The Piazza System: Technical Report

Varun Vijayan
MSc ACT – Cloud Computing
Department of Computer Science and Information Systems,
Birkbeck College, University of London
April 2021

TABLE OF CONTENTS

1. Introduction

The Piazza system is a twitter-like application where users have a wide range of functionality to exchange information. Users are able to post text messages on a range of topics. Users are able to view posts from other users based on topics of interest.

There are engagement functionalities available for users. This is detailed further in the 'Actions' section and test-cases showcase the usage of the same.

There are API end-points available for each action and the test-cases cover all functionality with examples.

The application uses the Django framework and its features to provide the needed functionality. It uses native features from Django like RESTful APIs for each functionality and OAuth Version 2 to provide authentication services to users utilizing the application.

A few additional tools are used to develop this application which are detailed in section 7.

2. Requirements

Application	**Version**
Django	3.2
Django Restframework	3.12.4
Markdown	3.3.4
Django Filter	2.4.0

A file called '***requirements.txt***' is placed in the '***src***' folder which can be used with pip to install all the requirements.

The source code being provided along with this report was developed in a virtual environment contained within the Piazza folder.

Example path for the requirement file in the lab VMs is located in the path below, the highlighted path would be the typical path when the source code is downloaded.

/home/student/piazza/src/requirements.txt

3. Application Functionality

The Piazza applications primary functionality provides six major actions.

i. Authorized users access the Pizza API using oAuth v2 Protocol.
ii. Authorized users post a message for a particular topic in the Piazza API.
iii. Registered Users browse messages per topic using the Piazza API.
iv. Registered Users perform basic operations, including ‘like’, ‘dislike, and ‘comment’ a message posted for a topic.
v. Authorized Users could browse for the most active post per topic that has the highest total number of likes and dislikes.
vi. Authorized Users could browse the history data of expired posts per topic.

This functionality is provided via API End points and they are detailed in the next section.

The application is expected to only allow authenticated users to perform any of the actions listed above. It needs to respond with relevant errors when unauthorized users attempt to perform actions.

The posts are expected to have an expiration time set. Post expiration the posts can only be viewed but no actions can be done on them.

4. REST API End Points

The API End Points available along with their expected responses are listed below. The examples were collected when the server was run on the local system. This results in the primarily IP Address on the API being 127.0.0.1 (loopback address), this needs to be modified to the server/virtual-machine where it is run.

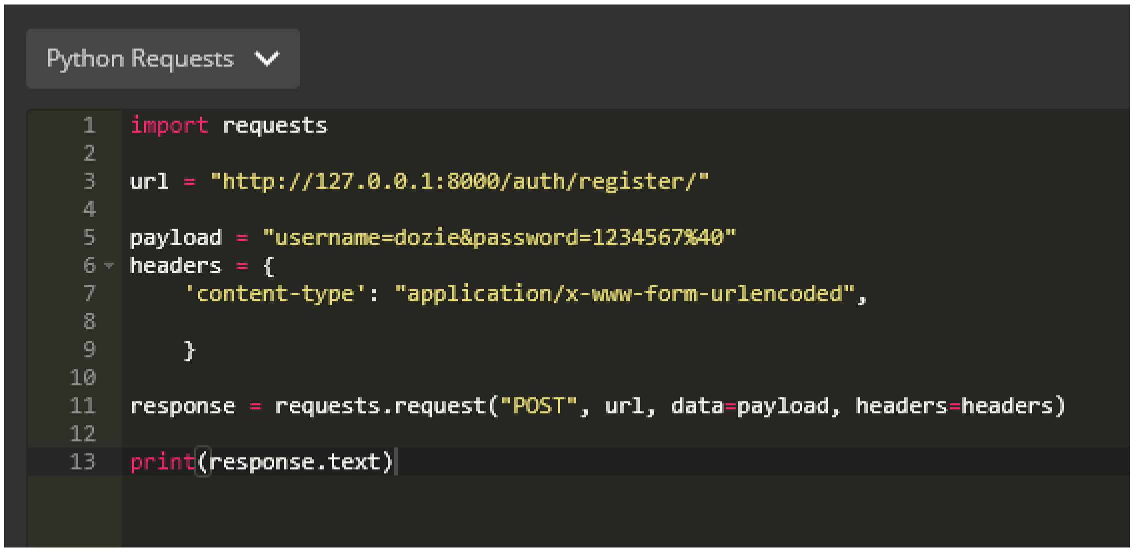
i. Authentication

POST/auth/register/

To register a user post request is sent to the end point with the username and password. A json object containing access token for further authentication is returned.

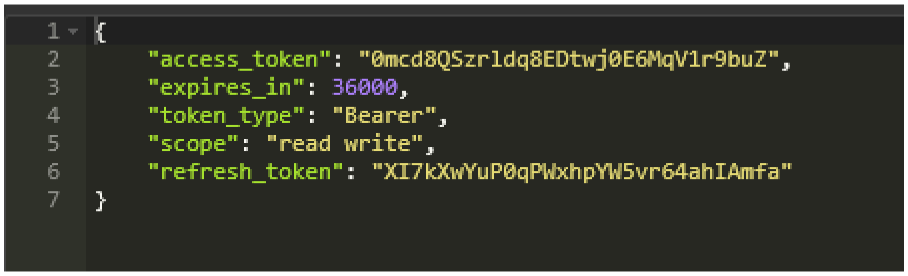
Example:

Request:



```python
import requests

url = "http://127.0.0.1:8000/auth/register/"

payload = "username=dozie&password=1234567%40"
headers = {
    'content-type': "application/x-www-form-urlencoded",

    }

response = requests.request("POST", url, data=payload, headers=headers)

print(response.text)
```

Response:



```json
{
    "access_token": "0mcd8QSzrldq8EDtwj0E6MqV1r9buZ",
    "expires_in": 36000,
    "token_type": "Bearer",
    "scope": "read write",
    "refresh_token": "XI7kXwYuP0qPWxhpYW5vr64ahIAmfa"
}
```

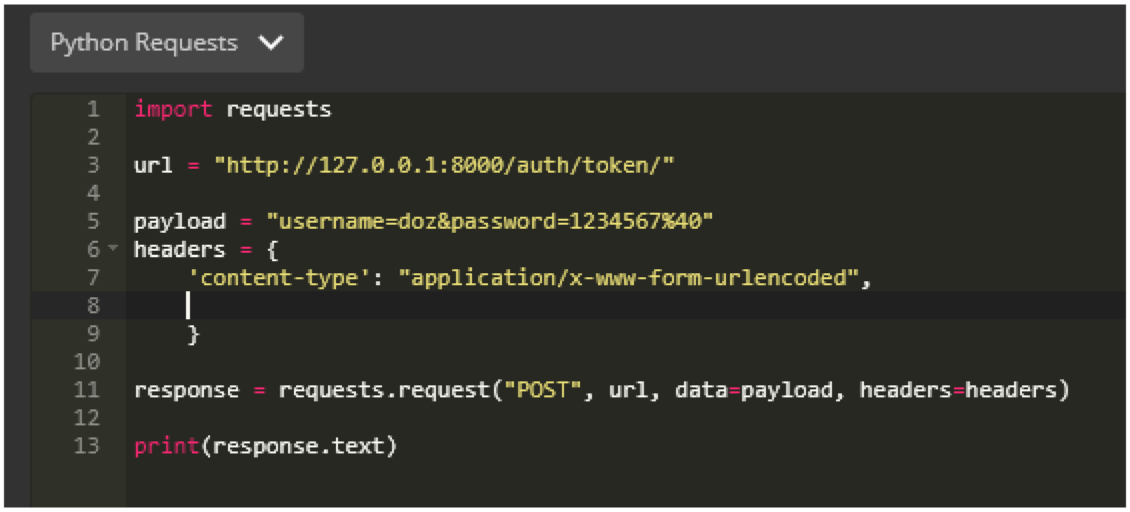
ii. Request Authentication Token

POST`/auth/token/`

This end point is used to get token for already registered users.

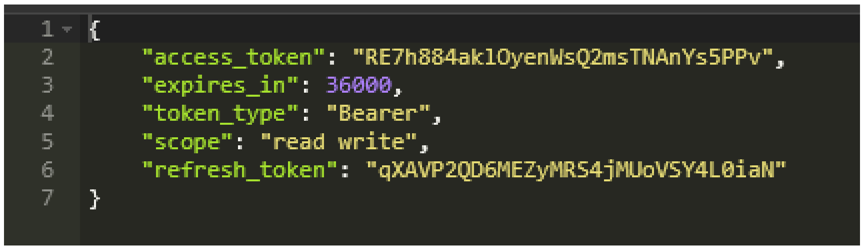
Example:

Request:



```
import requests

url = "http://127.0.0.1:8000/auth/token/"

payload = "username=doz&password=1234567%40"
headers = {
    'content-type': "application/x-www-form-urlencoded",
    
    }

response = requests.request("POST", url, data=payload, headers=headers)

print(response.text)
```

Response:



```
{
    "access_token": "RE7h884ak1OyenWsQ2msTNAnYs5PPv",
    "expires_in": 36000,
    "token_type": "Bearer",
    "scope": "read write",
    "refresh_token": "qXAVP2QD6MEZyMRS4jMUoVSY4L0iaN"
}
```

iii. Post a Message

POST`/posts/`

Authorized user can make a post to this endpoint. It takes the following parameter:

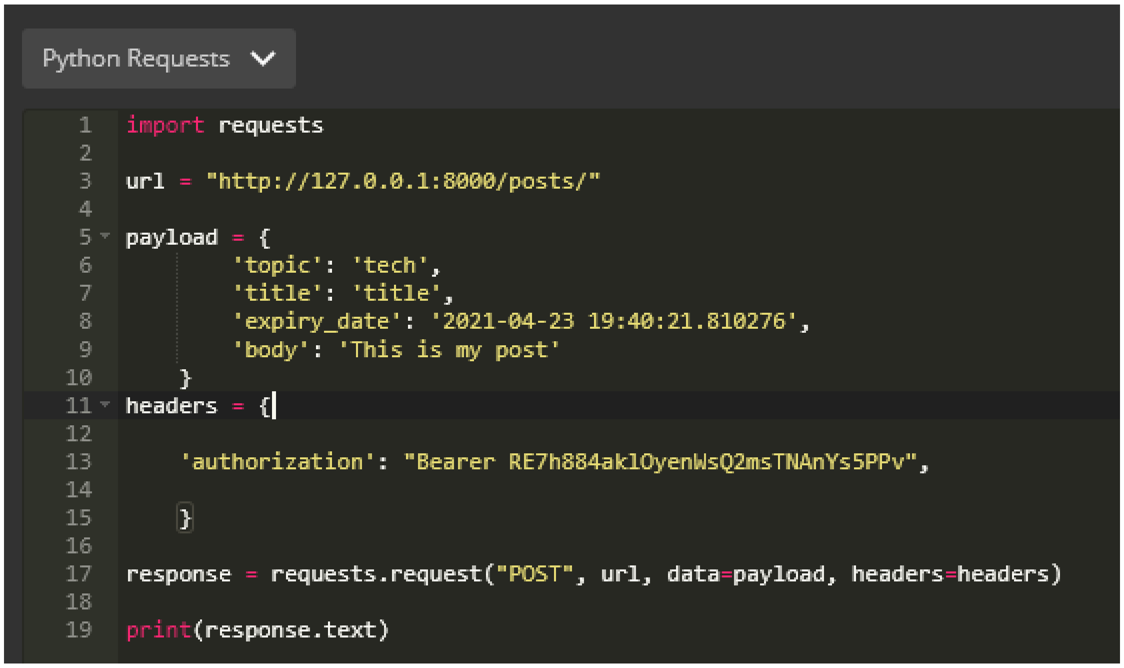
Post{

title*	string *title: Title* *maxLength: 300* *minLength: 1*
topic*	string *title: Topic* Enum: Array [tech, sport, health, politics]
body*	string *title: Body* *minLength: 1*
expiry_date*	string($date-time) *title: Expiry date*

}

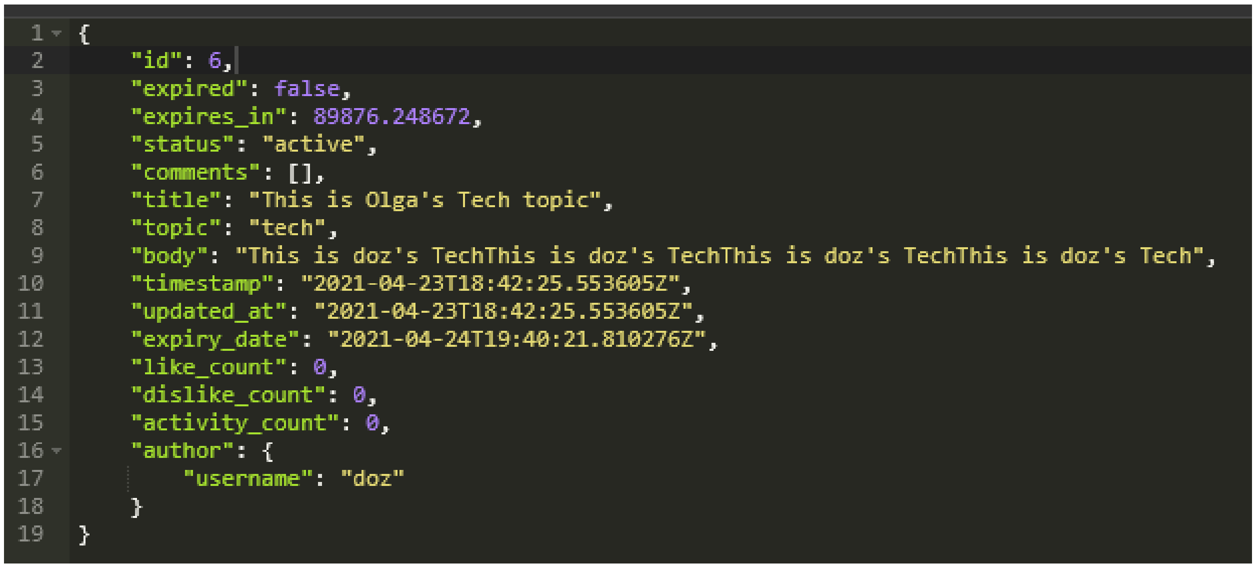
Example

Request:



```python
import requests

url = "http://127.0.0.1:8000/posts/"

payload = {
        'topic': 'tech',
        'title': 'title',
        'expiry_date': '2021-04-23 19:40:21.810276',
        'body': 'This is my post'
    }
headers = {

    'authorization': "Bearer RE7h884ak1OyenWsQ2msTNAnYs5PPv",

    }

response = requests.request("POST", url, data=payload, headers=headers)

print(response.text)
```

Response:



```
{
    "id": 6,
    "expired": false,
    "expires_in": 89876.248672,
    "status": "active",
    "comments": [],
    "title": "This is Olga's Tech topic",
    "topic": "tech",
    "body": "This is doz's TechThis is doz's TechThis is doz's TechThis is doz's Tech",
    "timestamp": "2021-04-23T18:42:25.553605Z",
    "updated_at": "2021-04-23T18:42:25.553605Z",
    "expiry_date": "2021-04-24T19:40:21.810276Z",
    "like_count": 0,
    "dislike_count": 0,
    "activity_count": 0,
    "author": {
        "username": "doz"
    }
}
```

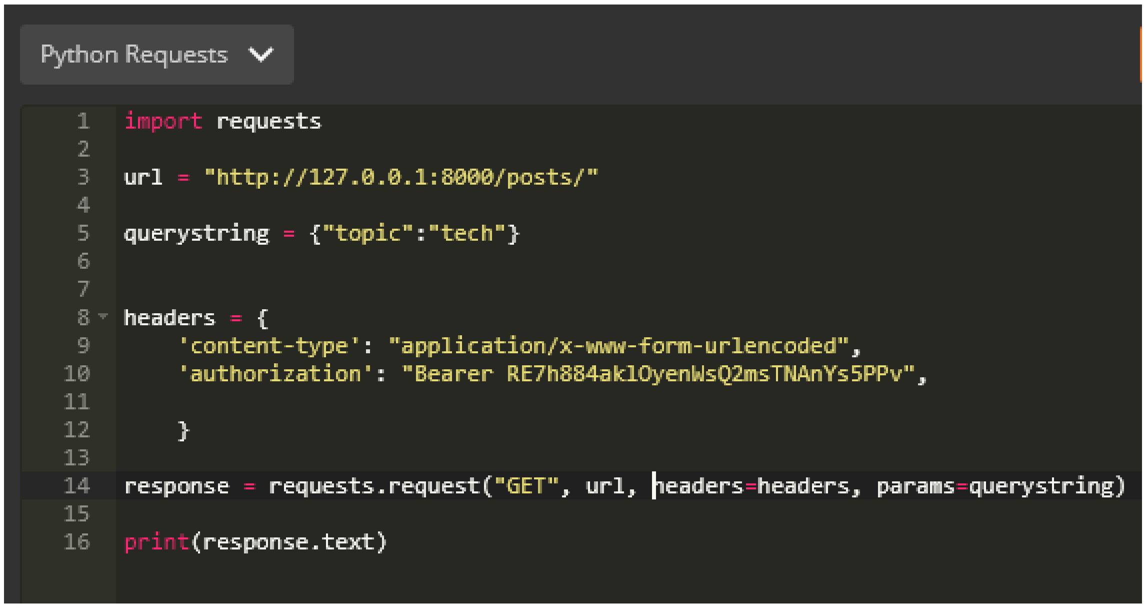
iv. Browse Posts

GET /posts/

Authorized users can browse post per topic, by making a get request to the end point above.

Example:

Request:



Python Requests

```python
import requests

url = "http://127.0.0.1:8000/posts/"

querystring = {"topic":"tech"}


headers = {
    'content-type': "application/x-www-form-urlencoded",
    'authorization': "Bearer RE7h884ak1OyenWsQ2msTNAnYs5PPv",

    }

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)
```

Response:

```
[
    {
        "id": 1,
        "expired": true,
        "expires_in": 0,
        "like_count": 0,
        "dislike_count": 0,
        "status": "expired",
        "activity_count": 0,
        "comments": [],
        "title": "Tech Talk",
        "topic": "tech",
        "body": "Tech Talk Tech Talk Tech Talk Tech Talk Tech
Talk Tech Talk Tech Talk Tech Talk Tech Talk Tech Talk",
        "timestamp": "2021-04-19T06:52:29.501266Z",
        "updated_at": "2021-04-19T14:58:04.259692Z",
        "expiry_date": "2021-04-20T07:51:00Z",
        "author": {
            "username": "dozie"
        }
    },
    {
        "id": 2,
        "expired": true,
        "expires_in": 0,
        "like_count": 0,
        "dislike_count": 0,
        "status": "expired",
        "activity_count": 0,
        "comments": [],
        "title": "This is Olga's Tech topic",
        "topic": "tech",
        "body": "This is Olga's TechThis is Olga's TechThis is
Olga's TechThis is Olga's Tech",
        "timestamp": "2021-04-19T12:30:14.227351Z",
        "updated_at": "2021-04-19T16:24:07.295438Z",
        "expiry_date": "2021-04-19T16:50:18Z",
        "author": {
            "username": "olga"
        }
    },
    {
        "id": 3,
        "expired": true,
        "expires_in": 0,
        "like_count": 2,
        "dislike_count": 1,
        "status": "expired",
        "activity_count": 3,
        "comments": [
            {
                "id": 1,
                "message": "Nick post for Mary",
                "post_id": 3,
                "action": "comment",
                "author": {
                    "username": "nick"
                }
            },
            {
                "id": 2,
                "message": "Olga post for Mary",
                "post_id": 3,
                "action": "comment",
                "author": {
                    "username": "olga"
                }
            }
        ],
        "title": "This is Mary's Tech topic",
        "topic": "tech",
        "body": "This is Mary's TechThis is Mary's TechThis is
Mary's TechThis is Mary's Tech",
        "timestamp": "2021-04-19T12:30:14.241532Z",
        "updated_at": "2021-04-19T16:23:58.158184Z",
        "expiry_date": "2021-04-19T16:50:18Z",
        "author": {
            "username": "mary"
        }
    },
    {
        "id": 4,
        "expired": true,
        "expires_in": 0,
        "like_count": 1,
        "dislike_count": 0,
        "status": "expired",
        "activity_count": 1,
        "comments": [],
        "title": "This is Nick's Tech topic",
        "topic": "tech",
        "body": "This is Nick's TechThis is Nick's TechThis is
Nick's TechThis is Nick's Tech",
        "timestamp": "2021-04-19T12:30:14.259869Z",
        "updated_at": "2021-04-19T16:23:50.858326Z",
        "expiry_date": "2021-04-19T16:50:18Z",
        "author": {
            "username": "nick"
        }
    },
    {
        "id": 6,
        "expired": false,
        "expires_in": 61741.075535,
        "like_count": 0,
        "dislike_count": 0,
        "status": "active",
        "activity_count": 0,
        "comments": [],
        "title": "This is Olga's Tech topic",
        "topic": "tech",
        "body": "This is doz's TechThis is doz's TechThis is
doz's TechThis is doz's Tech",
        "timestamp": "2021-04-23T18:42:25.553605Z",
        "updated_at": "2021-04-23T18:42:25.553605Z",
        "expiry_date": "2021-04-24T19:40:21.810276Z",
        "author": {
            "username": "doz"
        }
    }
]
```

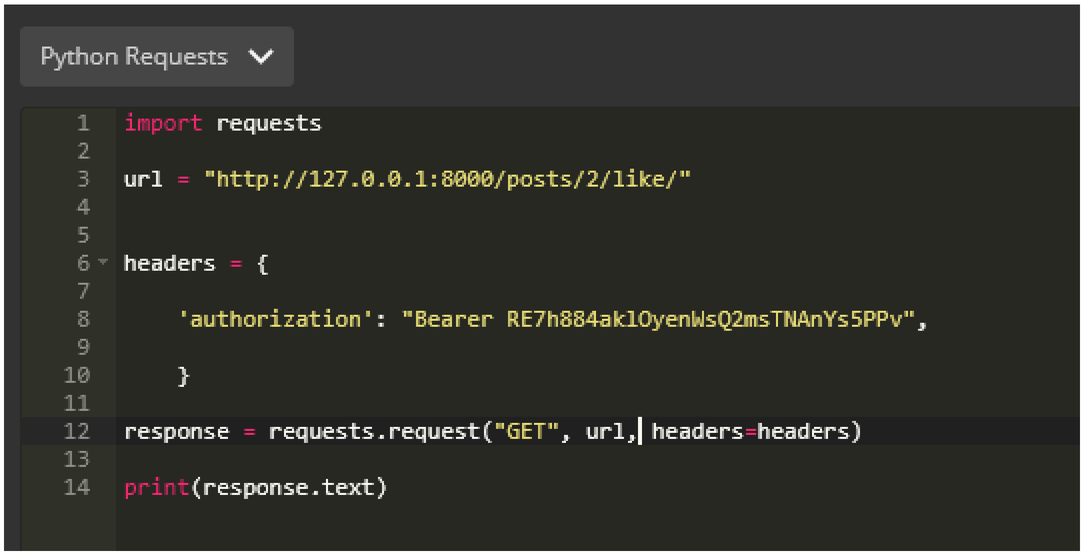
v. Like a Post

`GET/posts/{id}/like/`

To like a post a get request is sent to the above end point. Where {id} is the unique post identifier.

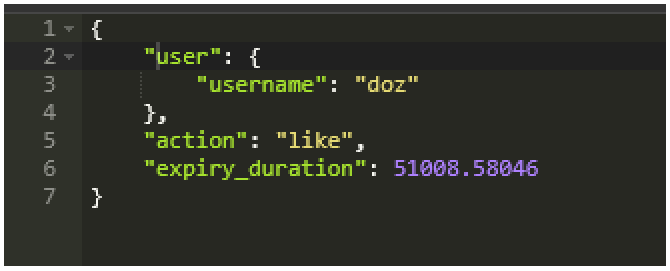
Example:

Request:


Python Requests

```python
import requests

url = "http://127.0.0.1:8000/posts/2/like/"


headers = {

    'authorization': "Bearer RE7h884ak1OyenWsQ2msTNAnYs5PPv",

    }

response = requests.request("GET", url, headers=headers)

print(response.text)
```

Response:



```json
{
    "user": {
        "username": "doz"
    },
    "action": "like",
    "expiry_duration": 51008.58046
}
```

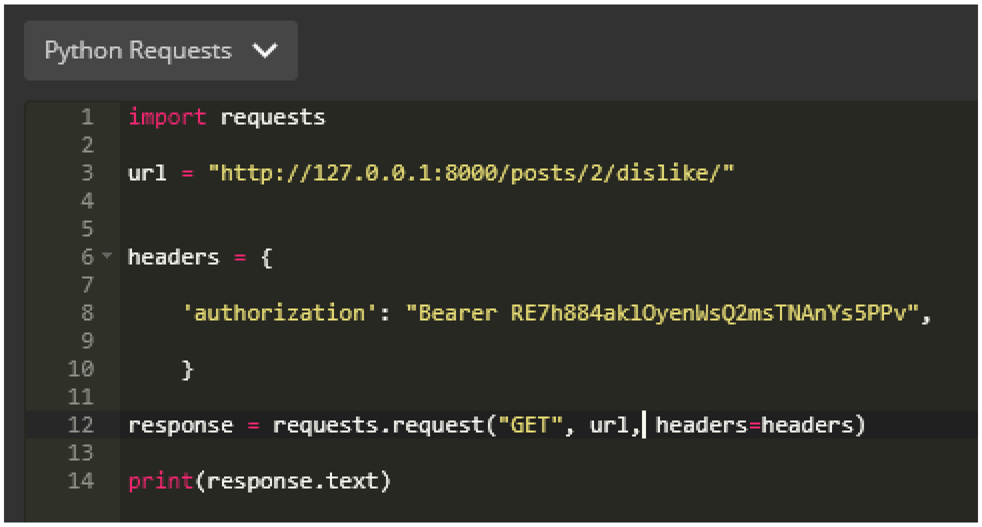
vi. Dislike a post

GET/posts/{id}/dislike/

To dislike a post a get request is sent to the above end point. Where {id} is the unique post identifier.

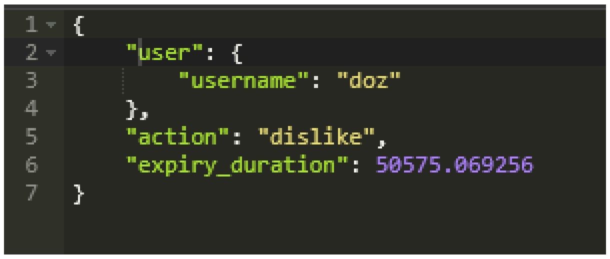
Example:

Request:



```python
import requests

url = "http://127.0.0.1:8000/posts/2/dislike/"


headers = {

    'authorization': "Bearer RE7h884akl0yenWsQ2msTNAnYs5PPv",

    }

response = requests.request("GET", url, headers=headers)

print(response.text)
```

Response:



```json
{
    "user": {
        "username": "doz"
    },
    "action": "dislike",
    "expiry_duration": 50575.069256
}
```

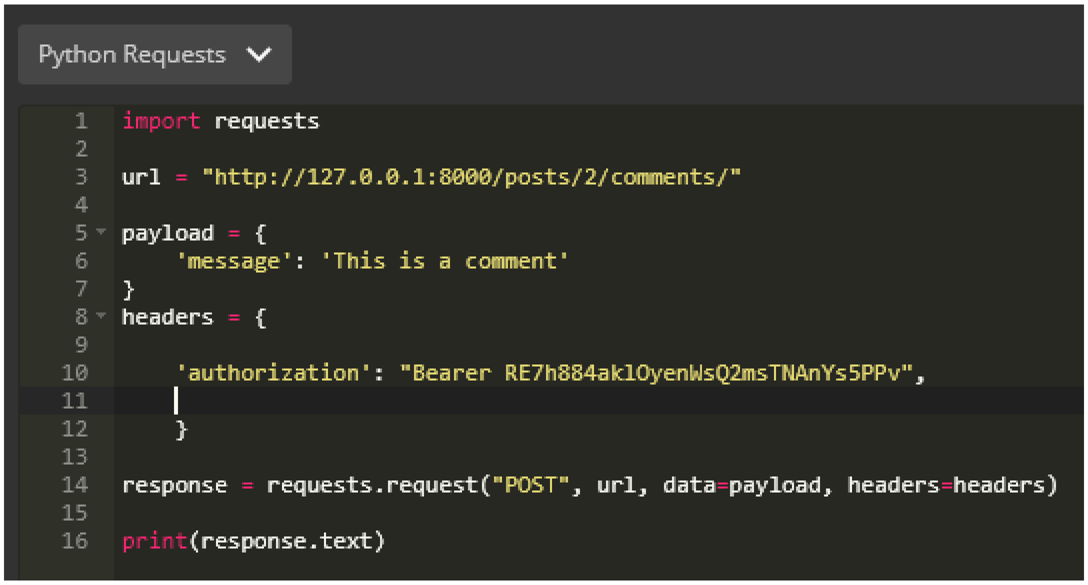
vii. Comment on a post

POST/posts/{id}/comments/

An authorized user can post a comment to post of id {id} using the above endpoint.

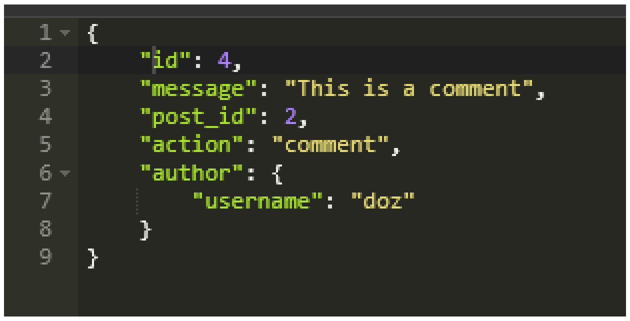
Example:

Request:



```python
import requests

url = "http://127.0.0.1:8000/posts/2/comments/"

payload = {
    'message': 'This is a comment'
}
headers = {

    'authorization': "Bearer RE7h884akl0yenWsQ2msTNAnYs5PPv",

    }

response = requests.request("POST", url, data=payload, headers=headers)

print(response.text)
```

Response:



```json
{
    "id": 4,
    "message": "This is a comment",
    "post_id": 2,
    "action": "comment",
    "author": {
        "username": "doz"
    }
}
```

5. Test Cases

The test cases requested as part of the coursework are summed up in the file ***tests.py*** within '***piazza/src/piazza/tests.py'*** .

It would be the below path when run on the college allocated VM (***10.61.64.83***).

/home/student/piazza/src/piazza/tests.py

The file is fully commented out and the user needs to uncomment one test at a time and proceed.

The 'host' variable on line 6 needs to be set to the local server ip where the server will be run. Additionally, the IP needs to be added to settings.py file which is part of the '***piazza_system***' folder. For eg. When the server is running on the College provided VM (10.61.64.83), the IP Address needs to be added to file '***/home/student/piazza/src/piazza_system/settings.py***' . It needs to be added within the '***ALLOWED_HOSTS'*** array.

```
ALLOWED_HOSTS = ['10.61.64.83']
```

The test cases (TC) file consists of elements which have been not been commented out as there are dependencies to the other test cases.

For eg. The code below lists out Test Case 1 and 2

```
'''
# TC 1 Olga, Nick, Mary and Nestor register and are ready to
access the Piazza API
 print(register('olga', '1234567@'))
 print(register('nick', '1234567@'))
 print(register('mary', '1234567@'))
 print(register('nestor', '1234567@'))
'''
# TC 2 Olga, Nick, Mary and Nestor use the oAuth v2
authorization service to get their tokens (registration is
handled in previous step)
olga = get_token('olga', '1234567@')
nick = get_token('nick', '1234567@')
mary = get_token('mary', '1234567@')
nestor = get_token('nestor', '1234567@')
```

The '***register'*** function is used to register a user. The ***get_token*** function is used to assign a dictionary to the respective user which consists of multiple parameters including an authorization token which can be used to access and perform functionality. Statements of TC2 needs to be left uncommented as the other cases are dependent on the dictionaries being created within TC2.

Additionally, static post ids have been assigned to make sure the other test cases work. Please use the tests file sequentially.

A few Test Cases and their respective outputs are listed below.

i. Test Case 3: Get posts without an authentication token

Code:
```
print(get_posts())
```

Test & Output:
```
(piazza) student@cc:~/piazza/src$ python3 piazza/tests.py
{'detail': 'Authentication credentials were not provided.'}
```

Server Log:
```
[24/Apr/2021 22:33:10] "GET
/posts/?topic=&status=&ordering=&activity= HTTP/1.1" 401 58
```

ii. Test Case 4: Olga posts a message to Tech topic with an expiration time 60seconds

Code:
```
print(
    create_post(
        token=olga['access_token'], topic='tech',
        title="This is Olga's Tech topic",
        duration=60,
        body="This is Olga's TechThis is Olga's TechThis is
Olga's TechThis is Olga's Tech"
    )
)
```

Test & Output:
```
(piazza) student@cc:~/piazza/src$ python3 piazza/tests.py
{'id': 32, 'expired': False, 'expires_in': 59.979994, 'status':
'active', 'comments': [], 'title': "This is Olga's Tech topic",
'topic': 'tech', 'body': "This is Olga's TechThis is Olga's
TechThis is Olga's TechThis is Olga's Tech", 'timestamp': '2021-
04-24T22:37:45.172123Z', 'updated_at': '2021-04-
24T22:37:45.172158Z', 'expiry_date': '2021-04-
24T22:38:45.160522Z', 'like_count': 0, 'dislike_count': 0,
'activity_count': 0, 'author': {'username': 'olga'}}
```

Server Log:
```
[24/Apr/2021 22:37:44] "POST /auth/token/ HTTP/1.1" 200 160
[24/Apr/2021 22:37:44] "POST /o/token/ HTTP/1.1" 200 169
[24/Apr/2021 22:37:44] "POST /auth/token/ HTTP/1.1" 200 160
[24/Apr/2021 22:37:44] "POST /o/token/ HTTP/1.1" 200 169
```

```
[24/Apr/2021 22:37:44] "POST /auth/token/ HTTP/1.1" 200 160
[24/Apr/2021 22:37:44] "POST /o/token/ HTTP/1.1" 200 169
[24/Apr/2021 22:37:45] "POST /auth/token/ HTTP/1.1" 200 160
[24/Apr/2021 22:37:45] "POST /o/token/ HTTP/1.1" 200 169
[24/Apr/2021 22:37:45] "POST /posts/ HTTP/1.1" 201 427
```

6. Database Model

The code which forms the Database model used is listed below.

```python
class Post(models.Model):
    TECH = 'tech'
    SPORT = 'sport'
    HEALTH = 'health'
    POLITICS = 'politics'

    TOPIC_CHOICES = (
        (TECH, 'Tech'),
        (SPORT, 'Sport'),
        (HEALTH, 'Health'),
        (POLITICS, 'Politics'),
    )
    ACTIVE = 'active'
    EXPIRED = 'expired'

    STATUS_CHOICES = (
        (ACTIVE, 'Active'),
        (EXPIRED, 'Expired'),
    )

    title = models.CharField(max_length=300)
    topic = models.CharField(max_length=300, choices=TOPIC_CHOICES)
    body = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    expiry_date = models.DateTimeField()
    status = models.CharField(max_length=300, choices=STATUS_CHOICES,
default='active')
    owner = models.ForeignKey(User, related_name='blog_posts',
on_delete=models.CASCADE)

    def __str__(self):
        return self.title

    @property
    def has_expired(self):
        return timezone.now() > self.expiry_date

    def expiry_duration(self):
        e = self.expiry_date - timezone.now()
        if e.total_seconds() < 0:
            return 0
        return e.total_seconds()

    def set_status(self):
        if self.has_expired:
            return self.EXPIRED

        return self.status


class Like(models.Model):
    post = models.OneToOneField(Post, related_name='likes', on_delete=models.CASCADE)
    users = models.ManyToManyField(User, related_name='post_likes')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.post.title


class Dislike(models.Model):
    post = models.OneToOneField(Post, related_name='dis_likes',
on_delete=models.CASCADE)
    users = models.ManyToManyField(User, related_name='post_dis_likes')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.post.title


class Comment(models.Model):
    post = models.ForeignKey(Post, related_name='comments', on_delete=models.CASCADE)
    owner = models.ForeignKey(User, related_name='user_comments',
on_delete=models.CASCADE)
    message = models.TextField()

    def __str__(self):
        return str(self.message)[:30]
```

We have the Post, Comment, Like and Dislike model. The Post model has required fields topic (a choice filed of tech, sport, health & politics), title, body, expiry_date and status. Status fields will be used to tell the current state of the post, whether it is expired or active.

The ‘like’ and ‘dislike’ models are similar models. They contain a post field with a one-to-one relationship with the Post model and also a many-to-many relationship to the user model via the ‘users’ field. The ‘users’ field will be used to determine the number of likes and dislike.

7. Additional Tools Used

Additional tools used to build the application are listed below.

i. oAuthv2: oAuthv2 is used to provide authentication and authorization services to users to utilize the application

ii. RESTful Framework: REST APIs are used to act as end points for all system functionalities.

iii. Django Filter: Django-filter is used, it allows users to filter down a query-set based on a model’s fields and is also customizable

8. References

1. OAuth:
   https://django-oauth-toolkit.readthedocs.io/en/latest/rest-framework/getting_started.html

2. Django Filter:
   https://www.django-rest-framework.org/api-guide/filtering/#djangofilterbackend

3. RESTful Framework:
   https://www.django-rest-framework.org/#quickstart

4. TweetMe App and Video Tutorial:
   https://github.com/codingforentrepreneurs/Tweetme-2
   https://www.youtube.com/watch?v=f1R_bykXHGE

5. Seat Allocation Django App:
   https://github.com/db-coder/Seat_Allocation

6. Cloud Computing LAB Guide 1,2,3,4