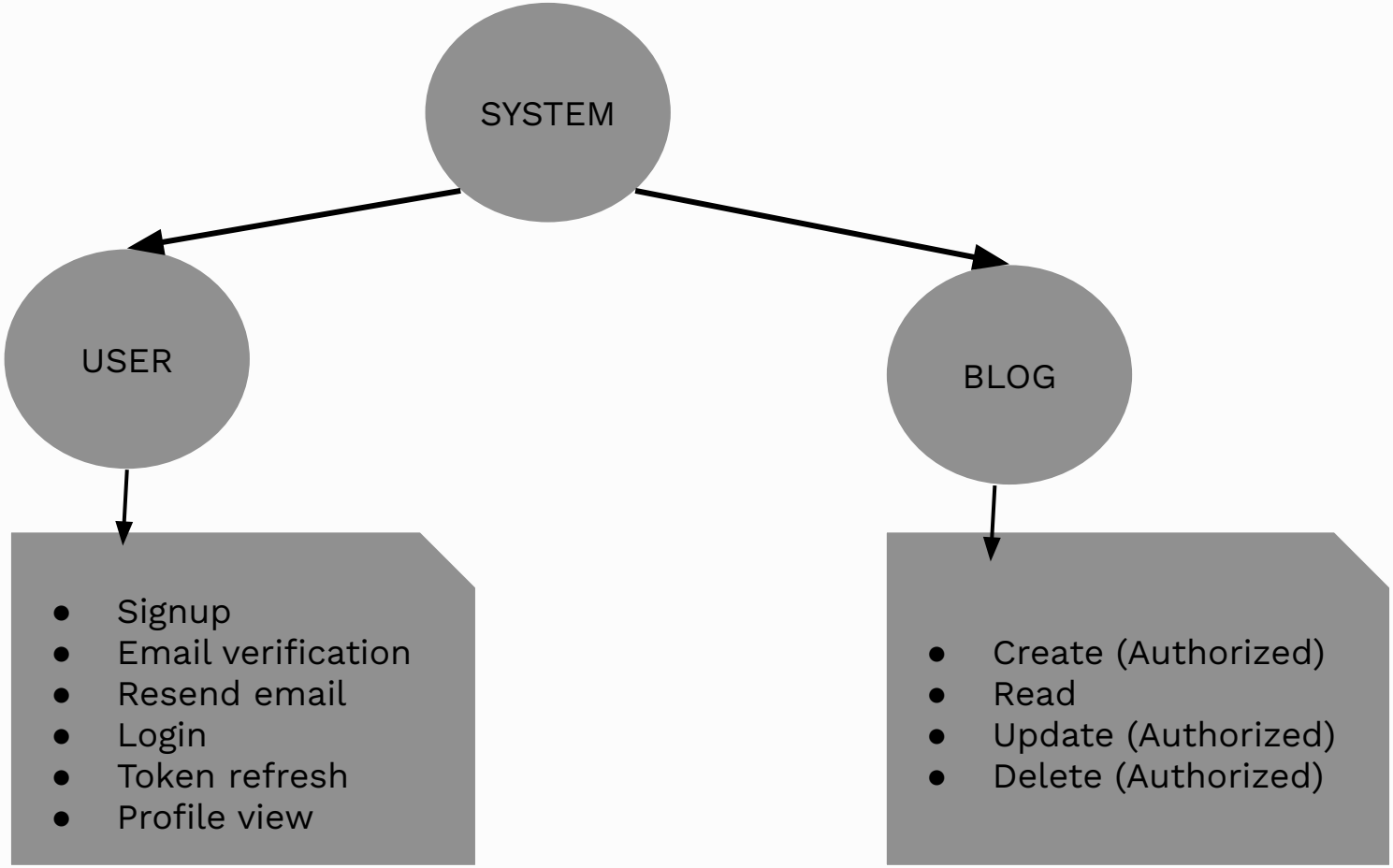
Django Rest Framework

A Project First approach

MD Amirul Islam Nahid

Django Rest Framework

2025


SYSTEM
USER
BLOG
Signup
Email verification
Resend email
Login
Token refresh
Profile view
Create (Authorized)
Read
Update (Authorized)
Delete (Authorized)

INSTALLATION

Place it into your requirements.txt

```
asgiref==3.8.1
Django==5.1.1
djangorestframework==3.15.2
djangorestframework-simplejwt==5.3.1
pillow==10.4.0
PyJWT==2.9.0
sqlparse==0.5.1
typing_extensions==4.12.2
```

- `djangorestframework-simplejwt` : for token management
- `pillow`: for handling image

Project setup

Run below commands into your terminal

```
django-admin startproject src .
```

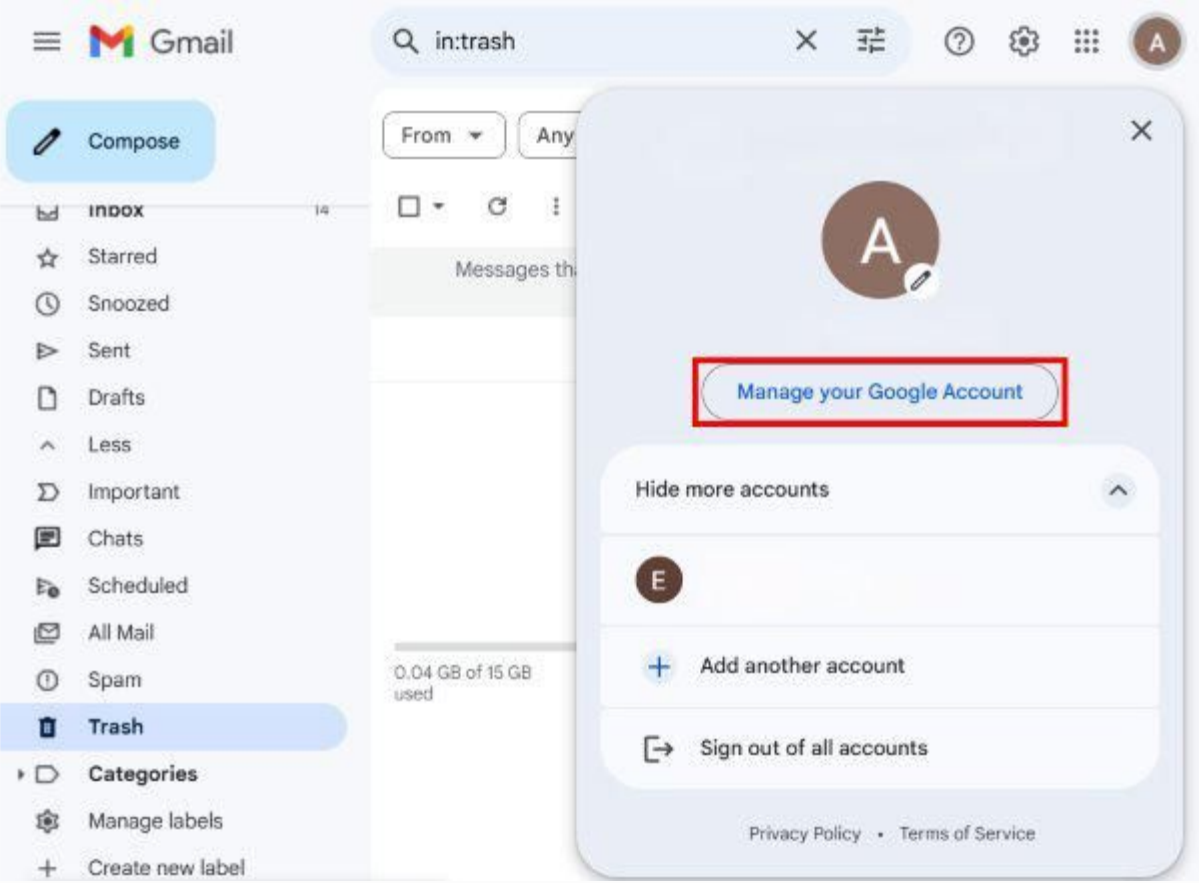
```
django-admin startapp account blog
```

Register our apps in settings.py file

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

   # our app
   'account',
   'Blog',

   # 3rd party
   'rest_framework',
   'rest_framework_simplejwt',
]
```

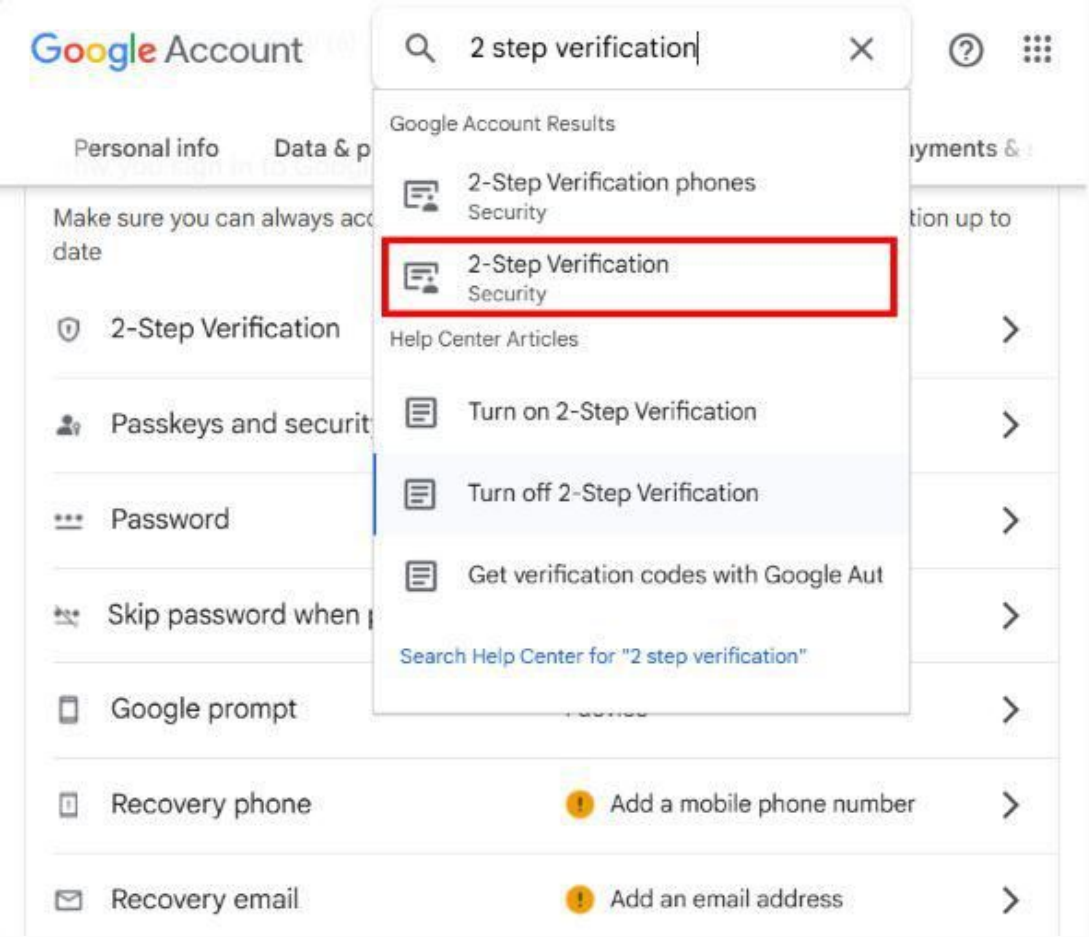
Gmail Setup (Generate App Password)

Step 1
Sign in to your google account and click on Manage account.



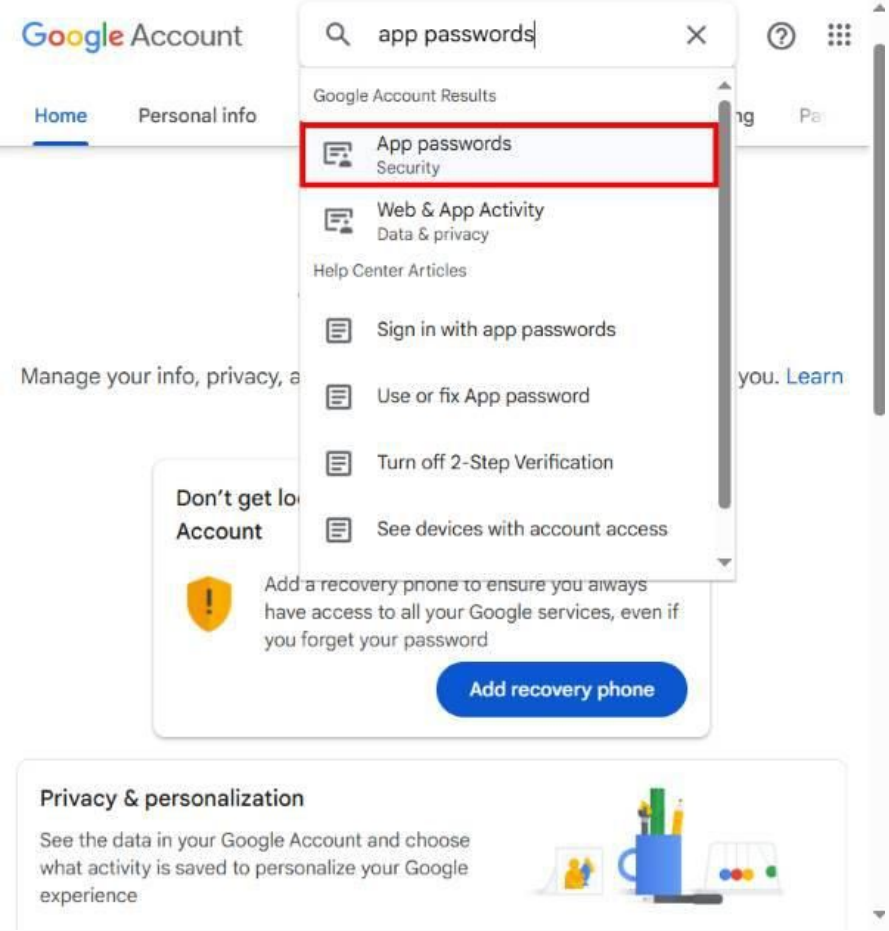
Gmail Setup (Generate App Password)

Step 2
Enable two factor authentication by typing “2 step verification” on the search bar.



Gmail Setup (Generate App Password)

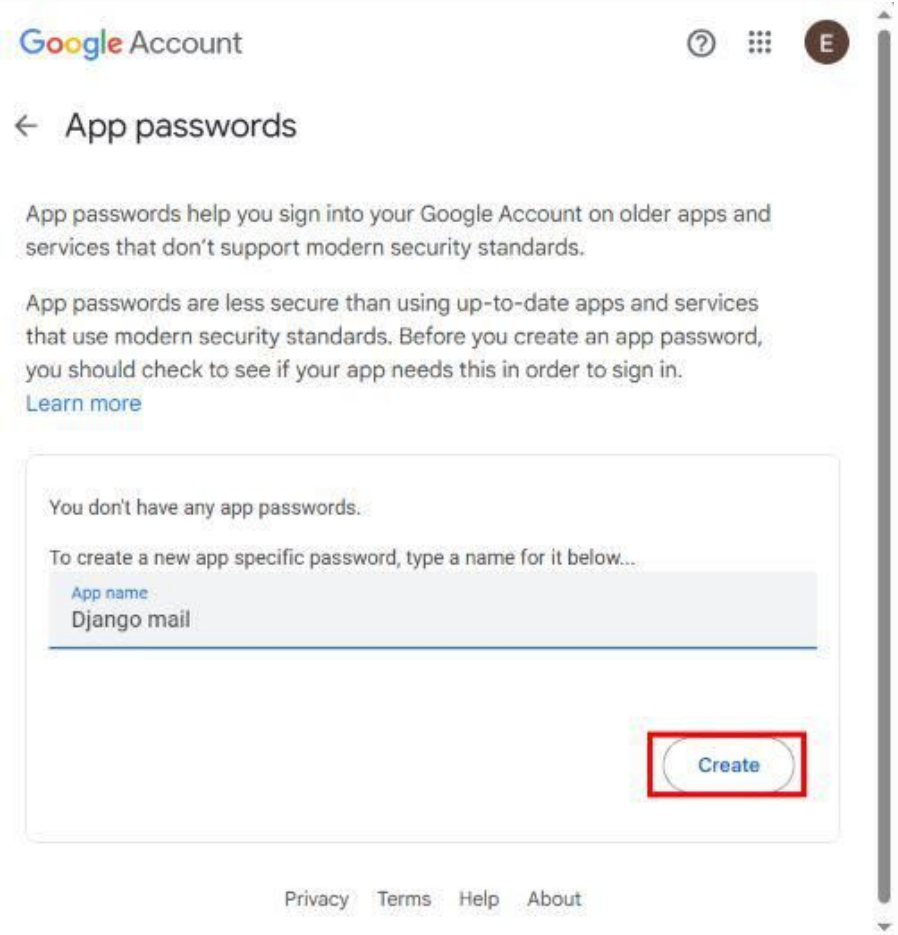
Step 3
After enabling 2FA, type “App passwords” on the search bar.



Gmail Setup (Generate App Password)

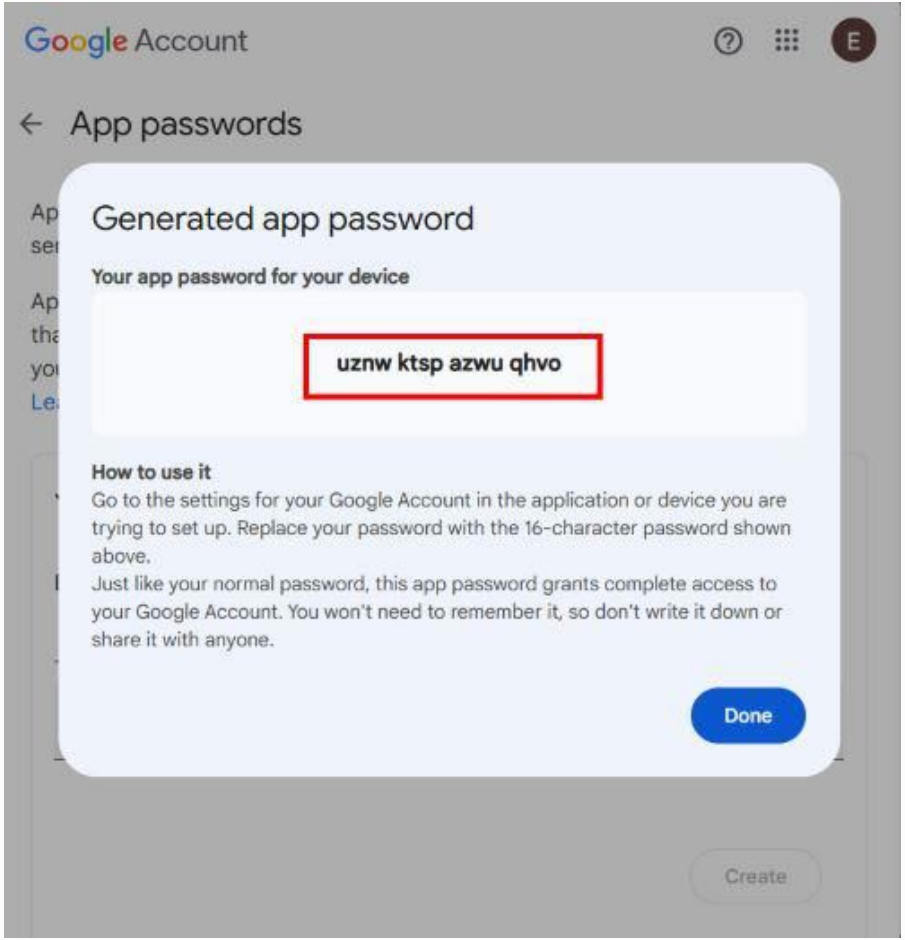
Step 4

Enter a project name and your unique password will be provided to use it in your project. Note that the password is only shown once, and it is valid until it is manually removed from the app passwords tab. Hence copy the password when it is shown and use it as `EMAIL_HOST_PASSWORD` in the Django settings.py file.



Gmail Setup (Generate App Password)

Copy this



Email setup in settings.py

```
# Email settings (enable 2FA in gmail , generate app password)
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'amirul.islam@vivasoftltd.com'
EMAIL_HOST_PASSWORD ='uzorvsdphegmirnm'
```

Authentication settings for token management (settings.py)

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    )
}

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=60),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
}
```

Media settings (settings.py)

Where the uploaded images will go ? We will setup this into settings.py file

```
# Media settings
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

Create our models: User

```
# accounts/models.py
from django.db import models
from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):
   is_verified = models.BooleanField(default=False)
   verification_token = models.CharField(max_length=32, blank=True, null=True)
   bio = models.TextField(blank=True)
   image = models.ImageField(upload_to='profle_images/', blank=True, null=True)
```

Set the CustomUser as default auth model (settings.py)

```
# User Model
AUTH_USER_MODEL = 'account.CustomUser' # 'app_name.ModelName'
```

Create Our Model: Blog

```
# blog/models.py
from django.db import models
from django.conf import settings
from django.utils.text import slugify
from datetime import datetime

class Blog(models.Model):
   title = models.CharField(max_length=255)
   content = models.TextField()
   slug = models.SlugField(unique=True, blank=True)  # New field for slug
   author = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name='blogs')
   created_at = models.DateTimeField(auto_now_add=True)
   updated_at = models.DateTimeField(auto_now=True)

   def __str__(self):
       return self.title

   def save(self, *args, **kwargs):
       # Automatically generate a slug if it is not set
       if not self.slug:
           self.slug = f"{slugify(self.title)}_{hash(datetime.now)}"
       super().save(*args, **kwargs)
```

Apply migration

```
python3 manage.py makemigrations
```

```
python3 manage.py migrate
```

Email Verification Template

- Create ‘templates/’ folder under accounts/
- Create ‘emails/’ folder under templates/
- Create ‘verification_email.html’ under emails/

```
accounts
     ├─ emails
          ├─ verification_email.html
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Verify your email</title>
</head>
<body>
   <h1>Hello {{user}}</h1>
   <p>Click the below link to verify you account</p>
   <a href="{{verification_link}}">Verify</a>
   <p>If this is not you, then ignore this mail.</p>
</body>
</html>
```

BASE URL SETUP : for account

```
# src/urls.py
from django.conf.urls.static import static

urlpatterns = [
   path('admin/', admin.site.urls),
   path('api/account/', include('account.urls')),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Serializer Setup for (SIgnup)

Import these, in case of some serializers aren’t implemented , then comment that line.

```
# accounts/serializers.py
from django.utils.crypto import get_random_string
from rest_framework import serializers
from .models import CustomUser
from django.urls import reverse
from django.template.loader import render_to_string
from django.core.mail import EmailMultiAlternatives
```

```python
# accounts/serializers.py

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = CustomUser
        fields = [
            'id',
            'username',
            'email',
            'password',
            'bio',
            'image'
        ]
        extra_kwargs = {"password" : {"write_only" : True}}

    def create(self, validated_data):
        user = CustomUser(**validated_data)
        user.set_password(validated_data['password'])

        user.verification_token = get_random_string(length=32)

        user.save()

        self.send_email(user=user)

        return user
```

send_email() method in the next slide

```python
def send_email(self, user):
    # verification_link = f"http://127.0.0.1:8000/api/account/verify-email/{user.verification_token}/"

    # Prepare things for sending mail
    verification_link = self.context['request'].build_absolute_uri(
        reverse(
            viewname='verify_email', # the url-name that handle email verification
            kwargs = {'token' : user.verification_token}
        ),
    )

    # Render the email template
    subject = 'Verify you email'
    html_content = render_to_string('emails/verification_email.html', {
        "user": user.username,
        "verification_link" : verification_link
    })

    # create an email message

    email = EmailMultiAlternatives(
        subject,
        "This is a plain text version of the email",
        "from@example.com",
        [user.email]
    )

    email.attach_alternative(html_content, "text/html")
    email.send(fail_silently=False)

    return True
```

This is the template where write our html template

Writing our views for signup

```
# accounts/views,py
from rest_framework import generics
from .serializers import UserSerializer


class UserSignUp(generics.CreateAPIView):
   serializer_class = UserSerializer
```

Writing the urls

```
# accounts/urls.py

urlpatterns = [
   path('signup/', UserSignUp.as_view(), name='signup'),
]
```

Write views for verifying the user email

```
from rest_framework import generics, status
from .serializers import UserSerializer
from .models import CustomUser
from rest_framework.response import Response

class VerifyEmail(generics.GenericAPIView):
   swagger_fake_view = True  # Bypass schema generation for this view
   def get(self, request , token):
       user = CustomUser.objects.filter(verification_token=token).first()

       if user:
           if user.is_verified:
               return Response({
                   "details" : "Email already verified!",
               }, status=status.HTTP_400_BAD_REQUEST)

           user.is_verified = True
           user.verification_token = None
           user.save()

           return Response({
               "details" : "Successfully verified!",
           }, status=status.HTTP_200_OK)

       return Response({
           "details" : "Invalid token",
       }, status=status.HTTP_400_BAD_REQUEST)
```

Writing urls for verify email

```
# accounts/urls.py

urlpatterns = [
   path('signup/', UserSignUp.as_view(), name='signup'),
   path('verify-email/<str:token>/', VerifyEmail.as_view() , name='verify_email'),
]
```

This **name** should be same as what we wrote in the send email section

Test the signup api

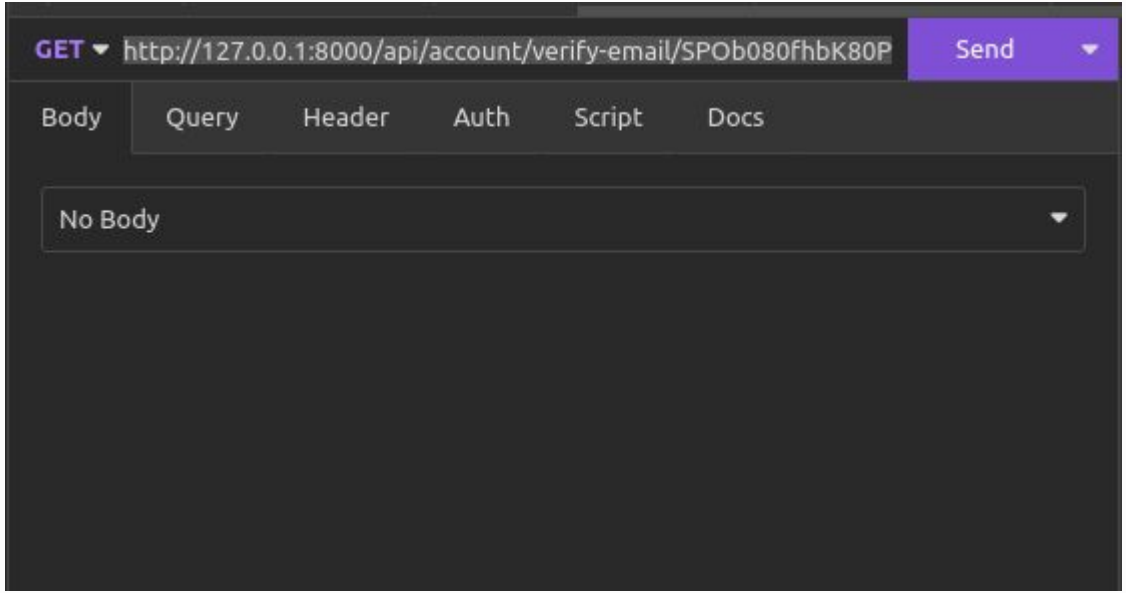
POST http://127.0.0.1:8000/api/account/signup/ Send

Body (JSON) Query Header (1) Auth Script Docs

JSON

```
1 ▼ {
2       "username" : "example",
3       "email" : "example@yopmail.com",
4       "password" : "example"
5   }
```

Test the verify account api



Complete import for views

```
# accounts/views.py
from django.shortcuts import render
from rest_framework import generics, status
from .serializers import (
   UserSerializer,
   UserLoginSerializer,
   UserUpdateSerializer
)
from .models import CustomUser
from rest_framework.response import Response
from django.core.mail import EmailMultiAlternatives
from django.template.loader import render_to_string
from django.utils.crypto import get_random_string
from django.urls import reverse
from rest_framework_simplejwt.tokens import RefreshToken
from rest_framework.permissions import IsAuthenticated
```

Some import may not work for now, just comment that part, when we implement those , comment out then

Views for ‘Resend Verification Email’

There’s no need to implement a serializer here. Just some functionality needed to be implement in a view function.

```
class ResendVerificationEmail(generics.GenericAPIView):
  swagger_fake_view = True  # Bypass schema generation for this view
  def post(self, request, *args, **kwargs):
     email = request.data.get('email')

     if email:
        user = CustomUser.objects.filter(email=email).first()

        if not user:
           return Response({
              "details" : "User with this email doesn't exist!",
           }, status=status.HTTP_404_NOT_FOUND)

        if user.is_verified:
           return Response({
              "details" : "Email already verified!",
           }, status=status.HTTP_400_BAD_REQUEST)


        user.verification_token = get_random_string(length=32)
        user.save()

        # Prepare things for sending mail
        verification_link = request.build_absolute_uri(
           reverse(
              viewname='verify_email',
              kwargs = {'token' : user.verification_token}
           ),
        )
       # rest of the code at the same indentation layer
```

ResendVerificationEmail Views Rest code

```
# Render the email template
subject = 'Verify you email'
html_content = render_to_string('emails/verification_email.html', {
    "user": user.username,
    "verification_link" : verification_link
})

# create an email message

email = EmailMultiAlternatives(
    subject,
    "This is a plain text version of the email",
    "from@example.com",
    [user.email]
)

email.attach_alternative(html_content, "text/html")
email.send(fail_silently=False)

return Response({
        "details" : "Verfication email sent!",
    }, status=status.HTTP_200_OK)
```

Make the Url for resent email verification

```
# accounts/urls.py
from django.urls import path
from .views import (
   UserSignUp,
   ResendVerificationEmail,
   VerifyEmail,
)

urlpatterns = [
   path('signup/', UserSignUp.as_view(), name='signup'),
   path('verify-email/<str:token>/', VerifyEmail.as_view() , name='verify_email'),
   path('resend-verification/', ResendVerificationEmail.as_view(), name= 'resend_verification'),

]
```

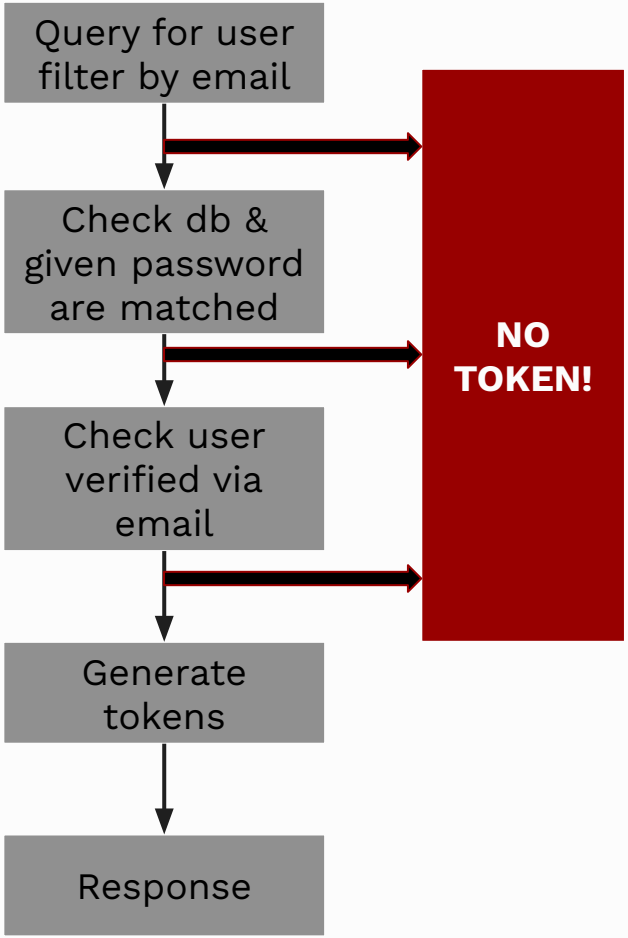
Write the Serializer for User Login

Look , this is just ‘serializers.Serializer’ instead of ‘serializers.ModelSerializer’ . Just because these serializer just an schema to take user input for login, No relationship with a model.

```
# accounts/serializers.py
class UserLoginSerializer(serializers.Serializer):
   email = serializers.EmailField()
   password = serializers.CharField()
```

Write the View for User Login

```
class UserLogin(generics.GenericAPIView):
    serializer_class = UserLoginSerializer

    def post(self, request):
        email = request.data.get('email')
        password = request.data.get('password')

        user = CustomUser.objects.filter(email=email).first()

        if user:
            matched_password = user.check_password(password)

            if matched_password:
                if not user.is_verified:
                    return Response({
                        "details" : "Email is not verified yet!",
                    }, status=status.HTTP_401_UNAUTHORIZED)

                refresh = RefreshToken.for_user(user)

                return Response({
                    "refresh_token" : str(refresh),
                    "access_token" : str(refresh.access_token)
                })

        return Response({
            "details" : "Invalid credentials",
        }, status=status.HTTP_401_UNAUTHORIZED)
```



Test User Login

POST http://127.0.0.1:8000/api/account/login/ Send

Body (JSON) | Query | Header (1) | Auth | Script | Docs

JSON

```
1 {
2     "email" : "kishan@yopmail.com",
3     "password" : "kishan"
4 }
```

200 OK | 709 ms | 495 B | 8 days ago | 27-Jan-25 05:20:35 PM | ...

Preview | Header | Request | Tests | Timeline


```
1 {
2     "refresh_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjo
icmVmcmVzaCIsImV4cCI6MTczODA2MzIzNSwiaWF0IjoxNzM3OTc2ODM1
LCJqdGkiOiJiZWNjMTVhNThjYzY0MGJmYTk2ZmE2ODRjOGYwYjE5NSIsI
nVzZXJfaWQiOjN9.m6OIsDhYAe1Ouk3pyy7tXKkTBHVS1aq5QrMwRH0j3
no",
3     "access_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjo
iYWNjZXNzIiwiZXhwIjoxNzM3OTgwNDM1LCJpYXQiOjE3Mzc5NzY4MzUs
Imp0aSI6IjVhZTVjZTViMTZjMTQ5NTJiZDIyMzEyNDM2OTY3Njc1Iiwid
XN1cl9pZCI6M30._edQfrKql5qjOBKb4PqugSOb_XeVzCQDZD86AeEjGi
c"
4 }
```

Profile Update & Get

What do we need now?

- We will create a serializer for profile update
- The profile get and update must be authenticated
- Now we can take two approach
  - **Approach-1:** Create two separate router say for example (get_profile/) & (update_profile/) as well as two separate view functions that will handle two routers
  - **Approach-2:** Create single router and one single view function to handle both task (get the profile details and also update the profile). Yes! Django Rest Framework has this power. And I prefer to choose this approach for now.

Serializer for profile update

```
# accounts/serializers.py
class UserUpdateSerializer(serializers.ModelSerializer):
    class Meta:
        model = CustomUser
        fields = ["bio", "image"]

    def update(self, instance ,validated_data):
        instance.bio = validated_data.get('bio' , instance.bio)
        instance.image = validated_data.get('image', instance.image)

        instance.save()
        return instance
```

`update()` method is being overridden as we inherit `ModelSerializer()`

Views for profile update and get the details

```
# accounts/views.py
class RetrieveUpdateProfile(generics.RetrieveUpdateAPIView):
   queryset = CustomUser.objects.all()
   permission_classes = [IsAuthenticated]

   def get_object(self):
       return self.request.user

   def get_serializer_class(self):
       if self.request.method in ['PUT' , 'PATCH']:
           return UserUpdateSerializer
       return UserSerializer
```

`get_object()` & `get_serializer_class()` methods are being overridden as we inherit from `generics`

Make the url

```
from django.urls import path
from .views import (
   UserSignUp,
   ResendVerificationEmail,
   VerifyEmail,
   UserLogin,
   RetrieveUpdateProfile
)


urlpatterns = [
   path('signup/', UserSignUp.as_view(), name='signup'),
   path('verify-email/<str:token>/', VerifyEmail.as_view() , name='verify_email'),
   path('resend-verification/', ResendVerificationEmail.as_view(), name= 'resend_verification'),
   path('login/', UserLogin.as_view(), name='login'),
   path('profile/', RetrieveUpdateProfile.as_view(), name='profile'),
]
```

Test this endpoint (get the profile details)

GET http://127.0.0.1:8000/api/account/profile/ Send

Body Query Header Auth (Bearer) Script Docs

Bearer Token

Enabled	☑
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWN
Prefix	

200 OK 44 ms 171 B 8 days ago | 27-Jan-25 04:08:44 PM | ht...

Preview Header Request Tests Timeline


```
1 ▼ {
2       "id": 3,
3       "username": "kishan",
4       "email": "kishan@yopmail.com",
5       "bio": "A CSE student",
6       "image":
  "http://127.0.0.1:8000/media/profle_images/Screenshot_fro
  m_2024-10-04_19-31-59.png"
7   }
```


Make sure to pass Token (access_token) as the Auth(Bearer) to get the data

Test this endpoint (update the profile details)

PUT http://127.0.0.1:8000/api/account/profile/ Send

Body (2) Query Header (1) Auth (Bearer) Script Docs

Multipart Form

image	Choose Files angel-in-hell-19.jpg x	File	☑
bio	A CSE student	Text	☑

+ Add Item

200 OK 100 ms 96 B 8 days ago | 27-Jan-25 04:09:41 PM

Preview Header Request Tests Timeline

```
1 ▼ {
2       "bio": "A CSE student",
3       "image":
  "http://127.0.0.1:8000/media/profle_images/angel-i
  hell-19.jpg"
4   }
```

In Body , now we have to set Multipart Form instead of JSON

Refresh the token

For User module we have one router left! Which is refresh the token. Luckily, we don't need any serializers or views to handle this. DRF simple-jwt is helpful to handle this for us. We just need to implement the url.

```
from django.urls import path
from .views import (
   UserSignUp,
   ResendVerificationEmail,
   VerifyEmail,
   UserLogin,
   RetrieveUpdateProfile
)
from rest_framework_simplejwt.views import TokenRefreshView


urlpatterns = [
   path('signup/', UserSignUp.as_view(), name='signup'),
   path('verify-email/<str:token>/', VerifyEmail.as_view() , name='verify_email'),
   path('resend-verification/', ResendVerificationEmail.as_view(), name= 'resend_verification'),
   path('login/', UserLogin.as_view(), name='login'),
   path('profile/', RetrieveUpdateProfile.as_view(), name='profile'),
   path('token/refresh/', TokenRefreshView.as_view(), name='token_refresh')
]
```

Test the url (refresh the token)

POST http://127.0.0.1:8000/api/account/token/refresh/ Send

Body (JSON) Query Header (1) Auth Script Docs

JSON


```
{
    "refresh":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTczODgzNjQxMiwiaWF0IjoxNzM4NzUwMDEyLCJqdGkiOiIwZmQwZTRkMDA5ZWQ0MTUwYjlkYTVjZDY4NDI5YWMyMSIsInVzZXJfaWQiOjN9.dOD6Bc9HlirJWPFJbe_QaTMwIQDV7SKpCvATW_0uRfI"
}
```


200 OK 16 ms 241 B Just Now | 05-Feb-25 04:07:05 PM

Preview Header Request Tests Timeline


```
{
    "access":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90
oiYWNjZXNzIiwiZXhwIjoxNzM4NzUzNjI1LCJpYXQiOjE3Mzg3
IsImp0aSI6Ij1lOTkwMjMzNjAxMzQwMzE5OGJiNTU2MGU4ZDln
widXNlcl9pZCI6M30.B75NGm5ME2uO_djxXq-
ZZM2CSVKXh8wSSqhMAK8f2Fc"
}
```

Let's Design our Blog crud endpoints (Serializers)

```
# blog/serializers.py
from rest_framework import serializers
from .models import Blog

class BlogSerializer(serializers.ModelSerializer):
   class Meta:
       model = Blog
       fields = ['id', 'title', 'content', 'slug', 'author', 'created_at',
'updated_at']
       read_only_fields = ['author', 'created_at', 'updated_at']

class BlogCreateUpdateSerializer(serializers.ModelSerializer):
   class Meta:
       model = Blog
       fields = ['title', 'content']
```

Blog Views (imports)

```
# blog/views.py
from rest_framework import generics, permissions
from .models import Blog
from .serializers import BlogSerializer, BlogCreateUpdateSerializer
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework import status
from rest_framework.pagination import PageNumberPagination
from django.http import Http404
```

Blog Views (List)

```
# blog/views.py
class BlogPagination(PageNumberPagination):
   page_size = 5  # You can adjust this as needed
   page_size_query_param = 'page_size'
   max_page_size = 10

class BlogListView(generics.ListAPIView):
   queryset = Blog.objects.all().order_by('-created_at')
   serializer_class = BlogSerializer
   pagination_class = BlogPagination
```

Blog Views (Detail / Single get)

```
# blog/views.py
class BlogDetailView(generics.RetrieveAPIView):
   queryset = Blog.objects.all()
   serializer_class = BlogSerializer
   lookup_field = 'id'
```

Blog view (Create)

```
# blog/views.py
class BlogCreateView(generics.CreateAPIView):
   serializer_class = BlogCreateUpdateSerializer
   permission_classes = [IsAuthenticated]

   def perform_create(self, serializer):
       serializer.save(author=self.request.user)
```

Blog view (Update)

```
# blog/views.py
class BlogUpdateView(generics.UpdateAPIView):
   queryset = Blog.objects.all()
   serializer_class = BlogCreateUpdateSerializer
   permission_classes = [IsAuthenticated]
   lookup_field = 'id'

   def get_object(self):
       blog = super().get_object()
       if blog.author ≠ self.request.user:
           raise Http404
       return blog
```

Blog view (Delete)

```
# blog/views.py
class BlogDeleteView(generics.DestroyAPIView):
   queryset = Blog.objects.all()
   permission_classes = [IsAuthenticated]
   lookup_field = 'id'

   def get_object(self):
       blog = super().get_object()
       if blog.author ≠ self.request.user:
           raise Http404
       return blog
```

Blog view (My blogs)

```
# blog/views.py
class MyBlogsView(generics.ListAPIView):
   serializer_class = BlogSerializer
   permission_classes = [IsAuthenticated]
   pagination_class = BlogPagination

   def get_queryset(self):
       return Blog.objects.filter(author=self.request.user)
```

Blog URL

```
# blog/urls.py
from django.urls import path
from .views import (
    BlogListView,
    BlogDetailView,
    BlogCreateView,
    BlogUpdateView,
    BlogDeleteView,
    MyBlogsView
)
urlpatterns = [
    path('', BlogListView.as_view(), name='blog-list'),
    path('create', BlogCreateView.as_view(), name='blog-create'),
    path('<int:id>', BlogDetailView.as_view(), name='blog-detail'),
    path('update/<int:id>', BlogUpdateView.as_view(), name='blog-update'),
    path('delete/<int:id>', BlogDeleteView.as_view(), name='blog-delete'),
    path('my-blogs/', MyBlogsView.as_view(), name='my-blogs'),
]
```

BASE URL SETUP : for blog

```
# src/urls.py
from django.conf.urls.static import static

urlpatterns = [
   path('admin/', admin.site.urls),
   path('api/account/', include('account.urls')),
   path('api/blogs/', include('blog.urls')),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Let’s test (Create)

POST http://localhost:8000/api/blogs/create/ Send

Body (JSON) Query Header (1) Auth (Bearer) Script Docs

JSON

```
1 ▼ {
2       "title" : "Simple",
3       "content" : "Simple is better than complex"
4   }
```

201 Created 36 ms 60 B 10 days ago | 27-Jan-25 05:09:39

Preview Header Request Tests Timeline

```
1 ▼ {
2       "title": "Simple",
3       "content": "Simple is better than complex"
4   }
```

Let’s test (List Get)

GET http://127.0.0.1:8000/api/blogs Send

Body Query Header Auth (Bearer) Script Docs

No Body

200 OK 40 ms 252 B 10 days ago | 27-Jan-25 05:18:45 PM | http://...

Preview Header Request Tests Timeline


```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 2,
            "title": "Simple",
            "content": "Simple is better than complex",
            "slug": "simple_8070450532248443581",
            "author": 3,
            "created_at": "2025-01-27T11:09:38.998387Z",
            "updated_at": "2025-01-27T11:09:38.998427Z"
        }
    ]
}
```

Let’s test (Single Get)

GET http://127.0.0.1:8000/api/blogs/1 Send

200 OK 36 ms 183 B 10 days ago | 27-Jan-25 05:14:51 PM | http://...

Body Query Header Auth (Bearer) Script Docs

No Body

Preview Header Request Tests Timeline


```
{
    "id": 1,
    "title": "My Simple",
    "content": "Simple is better than complex",
    "slug": "simple",
    "author": 3,
    "created_at": "2025-01-27T11:05:59.986146Z",
    "updated_at": "2025-01-27T11:14:12.080717Z"
}
```

Let’s test (My Blogs)

GET http://127.0.0.1:8000/api/blogs/my-blogs/ Send

200 OK 8 ms 252 B 10 days ago | 27-Jan-25 05:20:44 PM | http://1...

Body Query Header Auth (Bearer) Script Docs

Bearer Token

Enabled	☑
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWN
Prefix	

Preview Header Request Tests Timeline


```
1  {
2      "count": 1,
3      "next": null,
4      "previous": null,
5      "results": [
6          {
7              "id": 2,
8              "title": "Simple",
9              "content": "Simple is better than complex",
10             "slug": "simple_8070450532248443581",
11             "author": 3,
12             "created_at": "2025-01-27T11:09:38.998387Z",
13             "updated_at": "2025-01-27T11:09:38.998427Z"
14         }
15     ]
16 }
```

Let’s test (Update blog)

PUT http://127.0.0.1:8000/api/blogs/update/1 Send

Body (JSON) Query Header (1) Auth (Bearer) Script Docs

JSON

```
1 ▼ {
2       "title" : "My Simple",
3       "content" : "Simple is better than complex"
4   }
```

200 OK 48 ms 63 B 10 days ago | 27-Jan-25 05:18:17 PM | http://1...

Preview Header Request Tests Timeline

```
1 ▼ {
2       "title": "My Simple",
3       "content": "Simple is better than complex"
4   }
```

Please pass, Auth(Bearer)

Let’s test (Delete blog)

DELETE http://127.0.0.1:8000/api/blogs/delete/1 Send

Body (JSON) Query Header (1) Auth (Bearer) Script Docs

Bearer Token

Enabled	☑
Token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWN
Prefix	

204 No Content 62 ms 10 days ago | 27-Jan-25 05:18:40 PM | http://1...

Preview Header Request Tests Timeline

1

Please pass, Auth(Bearer)