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.

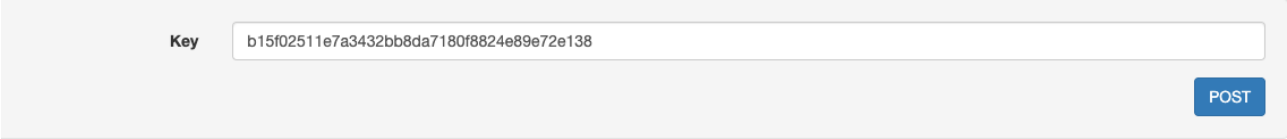
9. Once the user reaches request threshold, user has to wait for the next time window to continue sending requests.

10. If the rate limit is exceeded, a HTTP response of 'rate exceeded' is received by the client .

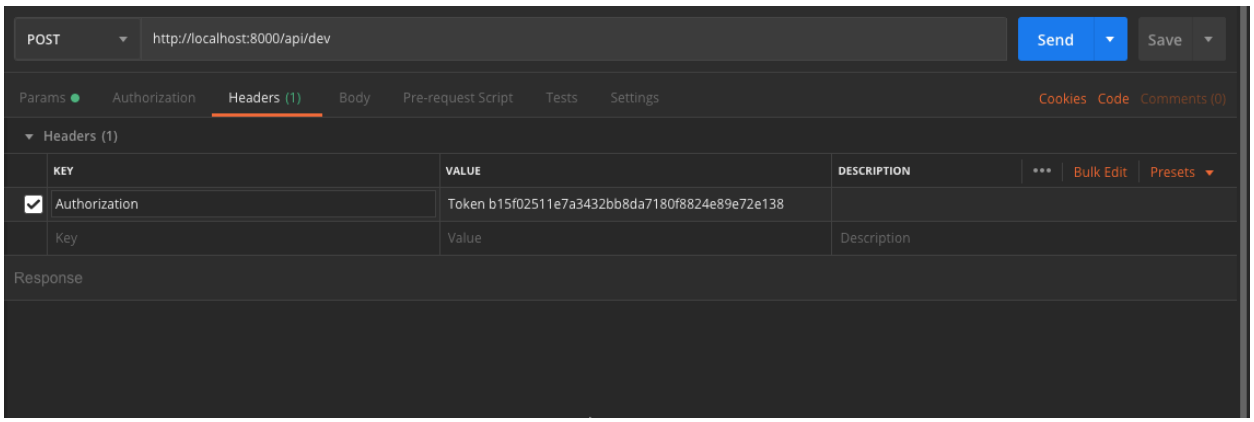
Illustration for a default rate limit:

1. Authenticating a user named ‘Test1’.

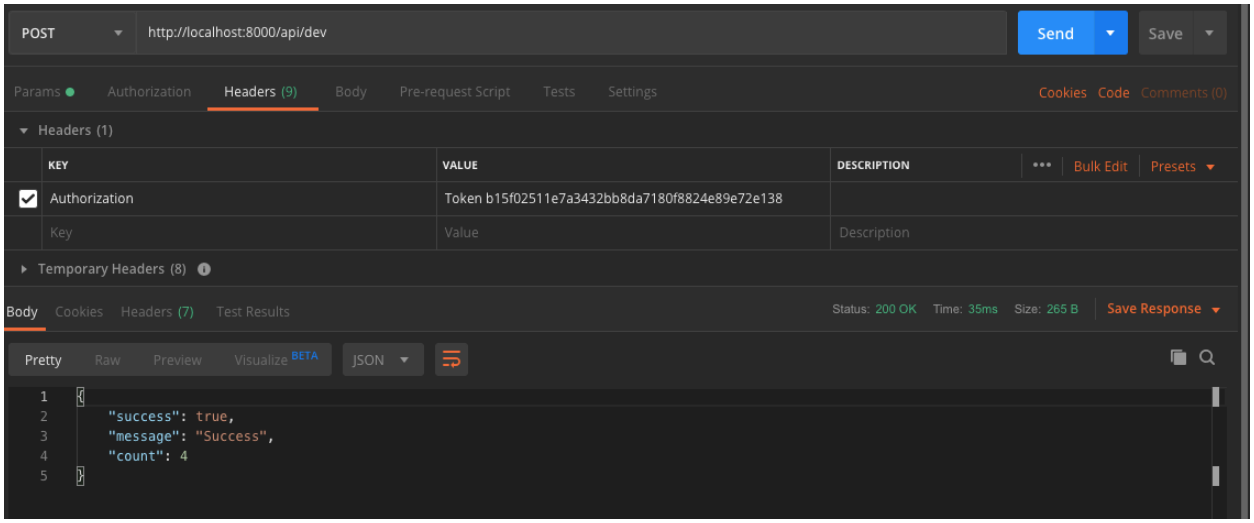
2. This results in generation of a **token**.



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



4. JSON response upon 4 requests.



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

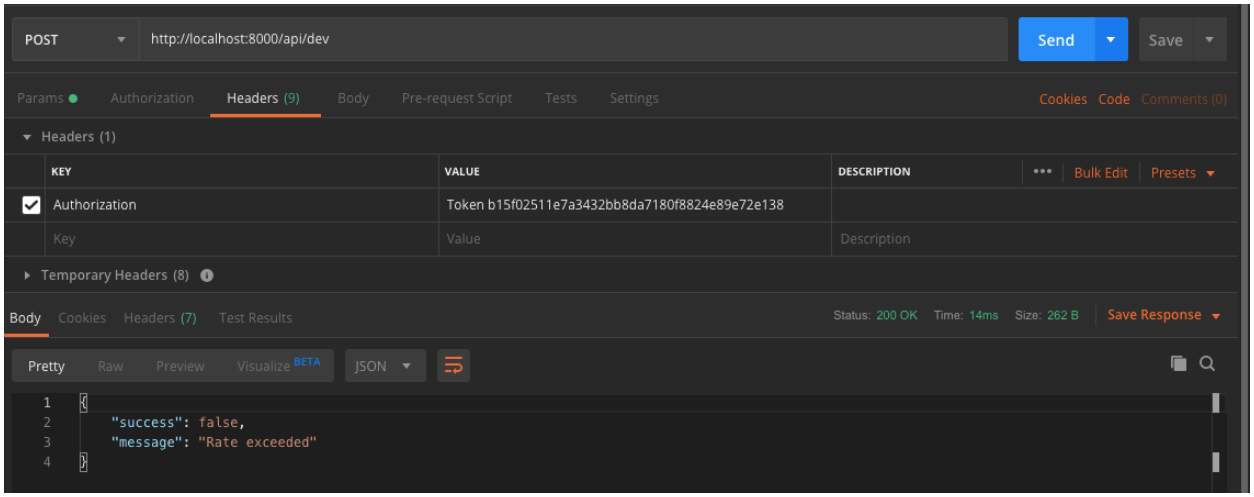
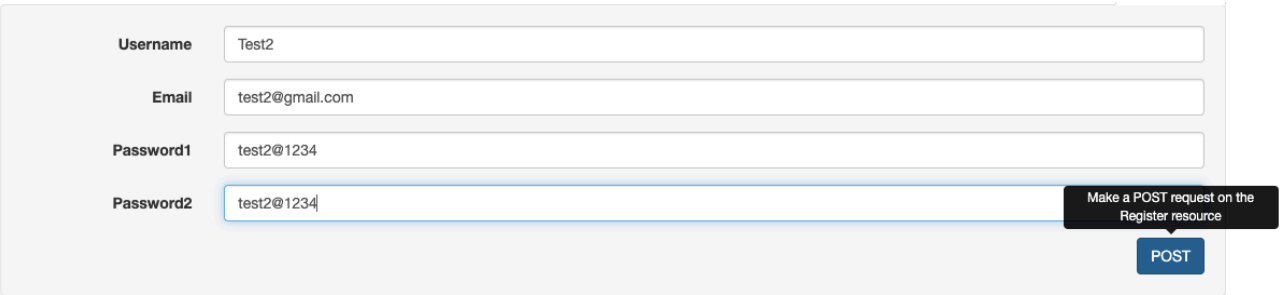
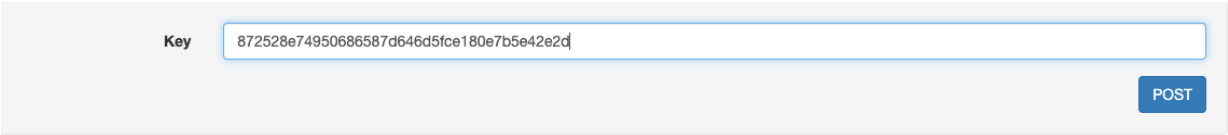


Illustration for an API+User combination rate limit:

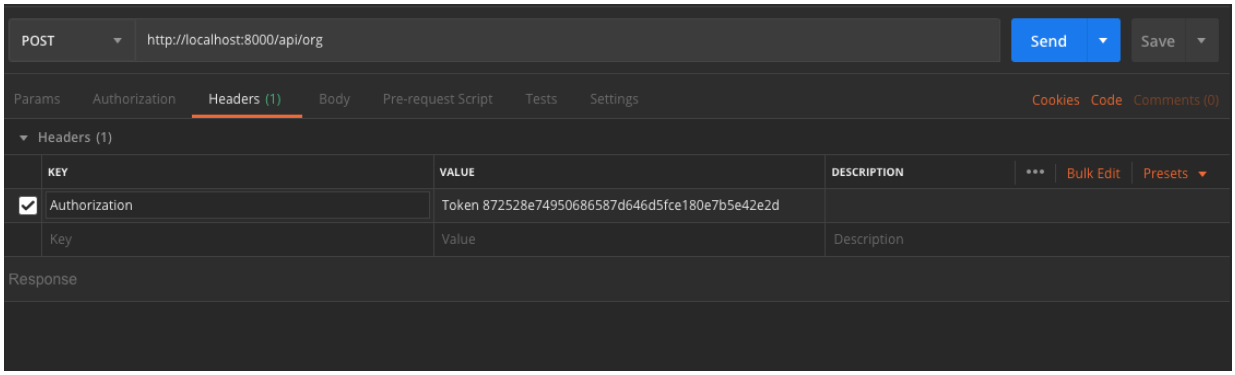
1. Authenticating a user named 'Test2'.



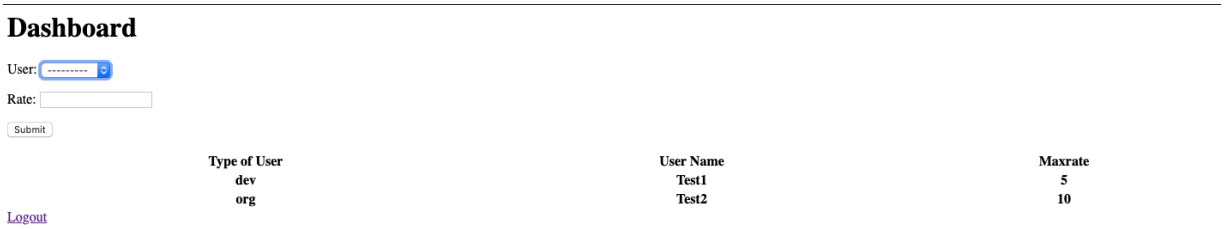
2. This results in generation of a **token**.

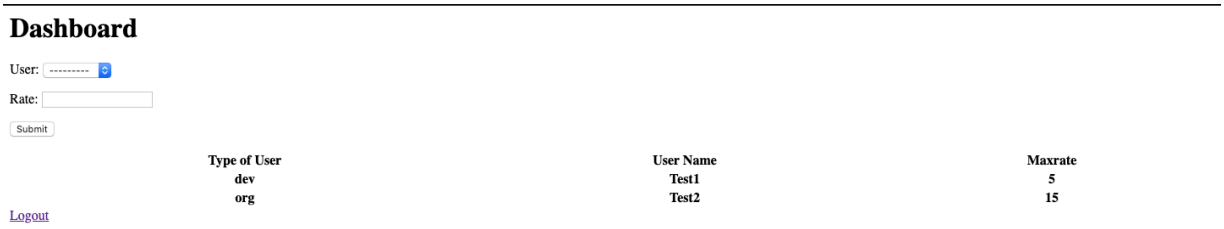


3. Simulating a **POST** request for '**org**' API endpoint on **Postman**.

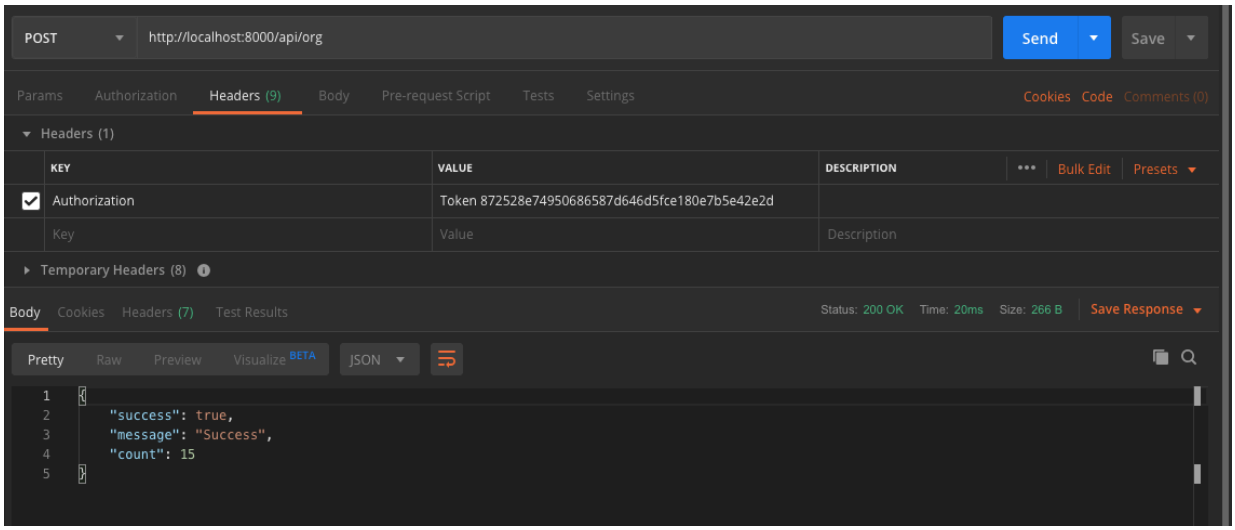


4. Changing the rate limit of a particular user+API combination(From 10 to 15 here using a drop down menu for User and Rate)

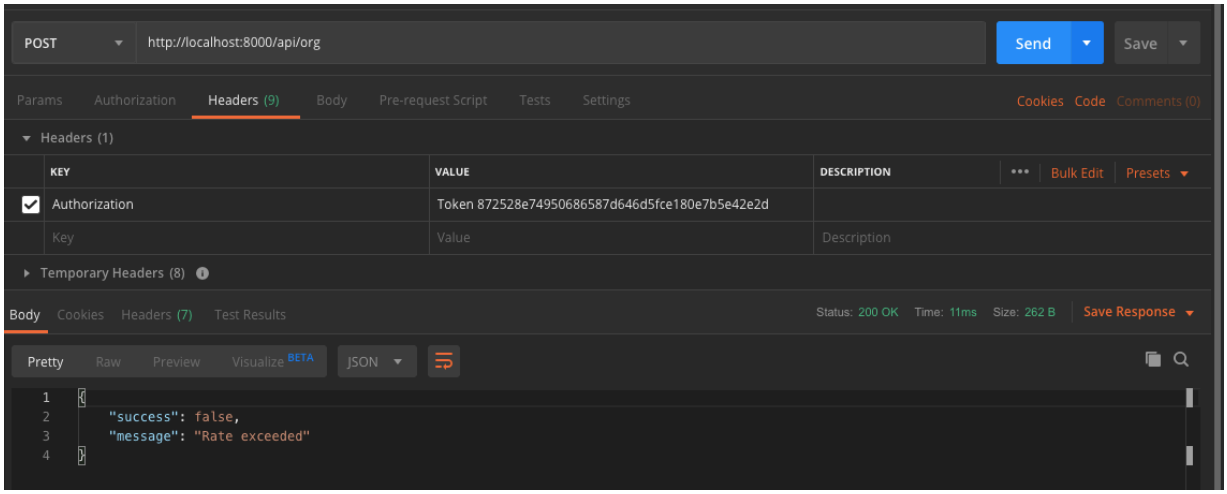




5. JSON response upon 15 requests.



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