

Problem Statement:

<https://docs.google.com/document/d/1F90KvSqxxPzIRyeX7kD3wULrT0xNJMLdigGVHq4KMxk/>

Installation

Download this repository and change into directory of project. Then use pip to install all required packages.

pip install -r requirements.txt

Files and Directory Structure

django_rest_kaushik_jadhav: Root folder of project

|---- cms_api : This folder contains our main api

|---- django_rest_kaushik_jadhav: This folder contains essential settings and configuration files of project.

Creating Admin via seeding

To create an Admin user via seeding, run the following command:

python manage.py createsuperuser

Note that all fields are validated even during admin creation and would throw error if rules mentioned in the problem statement are not followed

```
(base) E:\Kaushik\Projects\Github\test_env\django_rest_kaushik_jadhav>python manage.py createsuperuser
Email: admin@gmail.com
Name: Admin
Phone: 987
Error: Phone no. has to be exactly 10 characters
Phone:
```

Follow all validations for successful admin creation

```
(base) E:\Kaushik\Projects\Github\test_env\django_rest_kaushik_jadhav>python manage.py createsuperuser
Email: admin@gmail.com
Name: Admin
Phone: 987
Error: Phone no. has to be exactly 10 characters
Phone: 8652677172
Pincode: 400104
Password:
Password (again):
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Launching the API

Start the server on localhost by running

python manage.py runserver

Registering New User

Make a POST request to `http://domain_name/api/register/` to register a user. By default, domain_name is **127.0.0.1:8000**. All fields are validated. Invalid fields will throw respective errors as shown below

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/api/register/`. The request body is a JSON object: `{ "email": "user1@g.com", "name": "", "password": "password", "phone": "123", "pincode": "123" }`. The response status is **400 Bad Request**. The response body contains validation errors for the 'name', 'password', and 'phone' fields.

```
1 {
2   "email": "user1@g.com",
3   "name": "",
4   "password": "password",
5   "phone": "123",
6   "pincode": "123",
7 }
```

body Cookies Headers (9) Test Results Status: 400 Bad Request Time:

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": [
3     "This field may not be blank."
4   ],
5   "password": [
6     "Password must contain at least one uppercase and lowercase character and at least 8 characters."
7   ],
8   "phone": [
9     "Phone no. has to be exactly 10 characters"
10  ]
11 }
```

Activ Go to !

The response to a valid request will contain the user data and a token.

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/api/register/`. The request body is a JSON object: `{ "email": "user1@g.com", "name": "User1 Name", "password": "User1Pass", "phone": "9876543210", "pincode": "400104" }`. The response status is **200 OK**. The response body contains user data and a token.

```
1 {
2   "email": "user1@g.com",
3   "name": "User1 Name",
4   "password": "User1Pass",
5   "phone": "9876543210",
6   "pincode": "400104",
7 }
8 {
9   "phone": "9876543210",
10  "pincode": "400104",
11  "address": "",
12  "state": "",
13  "city": "",
14  "country": ""
15 },
16 {
17   "token": "f556b4745885c20d382c2ea9af6254a1cfb594dd"
18 }
```

Body Cookies Headers (9) Test Results Status: 200 OK Time: 1144 ms Si

Pretty Raw Preview Visualize JSON

Login Existing Users

Existing Users can Log In to get a token. Tokens can be used to access protected routes.

To login, make a POST request to `http://domain_name/api/login`

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8000/api/login/`. The 'Body' tab is selected, showing a JSON payload: `{ "username": "user1@g.com", "password": "User1Pass" }`. Below the request, the 'Body' tab of the response is shown, displaying a JSON token: `{ "token": "f556b4745885c20d382c2ea9af6254a1cfb594dd" }`. The status bar at the bottom right indicates a successful response with a status of 401.

```
POST http://127.0.0.1:8000/api/login/

Params  Authorization  Headers (9)  Body  Pre-request Script  Tests  Settings

● none  ● form-data  ● x-www-form-urlencoded  ● raw  ● binary  ● GraphQL  JSON

1 {
2   "username": "user1@g.com",
3   "password": "User1Pass"
4 }
```

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON ⌵

```
1 {
2   "token": "f556b4745885c20d382c2ea9af6254a1cfb594dd"
3 }
```

Viewing & Posting Content

Users can View and Create contents on the route `http://domain_name/api/content/`

This is a protected route and any user who is not logged in cannot access it.

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/api/content/`. The 'Body' tab is selected, showing an empty response. The status bar at the bottom right indicates a failed response with a status of 401.

```
GET http://127.0.0.1:8000/api/content/

Params  Authorization  Headers (7)  Body  Pre-request Script  Tests  Settings

● none  ● form-data  ● x-www-form-urlencoded  ● raw  ● binary  ● GraphQL  JSON  ⌵

1 |
```

Body Cookies Headers (10) Test Results Status: 401 Ur

Pretty Raw Preview Visualize JSON ⌵

```
1 {
2   "detail": "Authentication credentials were not provided."
3 }
```

As mentioned, users are of two roles: Admin and Author. Authors are regular users who can View, Create, Update and Delete only their own content. Whereas, Admins can View, Create, Update and Delete content of multiple Authors.

Before making any request, user needs to be logged in. To do this, the token of the user needs to be specified in the HTTP headers in the following format:

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Settings
Headers 7 hidden						
KEY		VALUE				
<input checked="" type="checkbox"/>	Authorization	Token 5fb4ed4b96e505bf2ef71d6d4604bd63a0ada2fd				
	Key	Value				

Viewing Content

View content by making a GET request to `http://domain_name/api/content/`

Regular users can view only their own content


GET ▼ http://127.0.0.1:8000/api/content

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

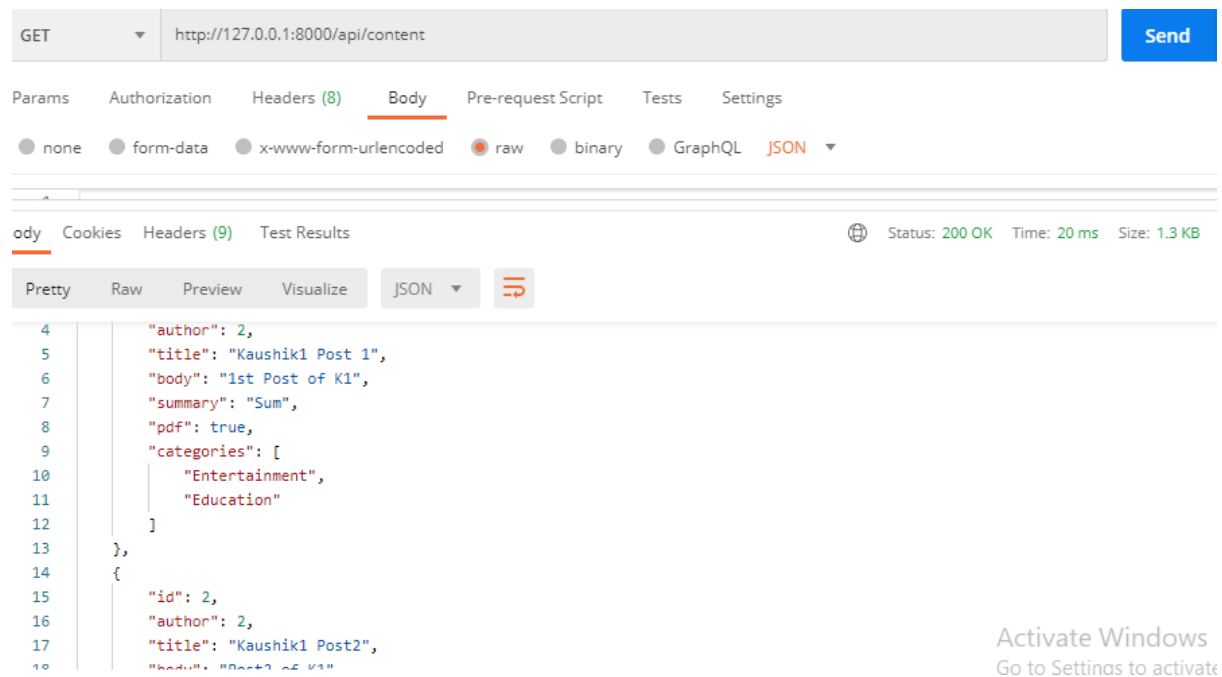
1

ody Cookies Headers (9) Test Results 🌐 Status: 200 OK

Pretty Raw Preview Visualize **JSON** ▼ 

```
5   "title": "User1 Post1",
6   "body": "Post1 Body of User1",
7   "summary": "Summary of U1 P1",
8   "pdf": true,
9   "categories": [
10    "Entertainment",
11    "Sports"
12  ]
13 }
```

Admins can view the contents of all users



GET ▼ http://127.0.0.1:8000/api/content Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

Body Cookies Headers (9) Test Results 🌐 Status: 200 OK Time: 20 ms Size: 1.3 KB

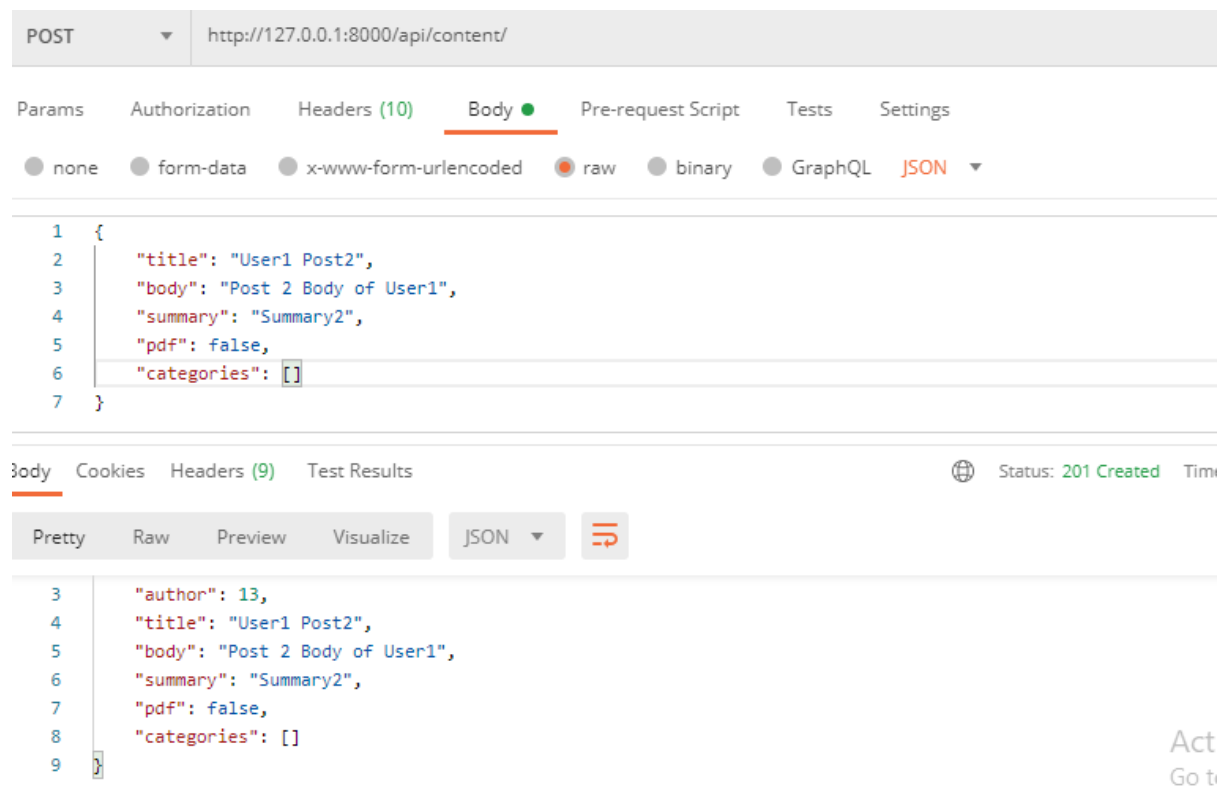
Pretty Raw Preview Visualize JSON ▼ ↻

```
4   "author": 2,
5   "title": "Kaushik1 Post 1",
6   "body": "1st Post of K1",
7   "summary": "Sum",
8   "pdf": true,
9   "categories": [
10    "Entertainment",
11    "Education"
12  ],
13 },
14 {
15   "id": 2,
16   "author": 2,
17   "title": "Kaushik1 Post2",
18   "body": "Post2 of K1"
```

Activate Windows
Go to Settings to activate

Adding New Content

New posts can be added by making a POST request to `http://domain_name/api/content/`



POST ▼ http://127.0.0.1:8000/api/content/

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

```
1  {
2    "title": "User1 Post2",
3    "body": "Post 2 Body of User1",
4    "summary": "Summary2",
5    "pdf": false,
6    "categories": []
7  }
```

Body Cookies Headers (9) Test Results 🌐 Status: 201 Created Time: 10 ms

Pretty Raw Preview Visualize JSON ▼ ↻

```
3   "author": 13,
4   "title": "User1 Post2",
5   "body": "Post 2 Body of User1",
6   "summary": "Summary2",
7   "pdf": false,
8   "categories": []
9 }
```

Act
Go to

Updating Content

Contents can be updated by making a PUT request to `http://domain_name/api/content/<content_id>/`

Authors can update only contents created by them. Admins can update contents created by all Authors.

The screenshot shows a REST client interface with a PUT request to `http://127.0.0.1:8000/api/content/11/`. The 'Body' tab is selected, showing a JSON payload: `{ "summary": "Summary2", "pdf": false, "categories": ["Entertainment", "Arts"] }`. The status bar at the bottom indicates a successful response with status 200 OK.

```
PUT http://127.0.0.1:8000/api/content/11/

Params Authorization Headers (10) Body Pre-request Script Tests Settings
● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼

5 body : Post 2 body of user1 ,
6 "summary": "Summary2",
7 "pdf": false,
8 "categories": [
9   "Entertainment",
10  "Arts"
11 ]
12 }
```

Body Cookies Headers (9) Test Results Status: 200 OK Time

Pretty Raw Preview Visualize JSON ↗

```
6 "summary": "Summary2",
7 "pdf": false,
8 "categories": [
9   "Entertainment",
10  "Arts"
11 ]
12 }
```

Act Go

Deleting Content

Contents can be deleted by making a DELETE request to `http://domain_name/api/content/<content_id>/`

Authors can delete only contents created by them. Admins can delete contents created by all Authors.

The screenshot shows a REST client interface with a DELETE request to `http://127.0.0.1:8000/api/content/9/`. The 'Body' tab is selected, and the status bar at the bottom indicates a successful response with status 204 No Content.

```
DELETE http://127.0.0.1:8000/api/content/9/

Params Authorization Headers (8) Body Pre-request Script Tests Settings
● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼

1
```

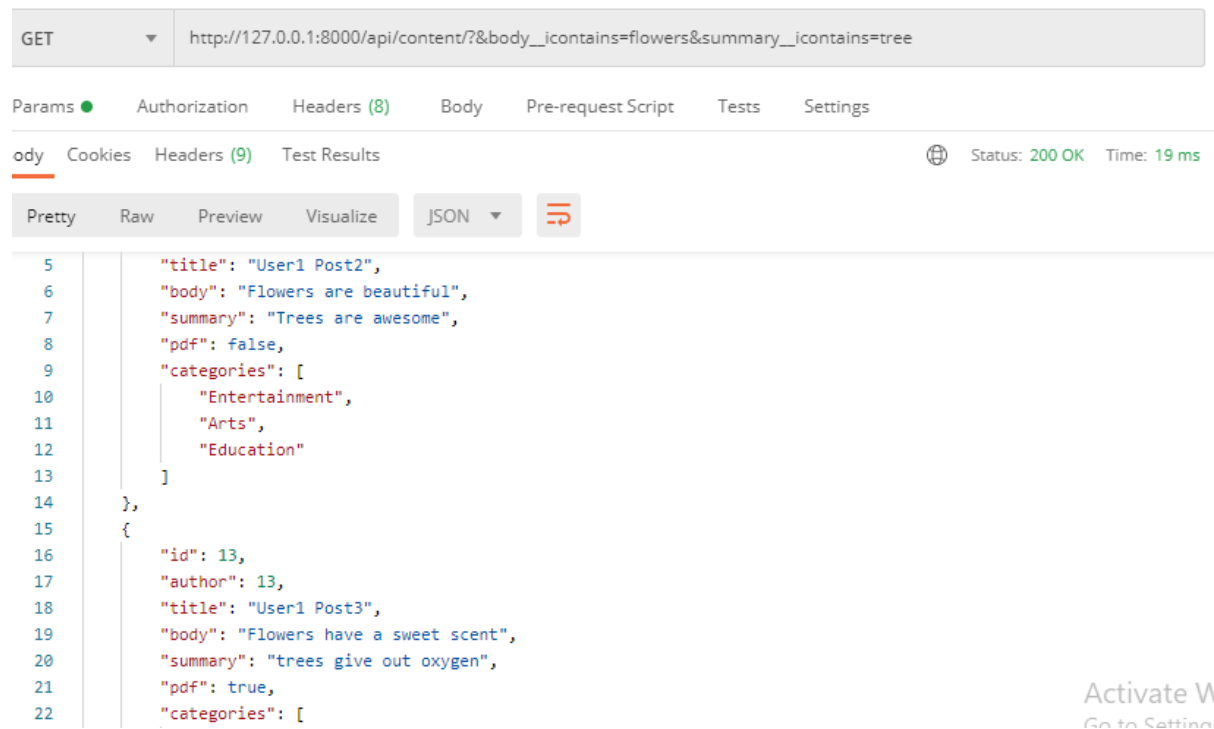
Body Cookies Headers (8) Test Results Status: 204 No Content

Searching and Filtering

Users can search content by matching terms in title, body, summary and categories by making a request to `/api/content/?` followed by your query in the form of `&<field_name>__icontains=<term>`

Example:

`http://127.0.0.1:8000/api/content/?&body__icontains=flowers&summary__icontains=tree`



The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8000/api/content/?&body__icontains=flowers&summary__icontains=tree`. The response is a JSON array of two objects, displayed in 'Pretty' format. The first object represents a post with ID 13, and the second object represents another post with ID 13. The response status is 200 OK and the time taken is 19 ms.

```
5     "title": "User1 Post2",
6     "body": "Flowers are beautiful",
7     "summary": "Trees are awesome",
8     "pdf": false,
9     "categories": [
10      "Entertainment",
11      "Arts",
12      "Education"
13    ]
14  },
15  {
16    "id": 13,
17    "author": 13,
18    "title": "User1 Post3",
19    "body": "Flowers have a sweet scent",
20    "summary": "trees give out oxygen",
21    "pdf": true,
22    "categories": [
```

Activate W
Go to Settings