

# **Project Documentation**

## Contents

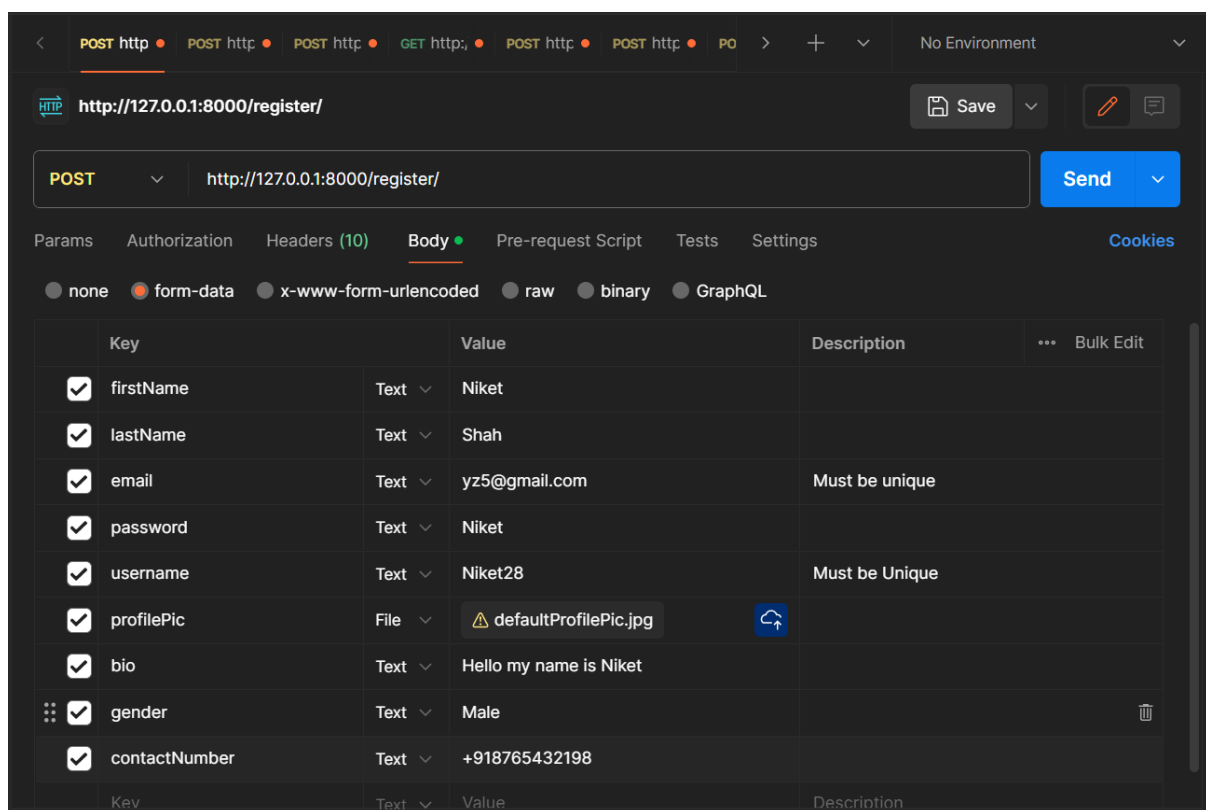
Registration API:-.....	2
Login Api:- .....	4
Update Profile API:-.....	5
Posts Api:- .....	8
Post Creation:-.....	8
Get List of All Posts:- .....	9
Like Post:- .....	10
Unlike Post:- .....	12
Connections Management:- .....	13
Request Stage:- .....	13
Request Handling:-.....	15
Recommendations:- .....	17

## Registration API:-

For registration the fields considered are:-

- User Id
- First Name
- Last Name
- Email
- Password
- Username
- Profile Pic
- Gender
- Contact Number

User Id is an auto created field and others are needed to be provided



Output:-

The screenshot shows a Postman interface for a POST request to `http://127.0.0.1:8000/register/`. The request body is set to form-data. The form fields are:

Key	Type	Value	Description
firstName	Text	Niket	
lastName	Text	Shah	
email	Text	yz5@gmail.com	Must be unique
password	Text	Niket	
username	Text	Niket28	Must be Unique

The response is a 200 OK status with a response time of 2.26 s and a body size of 327 B. The response body is a JSON object:

```
1 {
2   "success": true
3 }
```

If any field is not according to Format eg:- If Email is taken as niketgmail.com

The screenshot shows a Postman interface for a POST request to `http://127.0.0.1:8000/register/`. The request body is set to form-data. The form fields are:

Key	Type	Value	Description
firstName	Text	Niket	
lastName	Text	Shah	
email	Text	niketgmail.com	Must be unique
password	Text	Niket	

The response is a 400 Bad Request status with a response time of 12 ms and a body size of 391 B. The response body is a JSON object:

```
1 {
2   "success": false,
3   "error": {
4     "email": [
5       "Enter a valid email address."
6     ]
7   }
8 }
```

## Login Api:-

Login is done via email and password:-

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/login/`. The request body is set to `form-data` and contains the following parameters:

Key	Value	Description
email	yz5@gmail.com	
password	Niket	
password	Niket	

The response is shown in the `Body` tab, indicating a `200 OK` status with a response time of `2.29 s` and a size of `373 B`. The response body is a JSON object:

```
{  "success": true,  "auth_token": "5d903c7bb953505f2d607b5f64442694276e77dc8d48cb6c0c4ffcd7a8ebe67e"}
```

## If Authentication Fails

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/login/`. The request body is set to `form-data` and contains the following parameters:

Key	Value	Description
email	yz5@gmail.com	
password	Niket	
password	Niket123	

The response is shown in the `Body` tab, indicating a `400 Bad Request` status with a response time of `463 ms` and a size of `333 B`. The response body is a JSON object:

```
{  "success": false,  "error": "Invalid Credentials"}
```

## Update Profile API:-

Here whole object of User profile is sent with altered field values and then it is updated. The update function is in serializer which is used to update the instance:-

```
def update(self, instance, validated_data):
    print(instance.email)
    instance.firstName = validated_data['firstName']
    instance.lastName = validated_data['lastName']
    instance.username = validated_data['username']
    instance.profilePic = validated_data['profilePic']
    instance.bio = validated_data['bio']
    instance.gender = validated_data['gender']
    instance.contactNumber = validated_data['contactNumber']
    instance.email=validated_data["email"]

    password = validated_data['password']
    if password:
        instance.password=make_password(password)
    instance.save()
    return instance
```

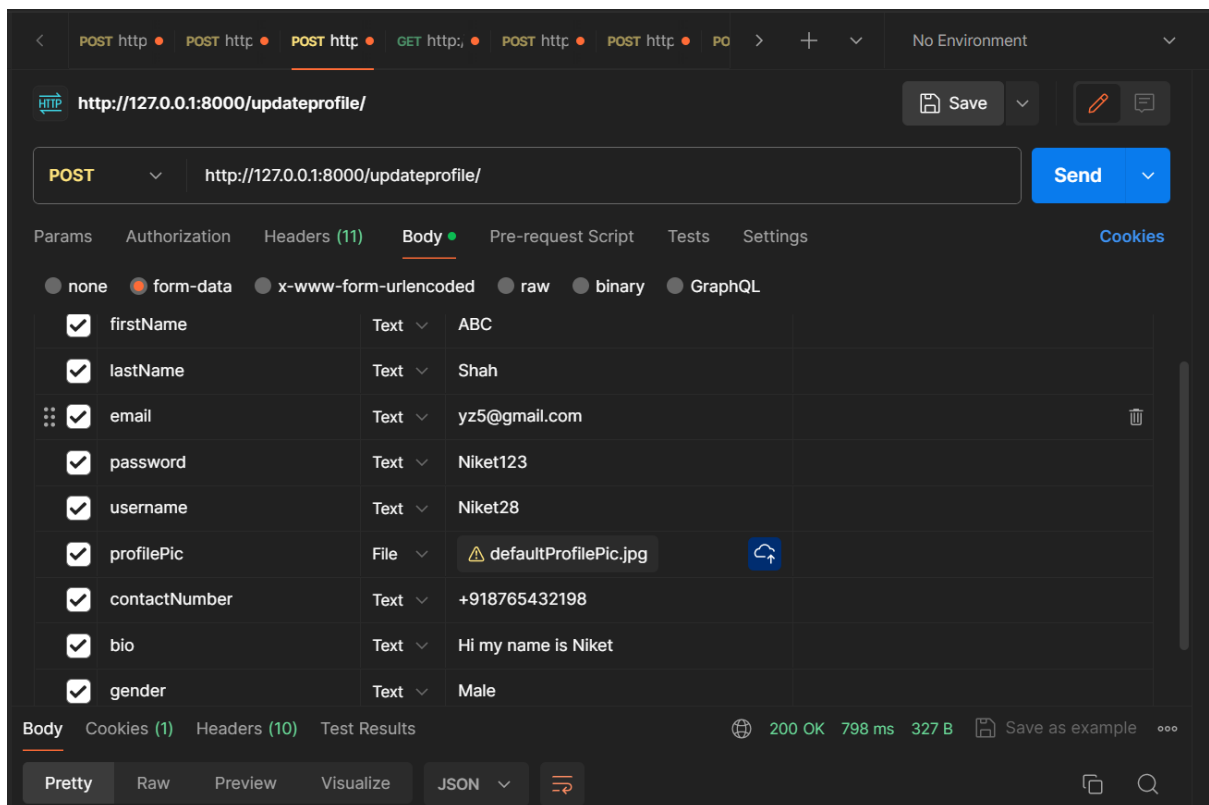
Eg:- If Password and First Name are to be changed for the newly registered user above

First the Token must be added in Header:-

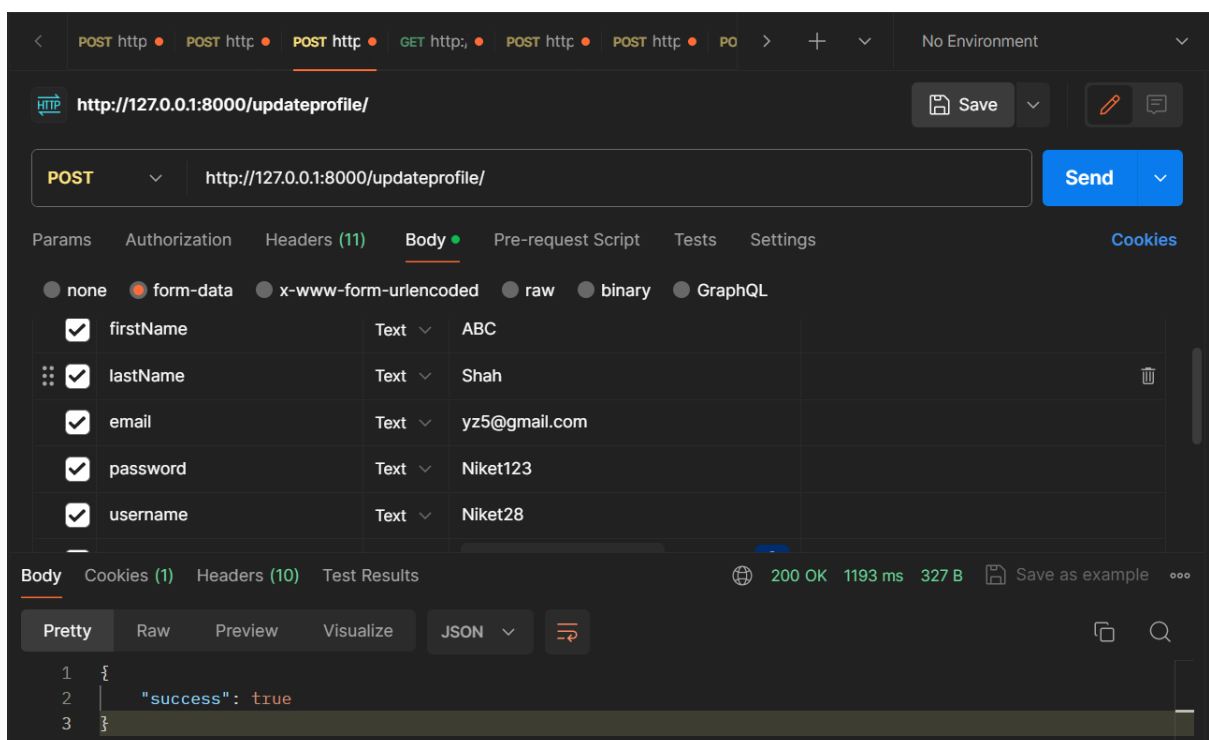
The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/updateprofile/`. The request is configured with the following headers:

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Token 9cfb8cd253b63ee64e497e67e12...	
Key	Value	Description

The interface also shows tabs for Params, Authorization, Headers (11), Body, Pre-request Script, Tests, Settings, and Cookies. The Response tab is visible at the bottom.



Output:-



As we test now the password has been changed :-

The screenshot shows the Postman interface for a POST request to `http://127.0.0.1:8000/login/`. The request body is set to `form-data` with the following parameters:

Key	Type	Value
email	Text	yz5@gmail.com
password	Text	Niket
password	Text	Niket123

The response is displayed in the 'Body' tab, showing a JSON object with the following structure:

```
1 {
2   "success": false,
3   "error": "Invalid Credentials"
4 }
```

The status bar indicates a `400 Bad Request` with a response time of `435 ms` and a size of `333 B`.

The screenshot shows the Postman interface for a POST request to `http://127.0.0.1:8000/login/`. The request body is set to `form-data` with the following parameters:

Key	Type	Value
email	Text	yz5@gmail.com
password	Text	Niket123
password	Text	Niket123

The response is displayed in the 'Body' tab, showing a JSON object with the following structure:

```
1 {
2   "success": true,
3   "auth_token": "3a59bb3c096299064cd5bd9b246272b725dedc462a317a584a986d9a10e68c33"
4 }
```

The status bar indicates a `200 OK` with a response time of `957 ms` and a size of `373 B`.

## Posts Api:-

- Post Creation and to get list of posts both are included in same api. i.e Through POST request you can create a post by sending Token and relevant data(considered only image/video file and caption for now) and by sending GET request just with Token of user you can get list of posts of that user.

## Post Creation:-

The screenshot shows the Postman interface for a POST request to `http://127.0.0.1:8000/post/`. The **Headers** tab is selected, showing 11 headers. The **Authorization** header is checked and set to `Token 8d613f0e3fc7fa71422ca4d607e4...`.

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Token 8d613f0e3fc7fa71422ca4d607e4...	
key	Value	Description

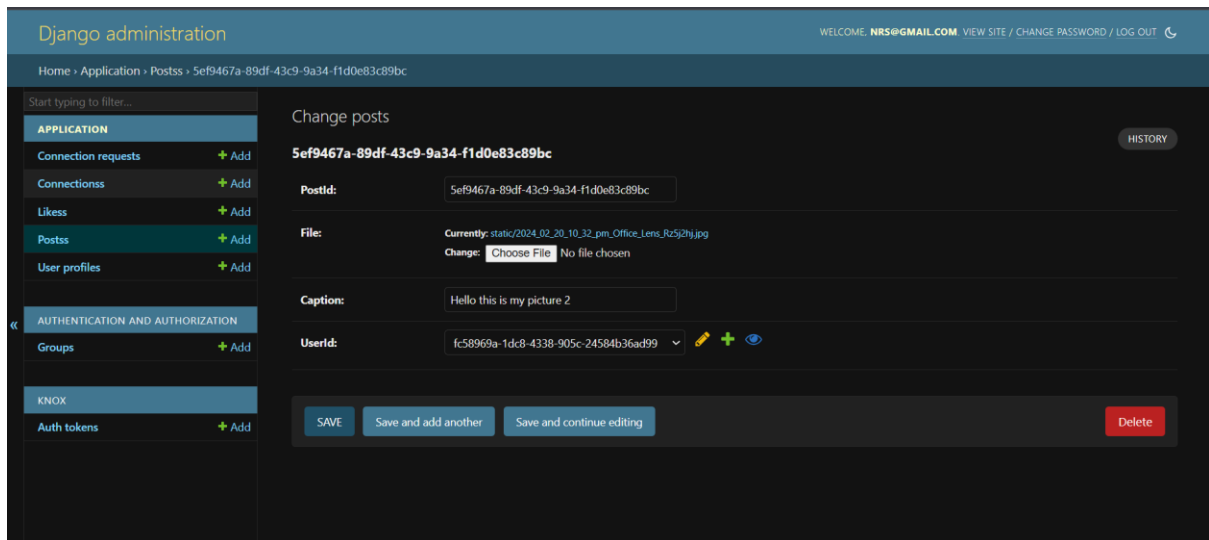
The screenshot shows the Postman interface for the same POST request, but with the **Body** tab selected. The body type is `form-data`. Two fields are defined:

Key	Value	Description
<input checked="" type="checkbox"/> file	File <code>2024_02_20 10_32 pm Offi...</code>	
<input checked="" type="checkbox"/> caption	Text <code>Hello this is my picture 2</code>	
Key	Text	Value

The response is shown in the **Body** tab, displaying a JSON object:

```
1 {
2   "success": true
3 }
```

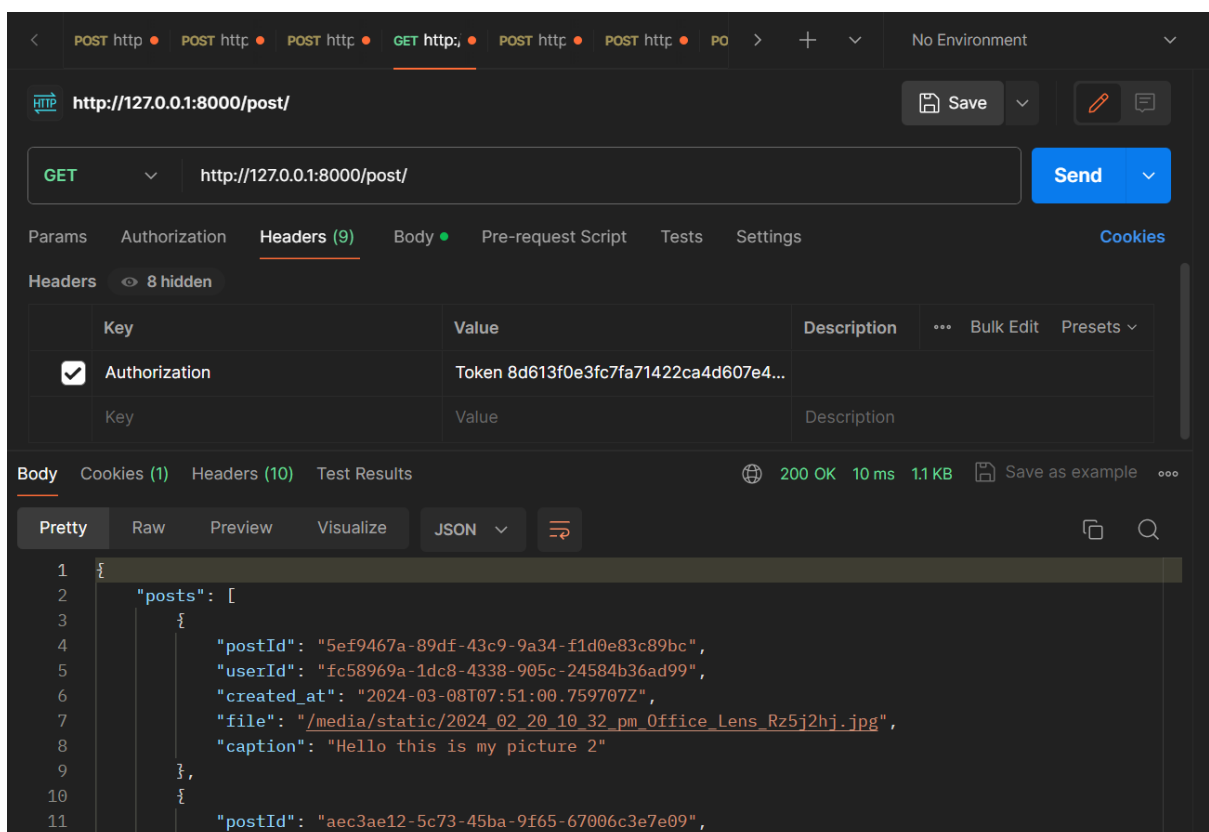




Get List of All Posts:-

Added 2 more posts and then will retrieve data

Just by sending Token we are able to retrieve all posts of the user

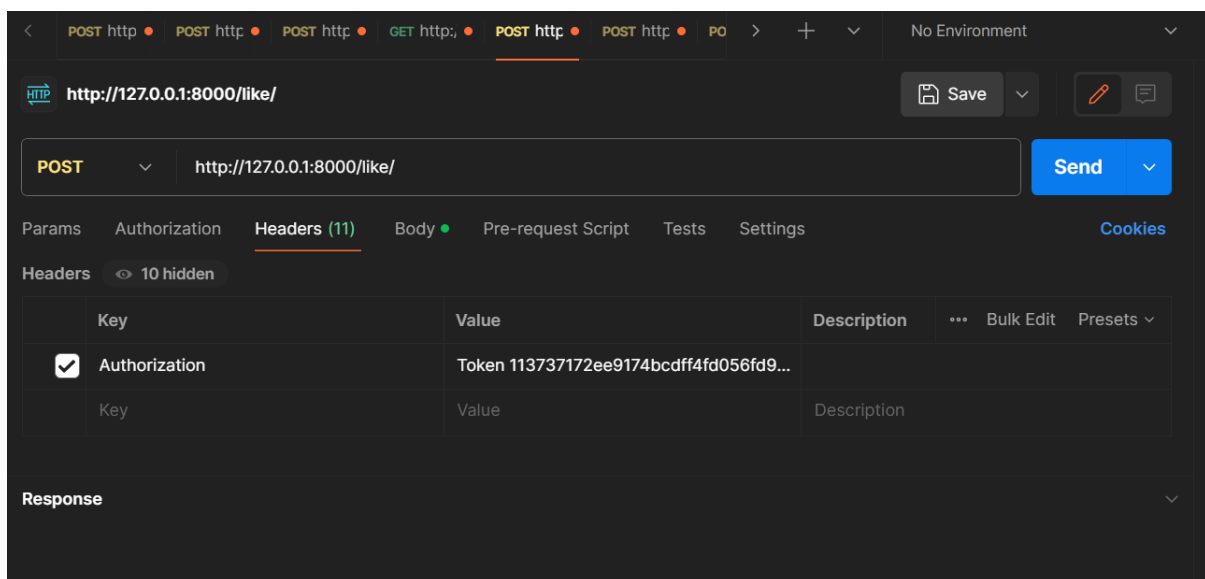


## Like Post:-

- Here If the user has liked once then if he/she like again it wont give any error but will not save in database again

```
@api_view(["POST"])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def likepost(request):
    try:
        liked_obj=Likes.objects.get(userId=str(request.user))
    except Likes.DoesNotExist:
        post_id=request.POST["post_id"]
        like_post_serializer=LikeSerializer(data={'postId':post_id,'userId':str(request.user)})
        if like_post_serializer.is_valid():
            like_post_serializer.save()
            return JsonResponse({'success':True},status=200)
        else:
            return JsonResponse({'success':False,'error':like_post_serializer.errors},status=400)
    return JsonResponse({'success':True},status=200)
```

Here first the token of user who has liked is taken:



The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/like/`. The request is configured with the following headers:

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Token 113737172ee9174bcdff4fd056fd9...	
Key	Value	Description

The interface also shows tabs for Params, Authorization, Headers (11), Body, Pre-request Script, Tests, and Settings. The Response tab is currently collapsed.

Now for data we need post id of posts user wants to like:-

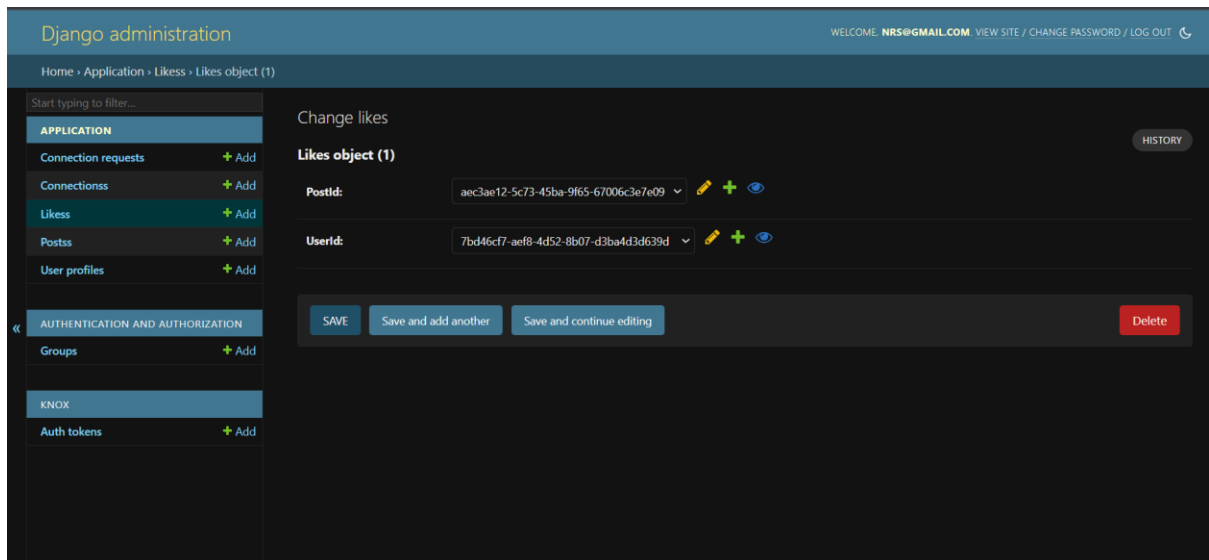
The screenshot shows the Postman interface with a POST request configured to `http://127.0.0.1:8000/like/`. The request body is set to `form-data`. A single key-value pair is defined in the body tab:

Key	Value	Description
post_id	aec3ae12-5c73-45ba-9f65-67006c3e7e...	

The response section is currently empty, displaying a cartoon character.

The screenshot shows the same POST request in Postman, but now the response is visible. The status is `200 OK` with a response time of `1207 ms` and a size of `327 B`. The response body is displayed in JSON format:

```
1 {
2   "success": true
3 }
```

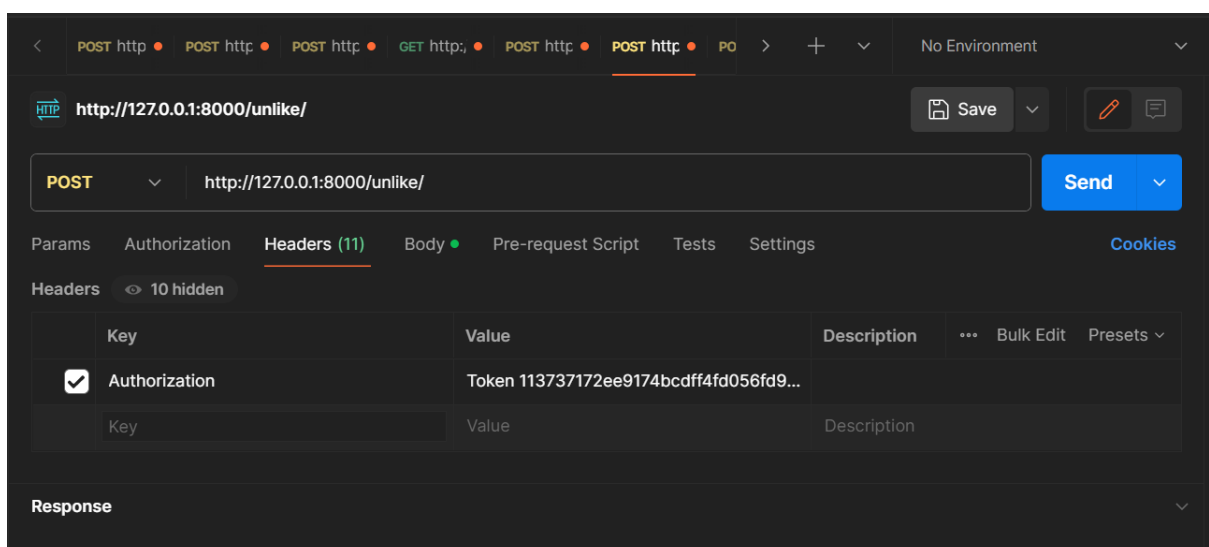


## Unlike Post:-

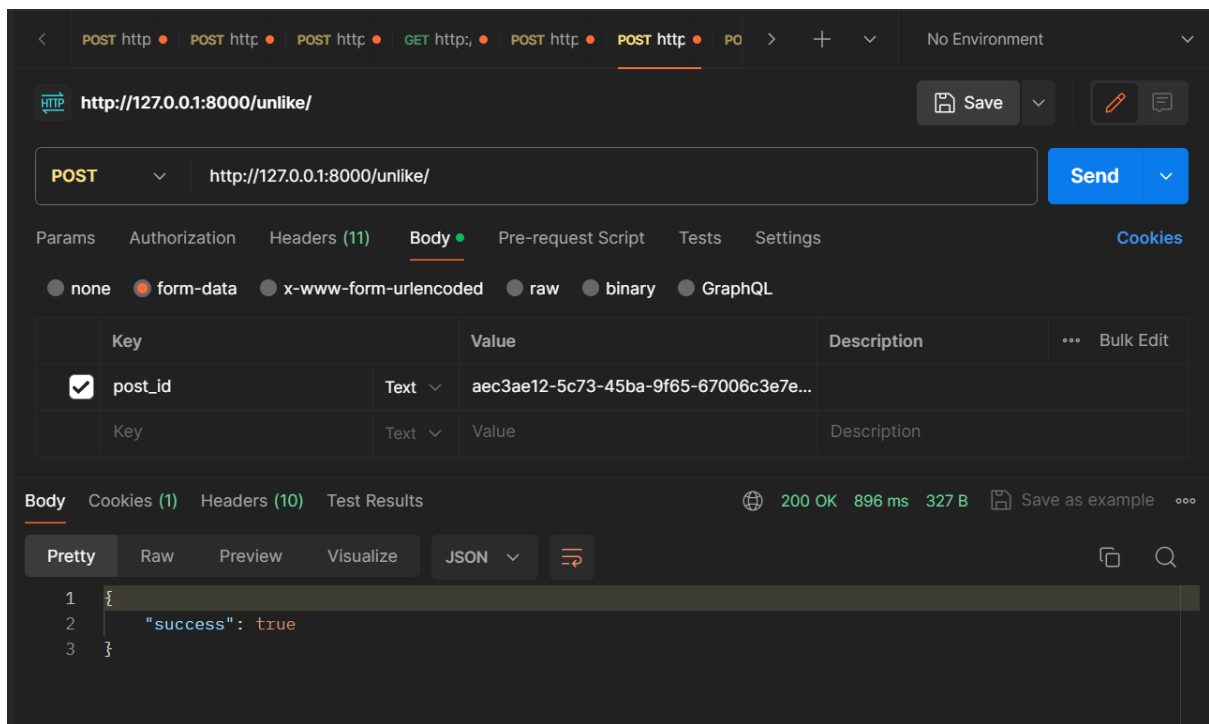
- When you unlike post here it removes the data from database and if he has not liked it will not give any error but will not perform any action as he/she hasn't liked the post

```
@api_view(["POST"])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def unlike(request):
    try:
        liked_obj=Likes.objects.get(userId=str(request.user),postId=request.POST['post_id'])
        liked_obj.delete()
        return JsonResponse({'success':True},status=200)
    except Likes.DoesNotExist:
        return JsonResponse({'success':True},status=200)
```

Here First we need to taken token of user who ants to unlike:-



After that for data we need to post id which helps in mapping which post is to be unliked:-

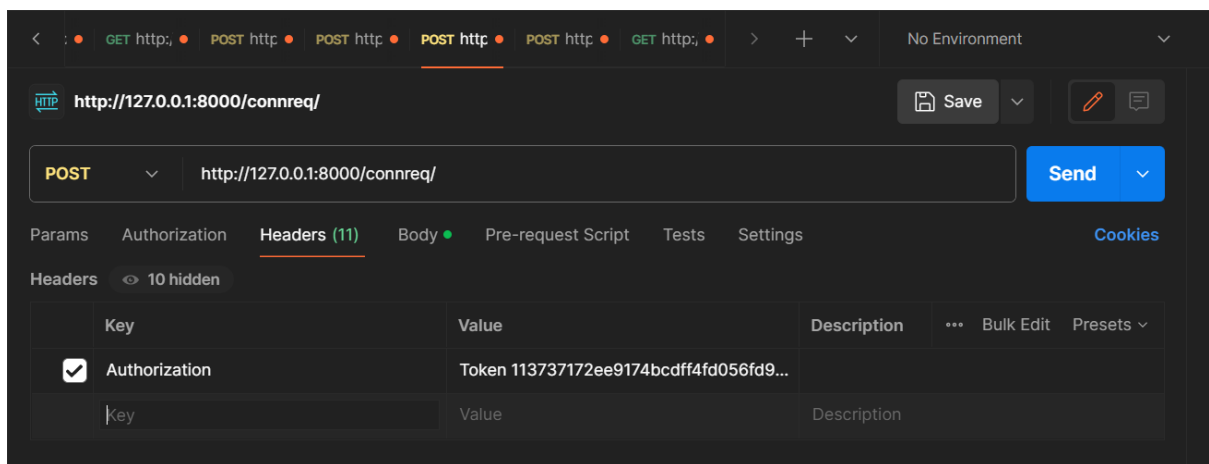


## Connections Management:-

- Here there are 2 stages First is user adds a connection request and after that the other user can accept or reject the request. If he /she accepts requests then the user is added to connections.

## Request Stage:-

Here first the token of the user sending request is taken:-



And for data , username of the user whom request is to be sent:

The screenshot shows a REST client interface with the following details:

- URL: `http://127.0.0.1:8000/connreq/`
- Method: `POST`
- Body (form-data):

Key	Value	Description
<input checked="" type="checkbox"/> username	Niket28	
- Response (JSON):

```
{  "success": true}
```
- Status: `200 OK`, Time: `491 ms`, Size: `327 B`

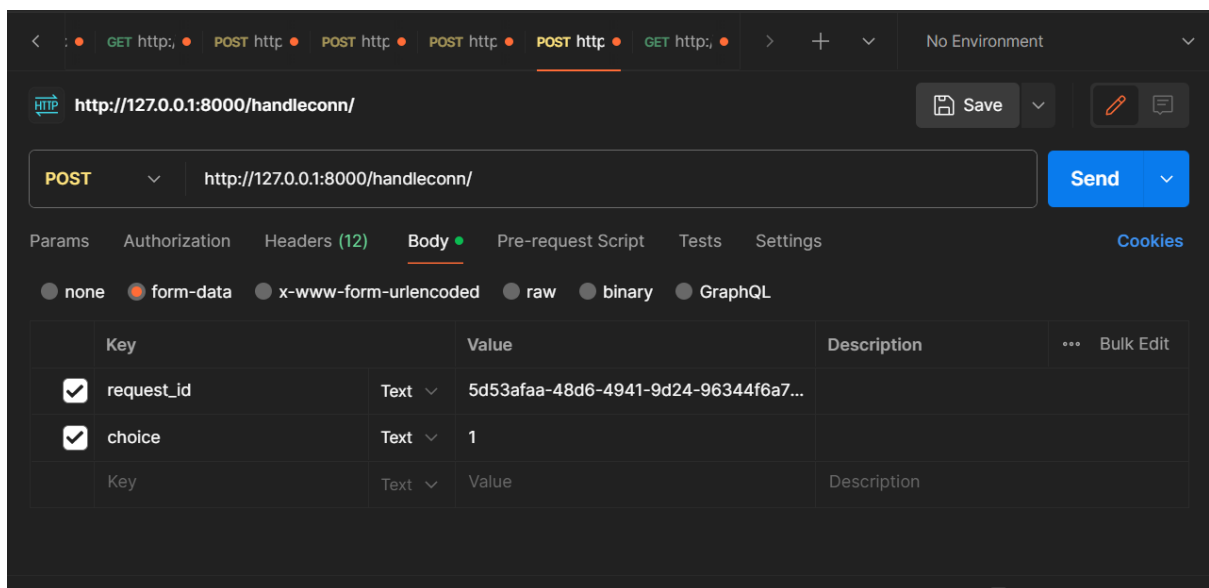
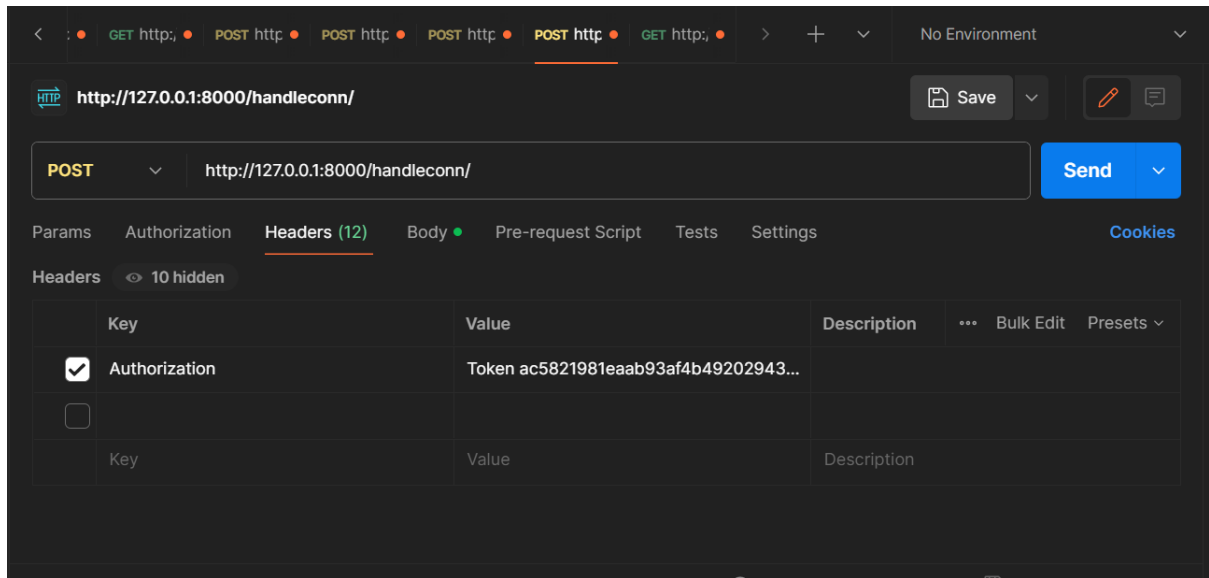
Code:-

```
@api_view(["POST"])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def connectionrequest(request):
    destination_username=request.POST['username']
    print(destination_username)
    try:
        destination_id=UserProfile.objects.get(username=destination_username).userId
    except UserProfile.DoesNotExist:
        return JsonResponse({'success':False,'error':'User with given username Does not exist!'},status=400)
    try:
        req_data=ConnectionRequest.objects.get(source=str(request.user),destination=destination_id)
    except ConnectionRequest.DoesNotExist:
        try:
            conn_exits=Connections.objects.get(user2=str(request.user),user1=str(destination_id))
        except Connections.DoesNotExist:
            try:
                rev_conn=Connections.objects.get(user1=str(request.user),user2=str(destination_id))
            except Connections.DoesNotExist:
                conn_req={'source':str(request.user),'destination':destination_id}
                conn_req_serializer=ConnectionRequestSerializer(data=conn_req)
                if conn_req_serializer.is_valid():
                    conn_req_serializer.save()
                    return JsonResponse({'success':True},status=200)
                else:
                    return JsonResponse({'success':False,'error':conn_req_serializer.errors},status=400)
            return JsonResponse({'success':True},status=200)
    return JsonResponse({'success':True},status=200)
```

## Request Handling:-

Here User can accept or decline .If user accepts the request then the record is added to Connection else it is just discarded.

Here we take the request Id and the Token of user whom request was sent:-



Here if choice is 1 it means user has accepted and if it is 0 it means user has rejected the request

The screenshot shows a REST client interface with the following details:

- URL:** `http://127.0.0.1:8000/handleconn/`
- Method:** `POST`
- Body:** `request_id` (5d53afaa-48d6-4941-9d24-96344f6a7...) and `choice` (1).
- Response:** `200 OK` with a body of `{ "success": true }`.

Code:-

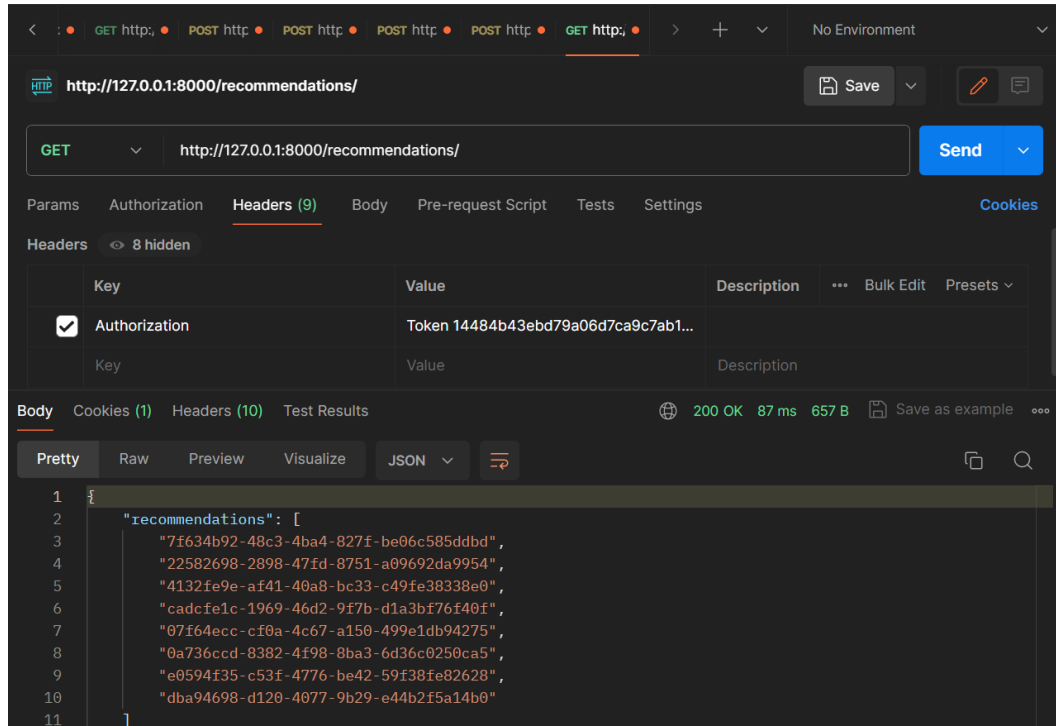
```
@api_view(["POST"])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def handle_connection_request(request):
    request_id = request.POST['request_id']
    choice = request.POST['choice']
    # Here choice represents whether user accepts or rejects the request so it would be either 0 or 1
    try:
        request_data=ConnectionRequest.objects.get(reqId=request_id)
        print(choice)
        if choice == "1":
            if str(request_data.destination)==str(request.user):
                conn_data={'user2':str(request_data.source),'user1':str(request.user)}
                conn_serializer=ConnectionSerializer(data=conn_data)
                if conn_serializer.is_valid():
                    conn_serializer.save()
                    request_data.delete()
                    return JsonResponse({'success':True},status=200)
            else:
                return JsonResponse({'success':False,'error':conn_serializer.errors},status=400)
        else:
            return JsonResponse({'success':False,'error':'Authentication Failed'},status=400)
    except ConnectionRequest.DoesNotExist:
        request_data.delete()
        return JsonResponse({'success':True},status=200)
    except ConnectionRequest.DoesNotExist:
        return JsonResponse({'success':False,'error':'Request Does Not Exist'},status=400)
```



## Recommendations:-

- Here we first take connections of the user and we check their connections who are not connected to user and recommend them to the user

Here we just take the token of user:-



We get the ids of the recommended users for connection.

Code:-

```
class Recommendations(APIView):
    permission_classes=[IsAuthenticated]
    authentication_classes=[TokenAuthentication]
    def get(self,request):
        connections = Connections.objects.filter(
            Q(user1=str(request.user)) | Q(user2=str(request.user))
        )

        user_connections = set()
        for connection in connections:
            user_connections.add(str(connection.user2))
            if str(connection.user1) == str(request.user)
            else str(connection.user1)
        )

        recommendation=set()
        for connection in user_connections:
            followers_connection = Connections.objects.filter(
                Q(user1=str(connection)) | Q(user2=str(connection))
            )

            followers_of_connection=set()
            for follower_of_connection in followers_connection:
                followers_of_connection.add(str(follower_of_connection.user2))
                if str(follower_of_connection.user1) == str(connection)
                else str(follower_of_connection.user1)
            )

            required_connections=followers_of_connection - user_connections
            required_connections.discard(str(request.user))

            recommendation=recommendation.union(required_connections)
        return JsonResponse({'recommendations':list(recommendation)},status=200)
```

Code Explanation:- Here we first take connections of user for whom recommendations are to be found. Then we find connections of the users who are not connected to user and add them to recommendation list and carry on . To avoid duplicates, set data structure is used and set union operator is used to merge the unique ids to recommendation.