

DOCUMENTATION

API Rate Limiter

Algorithm:

1. An access token is mapped to a particular user data upon authentication.
2. A HTTP POST request initiated by the user is processed.
3. Checks are made on whether the user is new or already linked with sever. If the user is new, it is added to the database.
4. A counter representing number of API hits is initialised. The program checks for which of the API endpoint, the requests are generated.
5. Accordingly, it assigns a default value of requests to the users.
6. The algorithm then checks the difference of the time window in which the user is making the requests.
7. If the time difference is less than the request time allowed to the user, the user can carry on the process of making requests.
8. The user request limit is updated in a counter which is maintained for every user in the database.
9. Once the user reaches request threshold, user has to wait for the next time window to continue sending requests.
10. If the rate limit is exceeded, a HTTP response of 'rate exceeded' is received by the client .

Illustration for a default rate limit:

1. Authenticating a user named 'Test1'.

Username	<input type="text" value="Test1"/>
Email	<input type="text" value="test1@gmail.com"/>
Password1	<input type="password" value="test1@1234"/>
Password2	<input type="password" value="test1@1234"/>
<input type="button" value="POST"/>	

2. This results in generation of a **token**.

Key	<input type="text" value="b15f02511e7a3432bb8da7180f8824e89e72e138"/>
<input type="button" value="POST"/>	

3. Simulating a **POST** request for 'dev' API endpoint on **Postman**.

POST

http://localhost:8000/api/dev

Send

Save

Params Authorization Headers (1) Body Pre-request Script Tests Settings Cookies Code Comments (0)

Headers (1)

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Token b15f02511e7a3432bb8da7180f8824e89e72e138				
	Key	Value	Description			

Response

4. JSON response upon 4 requests.

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/api/dev`. The **Headers** tab is active, showing an **Authorization** header with a token. The **Body** tab is also active, displaying a JSON response in the **Pretty** view. The response status is **200 OK** with a time of **35ms** and a size of **265 B**.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Authorization	Token b15f02511e7a3432bb8da7180f8824e89e72e138	
Key	Value	Description

```
1 {
2   "success": true,
3   "message": "Success",
4   "count": 4
5 }
```

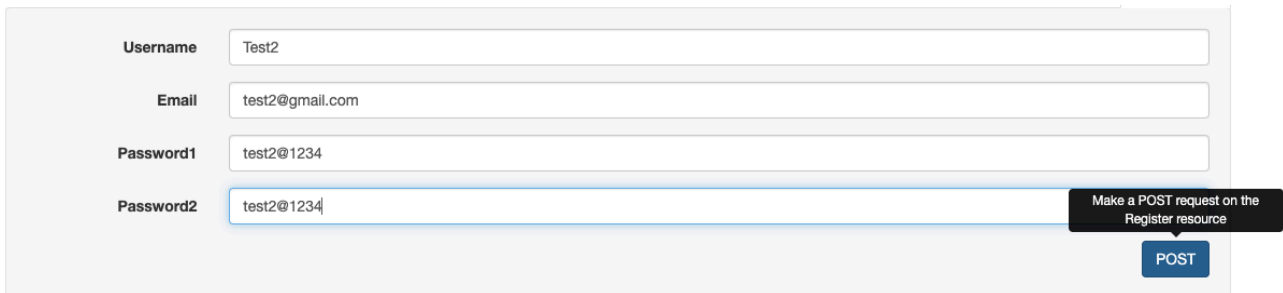
5. Upon exceeding the rate limit, the following error will be returned as response.

The screenshot shows the same REST client interface as before, but the response body now indicates a rate limit error. The response status is **200 OK** with a time of **14ms** and a size of **262 B**.

```
1 {
2   "success": false,
3   "message": "Rate exceeded"
4 }
```

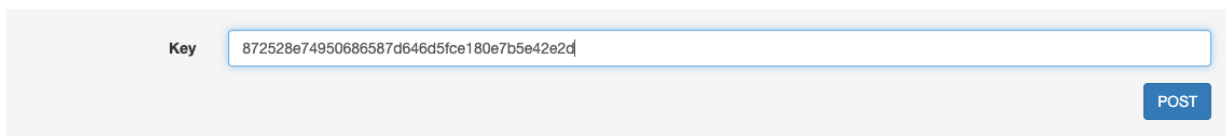
Illustration for an API+User combination rate limit:

1. Authenticating a user named 'Test2'.



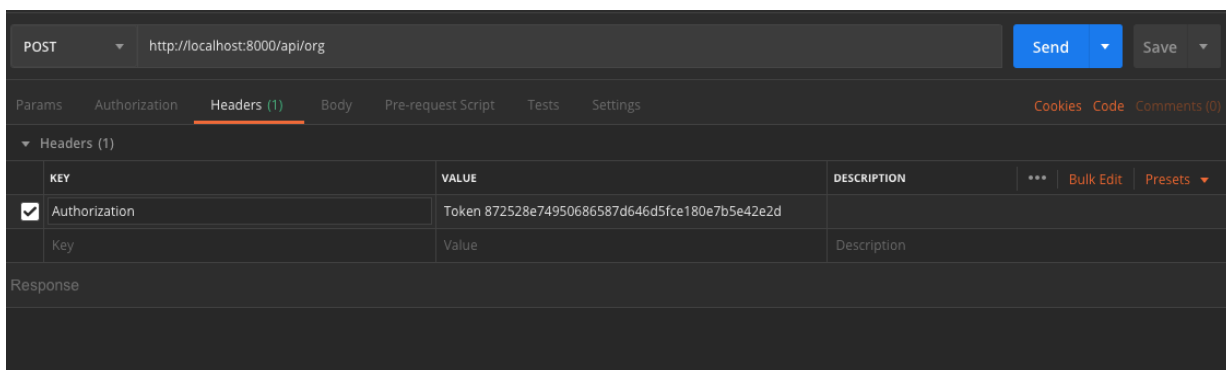
A Postman authentication form for a user named 'Test2'. The form includes fields for Username, Email, Password1, and Password2. The Username field is filled with 'Test2', Email with 'test2@gmail.com', Password1 with 'test2@1234', and Password2 with 'test2@1234'. A tooltip on the right says 'Make a POST request on the Register resource'. A 'POST' button is at the bottom right.

2. This results in generation of a **token**.



A Postman Key field showing a generated token: '872528e74950686587d646d5fce180e7b5e42e2d'. A 'POST' button is at the bottom right.

3. Simulating a **POST** request for 'org' API endpoint on **Postman**.



A screenshot of the Postman interface. The request method is 'POST' and the URL is 'http://localhost:8000/api/org'. The 'Headers' tab is selected, showing a table with one header: 'Authorization' with value 'Token 872528e74950686587d646d5fce180e7b5e42e2d'. The 'Response' section is empty.

KEY	VALUE	DESCRIPTION
Authorization	Token 872528e74950686587d646d5fce180e7b5e42e2d	

4. Changing the rate limit of a particular user+API combination(From 10 to 15 here using a drop down menu for User and Rate)

Dashboard

User:
Rate:

Type of User	User Name	Maxrate
dev	Test1	5
org	Test2	10

[Logout](#)

Dashboard

User:
Rate:

Type of User	User Name	Maxrate
dev	Test1	5
org	Test2	15

[Logout](#)

5. JSON response upon 15 requests.

The screenshot shows a REST client interface with a POST request to `http://localhost:8000/api/org`. The 'Headers' tab is active, showing an 'Authorization' header with a token. The 'Body' tab is also active, displaying a JSON response in 'Pretty' format. The response status is 200 OK, with a time of 20ms and a size of 266 B.

KEY	VALUE	DESCRIPTION
Authorization	Token 872528e74950686587d646d5fce180e7b5e42e2d	
Key	Value	Description

```
1  {
2    "success": true,
3    "message": "Success",
4    "count": 15
5  }
```

Status: 200 OK Time: 20ms Size: 266 B Save Response

6. Upon exceeding the rate limit, the following error will be returned as response.

The screenshot shows the same REST client interface, but the response is an error due to exceeding the rate limit. The JSON response in the 'Body' tab shows `"success": false` and `"message": "Rate exceeded"`. The response status is 200 OK, with a time of 11ms and a size of 262 B.

KEY	VALUE	DESCRIPTION
Authorization	Token 872528e74950686587d646d5fce180e7b5e42e2d	
Key	Value	Description

```
1  {
2    "success": false,
3    "message": "Rate exceeded"
4  }
```

Status: 200 OK Time: 11ms Size: 262 B Save Response

Data Storage Format and reason for approach:

1. The requires a lot of read and write operations. The database for this purpose must be structured such that it can be accessed easily and efficiently modified.
2. We are doing server-side processing. Hence we looked for frameworks that can manage the API requests.
3. Most APIs requires user to sign up for an API key in order to use the API.
4. We agreed upon using Django REST Framework.
5. Django allowed us to authorise a user by mapping a unique token to it.
6. The Django has a considerably large community of developers compared to other REST framework developing platforms.
7. It was thus easier for beginners like us to learn the framework and overcome our errors in less time.

Assumptions Made:

1. Existence of Two API endpoints - dev and org
2. Rate limits for (API + User) combination is being managed by a superuser. Super User holds the authority for altering rate limits of a particular combination.
3. Only implementing HTTP POST method for illustrative purposes.

Improvements:

1. Introducing multiple ways for authenticating the client, not just only on basis of user_token but additionally on the basis of application session.
2. Introducing a multi-tier profiling system, allowing users to choose a default rate limit.