Piazza: Development of a Cloud Software as a Service

Montel Moore

CONTENTS

1 SUMMARY

The aim of this project was to develop an RESTful API (separated, stateless, cacheable, uniform interface) for a “message posting” system, where users authenticate using the Oauth2 authentication framework.

The documentation outlines the project setup, authentication system, database setup and testing application. Additionally, the project’s adherence to RESTful protocols is appraised.

2 SETUP OF VIRTUALISED ENVIRONMENT FOR TESTING AND DEVELOPMENT

2.1 PACKAGES INSTALLED:

- Django == 3.0.2 – Framework for creating the project, models and app. [1]
- django-filter == 2.4.0 – to provide a filtering backend for the model views [2]
- django-oauth-toolkit == 1.4.0 – provides the Oauth2 authentication tools [3]
- django-property-filter == 1.1.0 – allows filtering via model class properties [4]
- djangorestframework == 3.12.2 – provides the packages necessary for to create REST APIs [5]

2.2 FILE STRUCTURE:

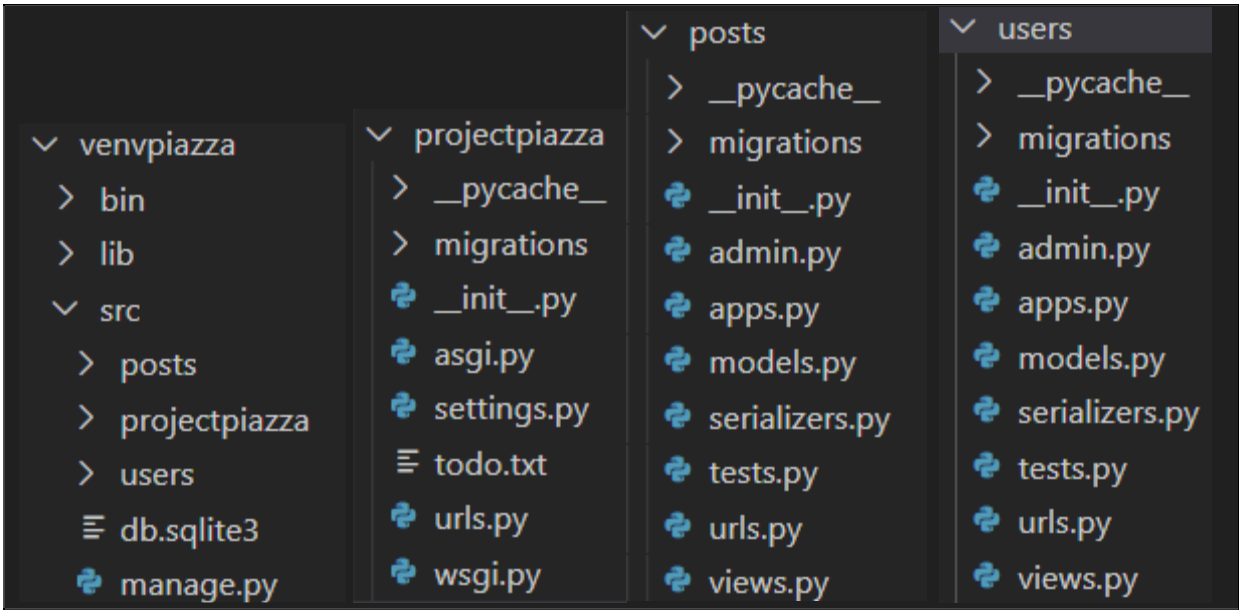

Figure 1: source folder, project folder and app folders

3 OAUTHV2 PROTOCOL AND IMPLEMENTATION

The Oauth protocol defines four roles: [6]

Resource owner – This is the end user. In the context of Piazza, the user is the person who creates and interacts with posts.

Resource server – The server hosting the protected resources i.e. posts. Authenticated users retrieve and add data through making requests to the resource server.

Client – This makes requests to the resource server on behalf of the resource owner, using the resource owner's token. In the context of this project, the resource owner and client are the same entity. However, if the scope was extended and Piazza became a visual frontend application, the application (e.g. browser or mobile application) would be the client, accepting input from the resource owner and making requests on behalf of the resource owner with their token.

Authentication server – this server issues access tokens to the client after authenticating the user credentials/refresh tokens.

In the context of piazza, the Authentication server is the same server as the resource server.

Settings.py was adjusted to allow Oauth to appear on the admin site and oauth2 to be set as the default authentication backend and authentication class.

The Oauth2 flow, as implemented in Piazza, is described as below:

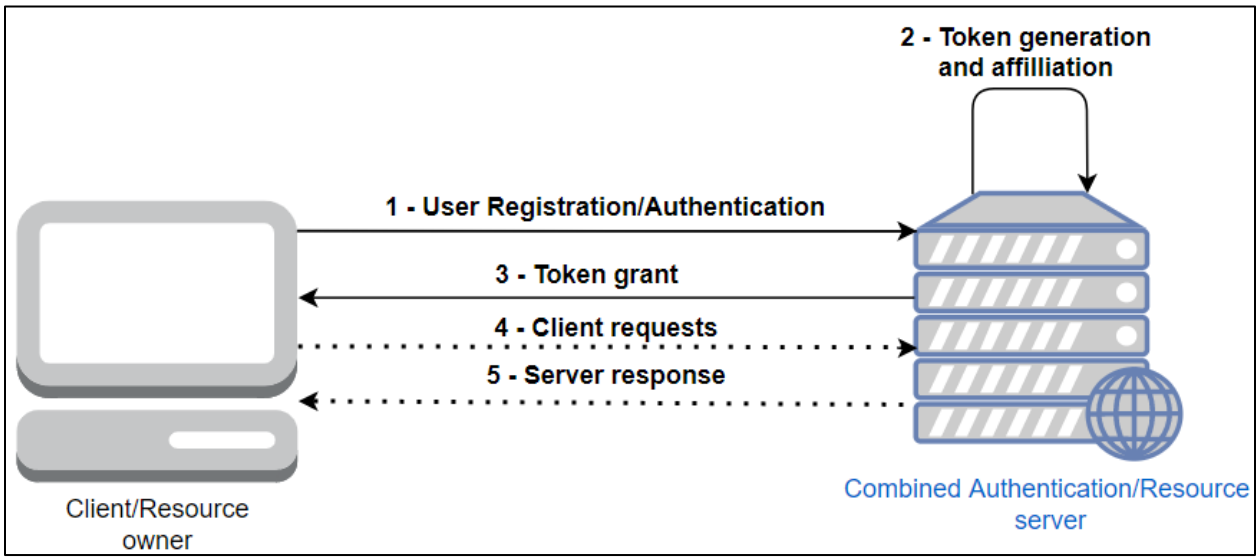

Figure 2: Client – server interaction. Created on https://online.visual-paradigm.com/

1. **User Registration/Authentication:** User/Resource owner registers on the server using a username and password/already registered user requests a token with their username and password/already registered user uses their refresh token to request new tokens.
2. **Token generation and affiliation:** Server generates new access and refresh tokens, links tokens to user in the database

3. **Token grant:** Client is provided with an access and refresh token
4. **Client requests:** Client makes requests to the combined resource/authentication server on behalf of the user using their valid access token. The user can be identified by their token.
5. **Server response:** The server responds to client requests. This can be repeated with step one as long as the token is valid.

4 DATABASE MODEL/API ENDPOINT DESCRIPTION

Two models were essential for this project: User and Post. The entity-relationship diagram is shown in Figure 3, and the description of each model and implementations and endpoints will be described in the subsections below.

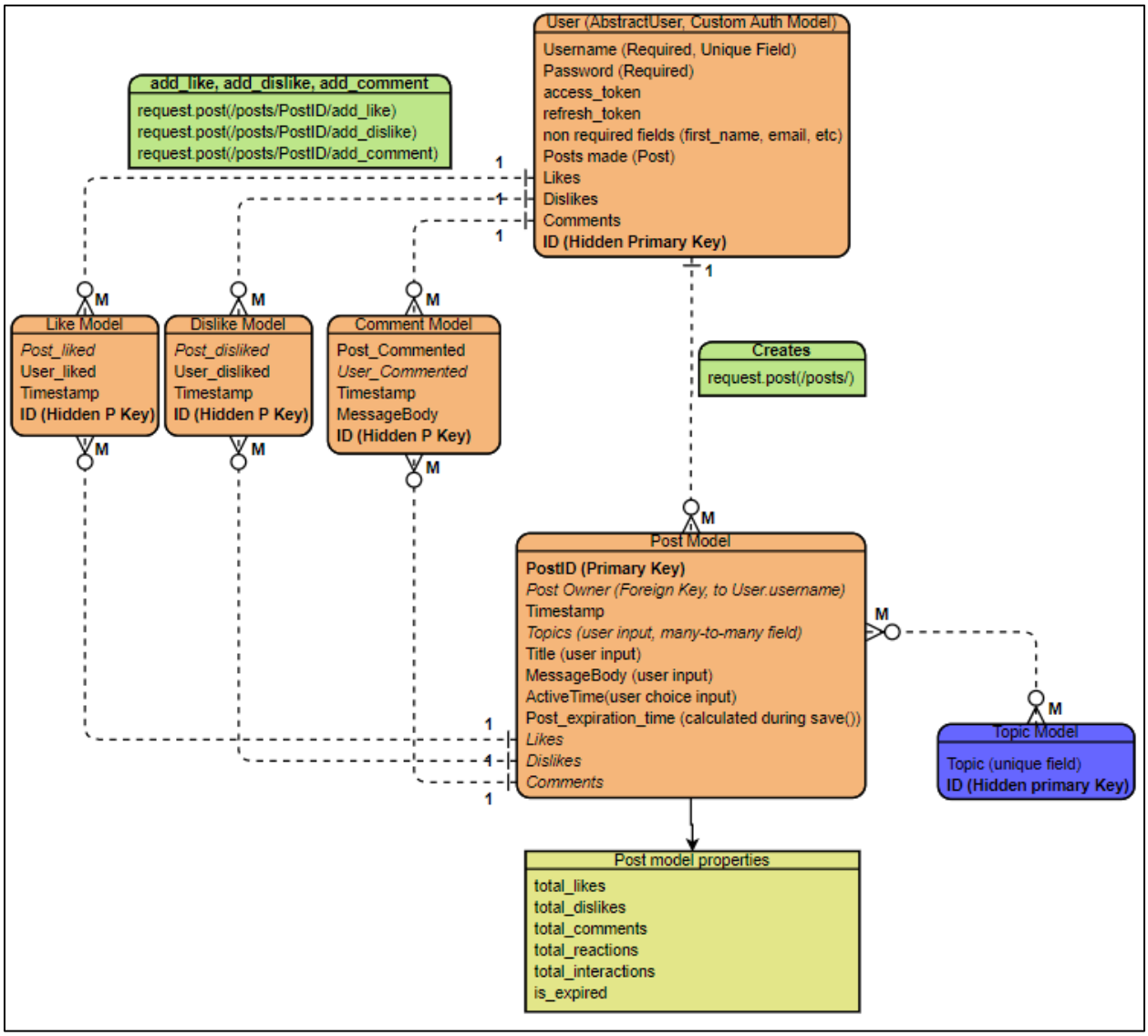

Figure 3: Piazza entity relationship diagram. Created on https://online.visual-paradigm.com/

4.1 Users

A custom user model was used for piazza. This model inherited from the Django auth user model [7], with the addition of *access_token* and *refresh_token* fields, obtained from the authorization server.

The settings.py file was configured to register the custom user model as the default user model for this project.

Figure 4 shows the method used to request an access token, and create/associate a user instance with the access token. The user sends a request containing a username and password to *10.61.64.70:8000/authentication/register/*, which creates a user model. The data is then posted to *10.61.64.70:8000/o/token/* which returns access and refresh tokens. The tokens are then affiliated with the created user, allowing the user to be identified by their access token when creating posts.

```
r = requests.post(IP_token,
    data={
        'grant_type': 'password',
        'username': request.data['username'],
        'password': request.data['password'],
        'client_id': CLIENT_ID,
        'client_secret': CLIENT_SECRET,
    },
)
User.objects.update_or_create(
    username = request.data['username'],
    defaults = {
                "access_token" : r.json()["access_token"], #saving the tokens to the user
                "refresh_token": r.json()["refresh_token"]})
```

Figure 4: views.register token affiliation method.

4.1.1 User Endpoints

10.61.64.70:8000/o/token/: Internal endpoint to request/refresh user tokens. Appears as variable IP_token in Figure 4.

10.61.64.70:8000/o/revoke_token/: Internal endpoint to revoke user tokens.

10.61.64.70:8000/admin/: Interface for admin users

10.61.64.70:8000/authentication/register/: Endpoint to allow users to register

10.61.64.70:8000/authentication/token/: Endpoint to allow registered users to get their tokens

10.61.64.70:8000/authentication/token/refresh/: Endpoint to refresh token

10.61.64.70:8000/authentication/token/revoke/: Endpoint to revoke token

10.61.64.70:8000/authentication/users/: get user details (admin users only)

4.2 Posts

4.2.1 Post model

The post model is designed for the end user to create, edit and view posts, governed by a set of constraints and features.

The post fields are described as follows:

PostID: The primary key of the post
Post_owner: Foreign key, connecting the user to the post
Timestamp: The time of post creation
Topics: User input. Many-to-many field, connected to a "Topic" model with choices of "Tech", "Health", "Politics" and "Sport"
Title: User input. The title of the post
MessageBody: User input. The main content of the post.
ActiveTime: User input. The period of time that a post is active for.
Post_expiration_time: The time of expiry, after which no comments, likes or dislikes can be made. Calculated during the first instance of the save() method, as shown below:

```
def save(self, *args, **kwargs):
    """ Populates the post_expiration_time value on creation of the post. """
    if self.Post_expiration_time == None:
        self.Post_expiration_time = now() + self.ActiveTime
    super(Post, self).save(*args, **kwargs)
```

Figure 5: Save method, setting the post expiration time.

likes, dislikes, comments: ManyToManyFields, connected to the user model through intermediate tables "Like", "Dislike" and "Comment"

4.2.2 Related post models

The like, dislike and comment models were created as intermediate tables relating to the post model. The three models contain the timestamp data, with the comment model containing an additional MessageBody field.

The Topic model was created in to allow users to input multiple topic choices. This can be done with a choice field, which requires a multiple choice form widget to allow multiple choices to be selected [8]. This option requires the inclusion of model forms, which were not used in this project.

4.2.3 Creating a post

As mentioned in 3, the user's access token, rather than their username, is used to associate the user with their post. Their access token is gained from the request headers and the server looks up the user associated with that access token. The server then populates the Post_owner field on behalf of the user:

```
access_token = get_access_token(request)
r = request.data.copy()
r["Post_owner"] = User.objects.get(access_token = access_token).username
```

4.2.4 *Post constraints*

Post model creation and interaction was guided by a set of constraints, which will be detailed below:

1. *Only registered and authenticated users can view and interact with posts:*
   Appropriate permission and authentication classes were added to the Post view set:
   ```
   permission_classes = [IsAuthenticated | IsAdminUser]
   authentication_classes = [OAuth2Authentication, SessionAuthentication]
   ```
2. *Posts expire at the end of post expiration time (governed by the user) after which no user can like, dislike or comment on the post.*
3. *A user cannot like or dislike their own post*
4. *If a user has already liked/disliked a post, repeating that action will undo the like/dislike. If a user dislikes post that they have liked*
   The save methods of the like, dislike and comment classes were overridden in order to satisfy constraints 2-4:
   ```
   def save(self, *args, **kwargs):
       if now() >= Post.objects.get(PostID = self.Post_liked.PostID).Post_expiration_time:
           raise ValidationError("This post has expired. You cannot interact with this post.")

       elif self.User_liked == Post.objects.get(PostID = self.Post_liked.PostID).Post_owner:
           raise ValidationError(_('You cannot like/dislike your own post'))

       # like and dislike cannot exist at the same time
       elif Dislike.objects.filter(Post_disliked = self.Post_liked, User_disliked = self.User_liked):
           Dislike.objects.filter(Post_disliked = self.Post_liked, User_disliked = self.User_liked).delete()

       # ability to remove like from post
       elif Like.objects.filter(Post_liked = self.Post_liked, User_liked = self.User_liked):
           Like.objects.filter(Post_liked = self.Post_liked, User_liked = self.User_liked).delete()
           raise ValidationError(_('User has already liked this post. Post un-liked.'))

       super(Like, self).save(*args, **kwargs)
   ```
5. *A user cannot delete another user's post*
   The destroy() method of the post view set was overwritten to prevent a user from deleting another user's post, using similar logic to that described in 4.2.3. If the user sends a DELETE request to the URL of another user's post, they are returned an error message.

There are additional constraints/functionality that should be placed on user requests (allowing users to delete their comments, preventing PUT requests, allowing users to edit their own posts), however these features were not included within the scope of this project.

4.2.5 *Post filtering/ordering by properties and attributes*

Properties were defined for the post model. These properties returned the total likes, dislikes, reactions, comments, interactions and expiry status of the post. The main purpose of these properties was to posts to be ordered by popularity, or filtered by their expiry status.

Users are able to filter posts by topic, owner, topic and active status, and order by total likes/comments/interactions/reactions/timestamp. The Django_filters [2] and Django-property-filter [4] packages were used to achieve this functionality, with some limitations to performance. [9]

4.3 STANDARD END USER FLOW

1. The user makes a request to register to the API by providing credentials to **10.61.64.70:8000/authentication/register/.**
2. The user receives their access and refresh tokens, and is able to access the Posts app by using their access token.
3. The user can post their message by sending a Post request to any list based view of posts i.e. **10.61.64.70:8000/posts/** or **10.61.64.70:8000/posts/?<attribute=>/**.
4. Users can view all posts under **10.61.64.70:8000/posts/,** (ordered by timestamp by default) and filter/order posts under **10.61.64.70:8000/posts/?<attribute=>/**. These filters can be stacked. Examples of URLs are listed below:
    a. **10.61.64.70:8000/posts/?is_expired__exact=false/:** Filter posts by active posts only
    b. **10.61.64.70:8000/posts/?is_expired__exact=true&ordering=-total_reactions:** Filter by all expired posts, ordered by the total reactions in descending order
    c. **10.61.64.70:8000/posts/?Topics=Tech&is_expired__exact=true&ordering=-total_reactions:** Filter by all expired posts in the tech topic, ordered by the total reactions in descending order
5. The user can like, dislike, comment or remove their likes and dislikes by sending a **10.61.64.70:8000/posts/<pk>/<add_interaction>/** or **10.61.64.70:8000/posts/<pk>/<remove_interaction>/,** where interactions are likes, dislikes and comments (comments cannot be removed)
6. A user can view a single post by sending a GET request to **10.61.64.70:8000/posts/<pk>/.** If the user is the post owner, they can delete their post by sending a DELETE request to the same URL.

4.3.1 Post endpoints summary

10.61.64.70:8000/posts/: Post list view, to view all posts

10.61.64.70:8000/posts/<pk>/: Post instance view, to view a post by its primary key pk. Post owner can also delete post if delete request is made.

10.61.64.70:8000/posts/<pk>/<add_interaction>/: the <add_interaction> field can take add_like, add_dislike, add_comment, remove_like, remove_dislike

10.61.64.70:8000/posts/?<attribute=>/: to filter or order by an attribute, where <attribute> = Topics, Post_owner, popularity, timestamp, is_expired, likes, dislikes, comments. These attribute filters can be concatenated.

5 ADHERENCE TO REST ARCHITECTURE

This section will describe Piazza's adherence to the five mandatory REST constraints [10]:

Client-server architecture:

The client is separated from the server. The client application is able to evolve separately from the server. This is demonstrated by the testing application in *6*.

Statelessness:

The state of the application in *6* is stored on the application itself. The server has no knowledge of the application state. The resources are stored on the server and are independent of the client's current context.

Cacheability:

The client application will cache the server responses appropriately. In the context of Piazza, the testing application stores the access tokens.

Layered system

This is the least RESTful property of the Piazza system, as there is only one degree of separation between the client and the server(s). The resource and authentication servers are the same, however they are partitioned to perform different actions (Oauth2 partition and API partition).

Uniform interface

When the client sends a request to the server, the server has all the information it needs to provide an appropriate response. However, The client cannot simply "browse" the api, as the server does not provide the client with hypermedia links. This violates the HATEOAS [11] constraint of RESTful style architecture. The server should either provide suitable follow-up URLs in the JSON formatted responses (example in Figure 6), or alternatively a webpage with suitable hyperlinks could be built.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.acme.account+json
Content-Length: ...

{
    "account": {
        "account_number": 12345,
        "balance": {
            "currency": "usd",
            "value": 100.00
        },
        "links": {
            "deposit": "/accounts/12345/deposit",
            "withdraw": "/accounts/12345/withdraw",
            "transfer": "/accounts/12345/transfer",
            "close": "/accounts/12345/close"
        }
    }
}
```

Figure 6: Example of a server response that satisfies the HATEOAS constraint. The Account model is returned and the hyperlinks are also returned.

6 TEST CASES

TC 1. olga, nick, mary, and nestor register and are ready to access the Piazza API.

```
import requests
registration_url = "http://10.61.64.70:8000/authentication/register/"

olga, nick, mary, nestor = (
{
    "username": "Olga",
    "password": "1234"
},
{
    "username": "Nick",
    "password": "1234"
},
{
    "username": "Mary",
    "password": "1234"
},
{
    "username": "Nestor",
    "password": "1234"
})
```

olga

```
olga_response = requests.post(registration_url, data = olga) # Sending request to register username and password data

olga_response = olga_response.json()

print(olga_response)
```

```
{'access_token': 'aWraC4n7PJPHHJwE1fOO6Q58FXdu0g', 'expires_in': 36000, 'token_type': 'Bearer', 'scope': 'read write', 'refresh
_token': 'Ew6kCwIM29GiRHkwg0thXVCoZ3CWjc'}
```

nick

```
nick_response = requests.post(registration_url, data = nick)

nick_response = nick_response.json()
print(nick_response)
```

```
{'access_token': '8VeR5GziRwHHpLzD8SGGJHoQlrJjm9', 'expires_in': 36000, 'token_type': 'Bearer', 'scope': 'read write', 'refresh
_token': 'gWPl1lz7JyryaLUp44yaoaLkiUtQas'}
```

mary

```
mary_response = requests.post(registration_url, data = mary)

mary_response = mary_response.json()

print(mary_response)
```

```
{'access_token': 'EZ1lcuUwhA3f0HL9xtyNocoucgjEbJ', 'expires_in': 36000, 'token_type': 'Bearer', 'scope': 'read write', 'refresh
_token': 'igtUZHpvbD8tKkVxLvAR6DEHP6szOa'}
```

nestor

```
nestor_response = requests.post(registration_url, data = nestor)

nestor_response = nestor_response.json()

print(nestor_response)
```

```
{'access_token': 'pMqczR2MZ0bVyqlcdXcGPslaQ1IB2t', 'expires_in': 36000, 'token_type': 'Bearer', 'scope': 'read write', 'refresh
_token': 'MqwgdsyMQnLDboCGaclUNjLWPOqMtm'}
```

TC 2. olga, nick, mary, and nestor use the oAuth v2 authorisation service to register and get their tokens.

Create authorisation token url

```
token_url = "http://10.61.64.70:8000/authentication/token/"
```

olga

```
requests.post(registration_url, data = olga) # Sending request to register username and password data
olga_token_response = requests.post(token_url, data = olga).json() # sending request to recieve token
olga_token = olga_token_response["access_token"]

print(olga_token)
```

```
94M6j2nxz2w8Oi2AAVJ0bbm6lcVzUC
```

nick

```
requests.post(registration_url, data = nick) # Sending request to register username and password data
nick_token_response = requests.post(token_url, data = nick).json() # sending request to recieve token
nick_token = nick_token_response["access_token"]

print(nick_token)
```

```
VurhLbzxQcX95K3aqkYl2ERFOty7Ge
```

mary

```
requests.post(registration_url, data = mary) # Sending request to register username and password data
mary_token_response = requests.post(token_url, data = mary).json() # sending request to recieve token
mary_token = mary_token_response["access_token"]

print(mary_token)
```

```
2bzrTsZJOTPVLh7KGDer3DRPy18GQx
```

nestor

```
requests.post(registration_url, data = nestor) # Sending request to register username and password data
nestor_token_response = requests.post(token_url, data = nestor).json() # sending request to recieve token
nestor_token = nestor_token_response["access_token"]

print(nestor_token)
```

```
b5E4h9sLAeIAqDdKE8nrv6gezSDf6r
```

TC 3. olga makes a call to the API without using her token. This call should be unsuccessful as the user is unauthorised.

```
posts_url = "http://10.61.64.70:8000/posts/"

olga_get = requests.post(posts_url)
olga_get.json()
```

```
{'detail': 'Authentication credentials were not provided.'}
```

TC 4. olga posts a message in the Tech topic with an expiration time (e.g. 5 minutes) using her token. After the end of the expiration time, the message will not accept any further user interactions (likes, dislikes, or comments).

```
olga_headers = {'Authorization': "Bearer " + str(olga_token)}

import datetime
T = datetime.timedelta
olga_post_1 = requests.post(posts_url, headers = olga_headers, data = {"Topics": "Tech",
                                                                       "Title" : "Elon Musk should get a brain implant: here's why.",
                                                                       "MessageBody": "Elon is endorsing Neuralink. "
                                                                             "He needs to show that he is not a skeptic",
                                                                       "ActiveTime" : T(hours = 1)}).json()
print(olga_post_1)
```

```
{'PostID': 85, 'Post_owner': 'Olga', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.495989Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.495407Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 5. nick posts a message in the Tech topic with an expiration time using his token.

```
nick_headers = {'Authorization': "Bearer " + str(nick_token)}



nick_post_1 = requests.post(posts_url, headers = nick_headers, data = {"Topics" : "Tech",
                                                                       "Title" : "Flying cars? Still viable.",
                                                                       "MessageBody": "Flying cars should be an industry. "
                                                                             "It is 100% possible",
                                                                       "ActiveTime" : T(hours = 1)}).json()
print(nick_post_1)
```

```
{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 0, 'total_likes': 0, 'likes': [], 'total_dislikes': 0, 'dislikes': [],
'total_comments': 0, 'comments': []}
```

TC 6. mary posts a message in the Tech topic with an expiration time using her token.

```
mary_headers = {'Authorization': "Bearer " + str(mary_token)}



mary_post_1 = requests.post(posts_url, headers = mary_headers, data = {"Topics" : "Tech",
                                                                       "Title" : "Elon Musk should get a brain implant: here's why.",
                                                                       "MessageBody": "Elon is endorsing Neuralink. "
                                                                             "He needs to show that he is not a skeptic",
                                                                       "ActiveTime" : T(hours = 1)}).json()
print(mary_post_1)
```

```
{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 7. nick and olga browse all the available posts in the Tech topic, there should be three posts available with zero likes, zero dislikes and without any comments. ¶

```
available_tech_posts_url = "http://10.61.64.70:8000/posts/?Topics=Tech&is_expired__exact=False"

nick_search_1 = requests.get(available_tech_posts_url, headers = nick_headers).json()

print("Nick's view of tech topics", *nick_search_1, sep = "\n")

olga_search_1 = requests.get(available_tech_posts_url, headers = olga_headers).json()

print("\n","Olga's view of tech topics", *olga_search_1, sep = "\n")
```

```
Nick's view of tech topics
{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 0, 'total_likes': 0, 'likes': [], 'total_dislikes': 0, 'dislikes': [],
'total_comments': 0, 'comments': []}
{'PostID': 85, 'Post_owner': 'Olga', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.495989Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.495407Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}


Olga's view of tech topics
{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 0, 'total_likes': 0, 'likes': [], 'total_dislikes': 0, 'dislikes': [],
'total_comments': 0, 'comments': []}
{'PostID': 85, 'Post_owner': 'Olga', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.495989Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.495407Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 8. nick and olga “like” mary’s post in the Tech topic.

```
mary_post_1_url = "http://10.61.64.70:8000/posts/" + str(mary_post_1["PostID"]) + "/"
add_like = "add_like/"

requests.post(mary_post_1_url+add_like, headers = nick_headers) # nick likes mary's post

requests.post(mary_post_1_url+add_like, headers = olga_headers) # olga likes mary's post

mary_post_1 = requests.get(mary_post_1_url, headers = olga_headers).json()

print(mary_post_1)
```

```
{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 2, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 9. nestor “likes” nick’s post, and “dislikes” mary’s post in the Tech topic.

```
nestor_headers = {'Authorization': "Bearer " + str(nestor_token)}

nick_post_1_url = "http://10.61.64.70:8000/posts/" + str(nick_post_1["PostID"]) + "/"
add_dislike = "add_dislike/"

requests.post(nick_post_1_url + add_like, headers = nestor_headers)

requests.post(mary_post_1_url + add_dislike, headers = nestor_headers)

nick_post_1 = requests.get(nick_post_1_url, headers = nestor_headers).json()

mary_post_1 = requests.get(mary_post_1_url, headers = nestor_headers).json()

print(mary_post_1)

print(nick_post_1)
```

```
{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 3, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 1, 'dislikes': ['Nestor'], 'total_comments': 0, 'comments': []}
{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 1, 'total_likes': 1, 'likes': ['Nestor'], 'total_dislikes': 0, 'dislik
es': [], 'total_comments': 0, 'comments': []}
```

TC 10. nick browses all the available posts in the Tech topic; at this stage he can see the number of likes and dislikes for each post (mary has 2 likes and 1 dislike and nick has 1 like). There are no comments made yet.

```
available_tech_posts_url = "http://10.61.64.70:8000/posts/?Topics=Tech&is_expired__exact=False"

print("Nick's view of tech topics", *requests.get(available_tech_posts_url, headers = nick_headers).json(), sep = "\n")
```

```
Nick's view of tech topics
{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 3, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 1, 'dislikes': ['Nestor'], 'total_comments': 0, 'comments': []}
{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 1, 'total_likes': 1, 'likes': ['Nestor'], 'total_dislikes': 0, 'dislik
es': [], 'total_comments': 0, 'comments': []}
{'PostID': 85, 'Post_owner': 'Olga', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.495989Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.495407Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 11. mary likes her post in the Tech topic. This call should be unsuccessful, as in Piazza a post owner cannot like their own messages.

```
print("Mary's post before she attempts to like her own post:", "\n",
      requests.get(mary_post_1_url, headers = mary_headers).json(), "\n")

print("Response following Mary's attempt to like her own post", "\n",
      requests.post(mary_post_1_url + add_like, headers = mary_headers).json(), "\n")
```

```
Mary's post before she attempts to like her own post:
 {'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Messag
eBody': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z',
'ActiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 3, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 1, 'dislikes': ['Nestor'], 'total_comments': 0, 'comments': []}

Response following Mary's attempt to like her own post
 You cannot like/dislike your own post
```

TC 12. nick and olga comment for mary's post in the Tech topic in a round-robin fashion (one after the other adding at least 2 comments each).

```
add_comment = "add_comment/"

print("Nick comment 1:","\n",
      requests.post(mary_post_1_url + add_comment,
                    headers = nick_headers,
                    data = {"MessageBody": "Totally agree! Why has nestor diskliked this?"}).json())

print("Olga comment 1:", "\n",
      requests.post(mary_post_1_url + add_comment,
                    headers = olga_headers,
                    data = {"MessageBody": "Backwards mentality. Ignore the dislikes Nick, "
                           "I'm for it. How is your little child, Nick?"}).json())

print("Nick comment 2:", "\n",
      requests.post(mary_post_1_url + add_comment,
                    headers = nick_headers,
                    data = {"MessageBody": "X-delta-T is doing well, what a delight she is! "
                            "she said her first work last week! How's your litte one doing, Olga?"}).json())

print("Olga comment 2:", "\n",
      requests.post(mary_post_1_url + add_comment,
                    headers = olga_headers,
                    data = {"MessageBody": "Backwards mentality. Ignore the dislikes Nick, "
                           "I'm for it. How is your little child,?"}).json())

mary_post_1 = requests.get(mary_post_1_url, headers = nestor_headers).json()

print("\n", "Mary's resulting post: ", "\n", mary_post_1)
```

```
Nick comment 1:
 {'id': 79, 'User_commented': 'Nick', 'MessageBody': 'Totally agree! Why has nestor diskliked this?', 'Timestamp': '2021-04-05T
05:57:17.371646Z'}
Olga comment 1:
 {'id': 80, 'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it. How is your li
ttle child, Nick?", 'Timestamp': '2021-04-05T05:57:17.473657Z'}
Nick comment 2:
 {'id': 81, 'User_commented': 'Nick', 'MessageBody': "X-delta-T is doing well, what a delight she is! she said her first work l
ast week! How's your litte one doing, Olga?", 'Timestamp': '2021-04-05T05:57:17.578003Z'}
Olga comment 2:
 {'id': 82, 'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it. How is your li
ttle child,?", 'Timestamp': '2021-04-05T05:57:17.683464Z'}

 Mary's resulting post:
 {'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Messag
eBody': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z',
'ActiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 3, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 1, 'dislikes': ['Nestor'], 'total_comments': 4, 'comments': [{'id': 79, 'User_commented':
'Nick', 'MessageBody': 'Totally agree! Why has nestor diskliked this?', 'Timestamp': '2021-04-05T05:57:17.371646Z'}, {'id': 80,
'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it. How is your little child,
Nick?", 'Timestamp': '2021-04-05T05:57:17.473657Z'}, {'id': 81, 'User_commented': 'Nick', 'MessageBody': "X-delta-T is doing we
ll, what a delight she is! she said her first work last week! How's your litte one doing, Olga?", 'Timestamp': '2021-04-05T05:5
7:17.578003Z'}, {'id': 82, 'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it.
How is your little child,?", 'Timestamp': '2021-04-05T05:57:17.683464Z'}]}
```

TC 13. nick browses all the available posts in the Tech topic; at this stage he can see the number of likes and dislikes of each post and the comments made.

```
print("Nick's view of tech topics", *requests.get(available_tech_posts_url, headers = nick_headers).json(), sep = "\n\n")
```

```
Nick's view of tech topics

{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 3, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 1, 'dislikes': ['Nestor'], 'total_comments': 4, 'comments': [{'id': 79, 'User_commented':
'Nick', 'MessageBody': 'Totally agree! Why has nestor diskliked this?', 'Timestamp': '2021-04-05T05:57:17.371646Z'}, {'id': 80,
'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it. How is your little child,
Nick?", 'Timestamp': '2021-04-05T05:57:17.473657Z'}, {'id': 81, 'User_commented': 'Nick', 'MessageBody': "X-delta-T is doing we
ll, what a delight she is! she said her first work last week! How's your litte one doing, Olga?", 'Timestamp': '2021-04-05T05:5
7:17.578003Z'}, {'id': 82, 'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it.
How is your little child,?", 'Timestamp': '2021-04-05T05:57:17.683464Z'}]}

{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 1, 'total_likes': 1, 'likes': ['Nestor'], 'total_dislikes': 0, 'dislik
es': [], 'total_comments': 0, 'comments': []}

{'PostID': 85, 'Post_owner': 'Olga', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.495989Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.495407Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 14. nestor posts a message in the Health topic with an expiration time using her token.

```
nestor_post_1 = requests.post(posts_url,
                              headers = nestor_headers,
                              data = {"Topics" : "Health",
                                      "Title" : "Want to lose 10 pounds in 5 days? Heres how.",
                                      "MessageBody": "The answer is simple. All you need is lemon juice and cockroaches. "
                                      "It is 100% possible",
                                      "ActiveTime" : T(minutes = 1)}).json()

print(nestor_post_1)
```

```
{'PostID': 88, 'Post_owner': 'Nestor', 'Topics': ['Health'], 'Title': 'Want to lose 10 pounds in 5 days? Heres how.', 'MessageB
ody': 'The answer is simple. All you need is lemon juice and cockroaches. It is 100% possible', 'Timestamp': '2021-04-05T05:57:
17.958534Z', 'ActiveTime': '60.0', 'Post_expiration_time': '2021-04-05T05:58:17.958256Z', 'total_reactions': 0, 'total_likes':
0, 'likes': [], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 15. mary browses all the available posts in the Health topic; at this stage she can see only nestor's post.

```
available_health_posts_url = "http://10.61.64.70:8000/posts/?Topics=Health&is_expired__exact=False"
print("Mary's view of health topics", *requests.get(available_health_posts_url, headers = mary_headers).json(), sep = "\n")
```

```
Mary's view of health topics
{'PostID': 88, 'Post_owner': 'Nestor', 'Topics': ['Health'], 'Title': 'Want to lose 10 pounds in 5 days? Heres how.', 'MessageB
ody': 'The answer is simple. All you need is lemon juice and cockroaches. It is 100% possible', 'Timestamp': '2021-04-05T05:57:
17.958534Z', 'ActiveTime': '60.0', 'Post_expiration_time': '2021-04-05T05:58:17.958256Z', 'total_reactions': 0, 'total_likes':
0, 'likes': [], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

TC 16. mary posts a comment in the nestor's message in the Health topic.

```
nestor_post_1_url = "http://10.61.64.70:8000/posts/" + str(nestor_post_1["PostID"]) + "/"

print("Mary's comment on nestor's message:", "\n",
      requests.post(nestor_post_1_url + add_comment,
                    headers = mary_headers, data = {"MessageBody": "What a load of nonsense"}).json())

print(requests.get(nestor_post_1_url,
                    headers = mary_headers).json())
```

```
Mary's comment on nestor's message:
 {'id': 83, 'User_commented': 'Mary', 'MessageBody': 'What a load of nonsense', 'Timestamp': '2021-04-05T05:57:18.226983Z'}
{'PostID': 88, 'Post_owner': 'Nestor', 'Topics': ['Health'], 'Title': 'Want to lose 10 pounds in 5 days? Heres how.', 'MessageB
ody': 'The answer is simple. All you need is lemon juice and cockroaches. It is 100% possible', 'Timestamp': '2021-04-05T05:57:
17.958534Z', 'ActiveTime': '60.0', 'Post_expiration_time': '2021-04-05T05:58:17.958256Z', 'total_reactions': 0, 'total_likes':
0, 'likes': [], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 1, 'comments': [{'id': 83, 'User_commented': 'Mary', 'Me
ssageBody': 'What a load of nonsense', 'Timestamp': '2021-04-05T05:57:18.226983Z'}]}
```

TC 17. mary dislikes nestor's message in the Health topic after the end of post expiration time. This should fail.

```
import time
time.sleep(60)

print(requests.post(nestor_post_1_url + add_like, headers = mary_headers).json(), "\n")

print(requests.get(nestor_post_1_url, headers = mary_headers).json(), "\n")
```

```
This post has expired. You cannot interact with this post.

{'PostID': 88, 'Post_owner': 'Nestor', 'Topics': ['Health'], 'Title': 'Want to lose 10 pounds in 5 days? Heres how.', 'MessageB
ody': 'The answer is simple. All you need is lemon juice and cockroaches. It is 100% possible', 'Timestamp': '2021-04-05T05:57:
17.958534Z', 'ActiveTime': '60.0', 'Post_expiration_time': '2021-04-05T05:58:17.958256Z', 'total_reactions': 0, 'total_likes':
0, 'likes': [], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 1, 'comments': [{'id': 83, 'User_commented': 'Mary', 'Me
ssageBody': 'What a load of nonsense', 'Timestamp': '2021-04-05T05:57:18.226983Z'}]}
```

TC 18. nestor browses all the messages in the Health topic. There should be only post (his own) with one comment (mary's).

```
all_health_posts_url = "http://10.61.64.70:8000/posts/?Topics=Health"

print("Nestor's view of all health posts:" , "\n",
      requests.get(all_health_posts_url, headers = mary_headers).json(), "\n")
```

```
Nestor's view of all health posts:
 [{'PostID': 88, 'Post_owner': 'Nestor', 'Topics': ['Health'], 'Title': 'Want to lose 10 pounds in 5 days? Heres how.', 'Messag
eBody': 'The answer is simple. All you need is lemon juice and cockroaches. It is 100% possible', 'Timestamp': '2021-04-05T05:5
7:17.958534Z', 'ActiveTime': '60.0', 'Post_expiration_time': '2021-04-05T05:58:17.958256Z', 'total_reactions': 0, 'total_like
s': 0, 'likes': [], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 1, 'comments': [{'id': 83, 'User_commented': 'Mary',
'MessageBody': 'What a load of nonsense', 'Timestamp': '2021-04-05T05:57:18.226983Z'}]}]
```

TC 19. nick browses all the expired messages in the Sport topic. These should be empty.

```
expired_sport_posts_url = "http://10.61.64.70:8000/posts/?Topics=Sport&is_expired__exact=True"

print("Nick's view of all expired sport posts:" , "\n",
      *requests.get(expired_sport_posts_url, headers = nick_headers).json(), "\n")
```

```
Nick's view of all expired sport posts:
```

TC 20. nestor queries for an active post having the highest interest (maximum sum of likes and dislikes) in the Tech topic. This should be mary's post.

```
tech_posts_most_popular_url = "http://10.61.64.70:8000/posts/?Topics=Tech&popularity=-total_reactions&is_expired__exact=False"

print("Nestors's view of all active tech posts sorted by popularity in descending order:" , "\n",
      *requests.get(tech_posts_most_popular_url, headers = nestor_headers).json(), sep = "\n")
```

```
Nestors's view of all active tech posts sorted by popularity in descending order:


{'PostID': 87, 'Post_owner': 'Mary', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.795017Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.794756Z', 'total_reactions': 3, 'total_likes': 2, 'likes':
['Nick', 'Olga'], 'total_dislikes': 1, 'dislikes': ['Nestor'], 'total_comments': 4, 'comments': [{'id': 79, 'User_commented':
'Nick', 'MessageBody': 'Totally agree! Why has nestor diskliked this?', 'Timestamp': '2021-04-05T05:57:17.371646Z'}, {'id': 80,
'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it. How is your little child,
Nick?", 'Timestamp': '2021-04-05T05:57:17.473657Z'}, {'id': 81, 'User_commented': 'Nick', 'MessageBody': "X-delta-T is doing we
ll, what a delight she is! she said her first work last week! How's your litte one doing, Olga?", 'Timestamp': '2021-04-05T05:5
7:17.578003Z'}, {'id': 82, 'User_commented': 'Olga', 'MessageBody': "Backwards mentality. Ignore the dislikes Nick, I'm for it.
How is your little child,?", 'Timestamp': '2021-04-05T05:57:17.683464Z'}]}
{'PostID': 86, 'Post_owner': 'Nick', 'Topics': ['Tech'], 'Title': 'Flying cars? Still viable.', 'MessageBody': 'Flying cars sho
uld be an industry. It is 100% possible', 'Timestamp': '2021-04-05T05:57:15.662487Z', 'ActiveTime': '3600.0', 'Post_expiration_
time': '2021-04-05T06:57:15.662216Z', 'total_reactions': 1, 'total_likes': 1, 'likes': ['Nestor'], 'total_dislikes': 0, 'dislik
es': [], 'total_comments': 0, 'comments': []}
{'PostID': 85, 'Post_owner': 'Olga', 'Topics': ['Tech'], 'Title': "Elon Musk should get a brain implant: here's why.", 'Message
Body': 'Elon is endorsing Neuralink. He needs to show that he is not a skeptic', 'Timestamp': '2021-04-05T05:57:15.495989Z', 'A
ctiveTime': '3600.0', 'Post_expiration_time': '2021-04-05T06:57:15.495407Z', 'total_reactions': 0, 'total_likes': 0, 'likes':
[], 'total_dislikes': 0, 'dislikes': [], 'total_comments': 0, 'comments': []}
```

7 DATABASE TABLES

This section describes the tables created by the testing application. users_User contains the admin user information, which was created prior to the testing application.

7.1 USERS_USER

id	password	last_login	is_superus	username	first_name	last_name	email	is_staff	is_active	date_joined	access_tol	refresh_to
1	pbkdf2_sh	18:14.0	1	student				1	1	05/04/2021	1	1
6	pbkdf2_sha256$1800		0	Olga				0	1	05/04/2021	RH1iE8zLB	kR8WwRJe
7	pbkdf2_sha256$1800		0	Nick				0	1	05/04/2021	FJ4NnJUY2	guzU6rNzL
8	pbkdf2_sha256$1800		0	Mary				0	1	05/04/2021	FFlHJaDPzl	8mMAGpy
9	pbkdf2_sha256$1800		0	Nestor				0	1	05/04/2021	93A2rezlJl	XiZXpVKPt

7.2 POSTS_POST

PostID	Timestamp	Title	MessageBody	ActiveTime	Post_expiration_time	Post_owner_id
4	05/04/2021 06:28:30	Elon Musk should get a brain impla	Elon is endorsing Neuralink. He needs to show that he is n	3600000000	05/04/2021 07:28:30	Olga
5	05/04/2021 06:28:30	Flying cars? Still viable.	Flying cars should be an industry. It is 100% possible	3600000000	05/04/2021 07:28:30	Nick
6	05/04/2021 06:28:30	Elon Musk should get a brain impla	Elon is endorsing Neuralink. He needs to show that he is n	3600000000	05/04/2021 07:28:30	Mary
7	05/04/2021 06:28:32	Want to lose 10 pounds in 5 days? I	The answer is simple. All you need is lemon juice and cock	60000000	05/04/2021 06:29:32	Nestor

7.3 POSTS_POST_TOPICS

id	post_id	topic_id
4	4	1
5	5	1
6	6	1
7	7	2

7.4 POSTS_COMMENT

id	MessageBody	Timestamp	Post_com	User_com	Post_time_remaining	
1	Totally agree! Why has nestor diskliked this?	05/04/2021 06:28:31	6	Nick	3.6E+09	
2	Backwards mentality. Ignore the dislikes Nick, I'm for it. How is yo	05/04/2021 06:28:31	6	Olga	3.6E+09	
3	X-delta-T is doing well, what a delight she is! she said her first wor	05/04/2021 06:28:31	6	Nick	3.6E+09	
4	Backwards mentality. Ignore the dislikes Nick, I'm for it. How is yo	05/04/2021 06:28:32	6	Olga	3.6E+09	
5	What a load of nonsense	05/04/2021 06:28:32	7	Mary	59765253	

7.5 POSTS_LIKE

id	Timestamp	Post_liked	User_liked	Post_time_remaining	
3	05/04/2021 06:28:30	6	7	3.6E+09	
4	05/04/2021 06:28:30	6	6	3.6E+09	
5	05/04/2021 06:28:30	5	9	3.6E+09	

7.6 POSTS_DISLIKE

id	Timestamp	Post_dislik	User_dislik	Post_time_remaining	
1	05/04/2021 06:28:31	6	9	3.6E+09	

REFERENCES

[1] Django Software Foundation, "Django: The web framework for perfectionists with deadlines.," 2021. [Online]. Available: https://www.djangoproject.com/. [Accessed 05 April 2021].

[2] A. Gaynor, C. Gibson and others, "django-filter," 2020. [Online]. Available: django-filter.readthedocs.io/en/stable/. [Accessed 5 April 2021].

[3] Jazzband, "Django OAuth Toolkit Documentation," [Online]. Available: https://django-oauth-toolkit.readthedocs.io/en/latest/. [Accessed 5 April 2021].

[4] pypi, "django-property-filter," 2021. [Online]. Available: https://pypi.org/project/django-property-filter/. [Accessed 5 April 2021].

[5] pypi, "djangorestframework," 2021. [Online]. Available: https://pypi.org/project/djangorestframework/. [Accessed 5 April 2021].

[6] Internet Engineering Task Force, "rfc6749: The OAuth 2.0 Authorization Framework," October 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Accessed 05 April 2021].

[7] Django Software Foundation, "django.contrib.auth," 2021. [Online]. Available: https://docs.djangoproject.com/en/3.1/ref/contrib/auth/. [Accessed 5 April 2021].

[8] geeksforgeeks, "MultipleChoiceField - Django Forms," 13 February 2020. [Online]. Available: https://www.geeksforgeeks.org/multiplechoicefield-django-forms/. [Accessed 5 April 2021].

[9] E. Ziethen, "Limitations - Django Property Filter," 2020. [Online]. Available: https://django-property-filter.readthedocs.io/en/latest/guide/limitations.html. [Accessed 05 April 2021].

[10] L. Richardson and S. Ruby, RESTful Web Services, O'Reilly Media, Inc., 2007.

[11] restfulapi.net , "HATEOAS Driven REST APIs," 2020. [Online]. Available: https://restfulapi.net/hateoas/. [Accessed 5 April 2021].