Cloud Computing Coursework
Laurence Tickell
MSC Data Science
12833146

VM IP: 10.61.64.76
HTML Output: http://10.61.64.76:8000/viewstatus/

Endpoints:

http://10.61.64.76:8000/v1/status/
http://10.61.64.76:8000/v1/reaction/
http://10.61.64.76:8000/v1/user/
http://10.61.64.76:8000/v1/like/
http://10.61.64.76:8000/v1/dislike/
http://10.61.64.76:8000/v1/datadump/
http://10.61.64.76:8000/v1/StatusCommentsAndLikes/

The endpoints are protected unless logged in as an admin - Here is a login
Username: **MarkerAdmin**
Password: **SecondSuperUser**

Which can be entered here
http://10.61.64.76:8000/admin/login/?next=/admin/

16 February 2021 - before stating
Plan: The Coursework phases seems a sensible way to approach the coursework. The early phases of authorisation and installation are quite similar to the labs, whereas the later phases use the labs as a starting point but require a significantly more complex functionality to be added.

I want to get the early parts working correctly then I can start to add methods to the working classes one by one. This way I can test each one. I will probably want to make the functions for the testing application one by one too so I can test each action and then join them together at the end in the required order. This will make debugging easier.

Finally I will try and use some HTML and CSS as was detailed in the first lab to make a simple front end. I have used both before but not for a long while so I should be able to do it. However I can see almost all the marks are connected to the backend

development so I will do no front end tasks until I think I can make no simple improvements to the backend.

Phase 1

In the first phase I want to create a virtual environment called Piazza and install Django. I will follow the instructions from lab 4 as this should install all the packages I need for phase 2, Django, Rest framework and oAuth toolkit.

```
(piazza) student@cc:~/piazza$ tree src/
src/
├── manage.py
└── piazza
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

1 directory, 6 files
```

Once activated we can see the environment active below

```
Last login: Sun Apr 11 10:18:50 2021 from 10.61.210.10
student@cc:~$ cd piazza
student@cc:~/piazza$ source bin/activate
(piazza) student@cc:~/piazza$ cd src
(piazza) student@cc:~/piazza/src$ python3 manage.py runserver 10.61.64.76:8000
Watching for file changes with StatReloader
Performing system checks...
```

Phase 2

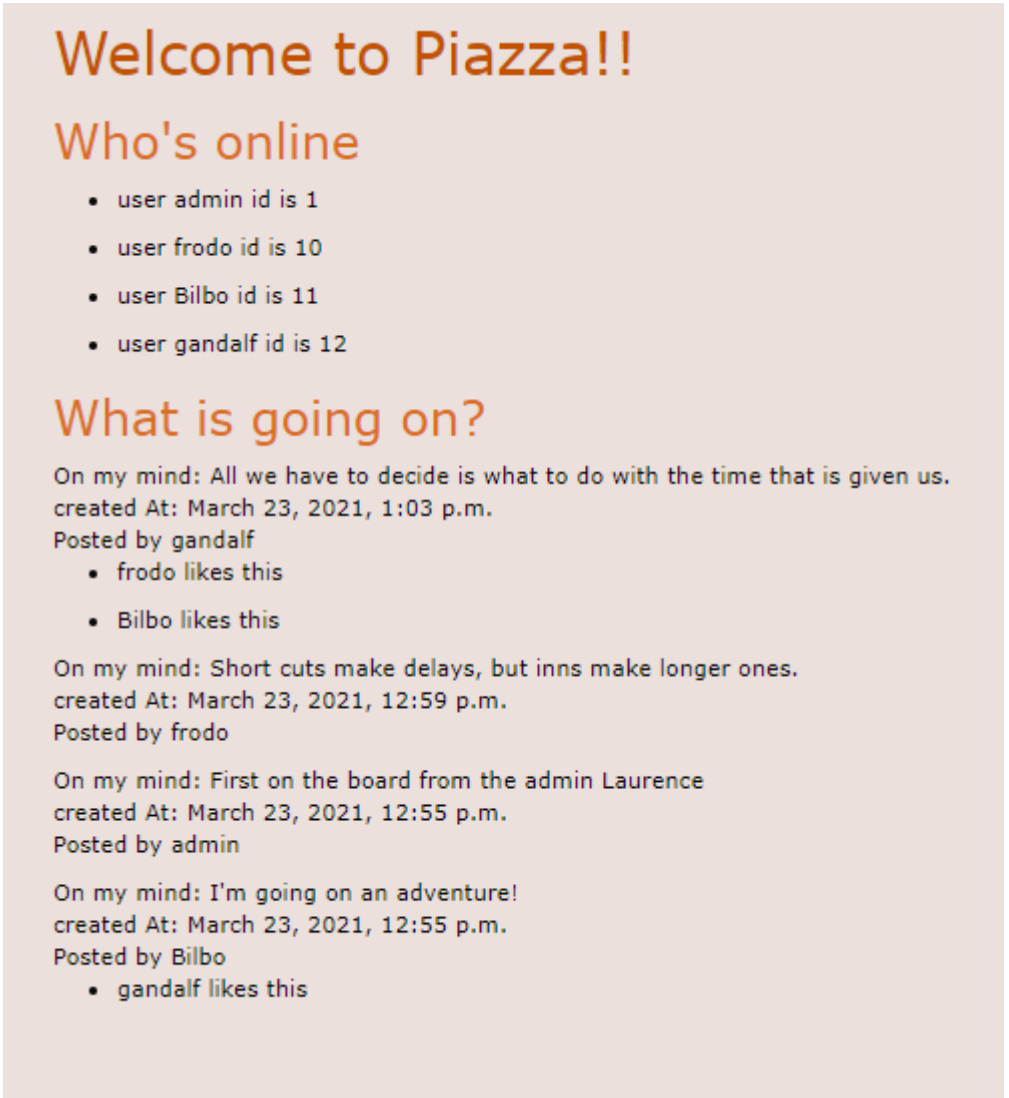
I will continue to follow the instructions for lab 4, except instead of a movie database I will use posts. To start with the equivalent of a status will be the only field. I will expand the other functionality such as poster, subject and timestamps and liking once I have this basic process and a simple test script working with the tokens.

Once my status class was up and running, I added some more fields with some stackoverflow help. I added timestamps and the author. Some of these may need to be adjusted later but I will need to interact with these later to fulfil aspects of the applications requirements.

I then created a reaction model so people can like and comment on statuses. This was set up as a Forgein Key Relationship. At this stage the model was responsible for comments likes and dislikes, although this was changed later as I detail in other sections of this report. At this point I could not work out how to stop people liking the same status multiple times.

I then set up so test scripts can alter the database and add users. These worked but an issue was that a user can effectively post on behalf of any user. A post requires a PK. At this point I was hoping to solve this problem. I could have constrained the test scripts by having the token return the PK. This would have constrained the test scrip but not secured the system in any way. In the end I was not able to solve this problem.

I created a basic front end, which is more of a database dump, making it easier to see how the data looks and see problems with the database.

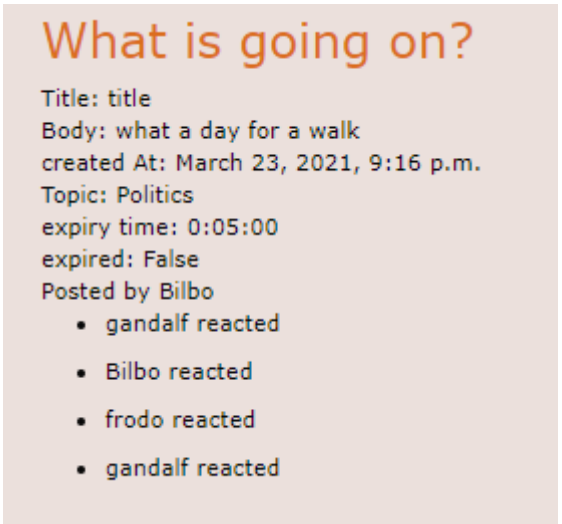


From this stage I needed to expand the status mode to fulfil the coursework requirements. The needed fields are

Title: This should be a shorter Charfield I will start with 100
Topic: This is a choicefield. Best practice is to use a single letter to save database space. I had problems rendering this on my webpage so gave made the value and label.
Expiration Time: a DurationField seems most suitable here
Expired or live: I used a boolean defaulted to False for live. I then wanted to add a method to the class that uses the duration time and date stamp to update to 1 once the expiration time had passed.

I then need to look at how best to organise the reactions for comments likes and dislikes. The other table is the user table which links to all the others. It works at present with oAuth, so I do not want to adjust anything that could affect that, but there may be some more advanced functionality that could be added to Stelios' implementation from the labs.

This was an example with Likes comments and dislikes being outputted on the front end as reactions, as I knew the design would change later.



Final Database design

After running through the test cases and adjusting my models according to problems and failures I came up with the following structure.

Users Table (which is linked to Authentication).

The PK is generated when a user is made and this acts as a FK on all the other tables. Every action in Piazza is attributed to a person. If I had more time I would have liked to have a login system and then added permission classes so that only the logged in user or admin could make posts in that person's name.

Status Table

This has the title, topic and the dates the status is created and updated. It also has an expiry date. I removed the live/expired boolean as I found ways to calculate this when needed. This removed the need to save redundant or out of date information. Unless I was going to have an ongoing While Loop I needed an event (most likely save or POST) for the date to be checked. However this does cause some issues which I will detail during the test cases.

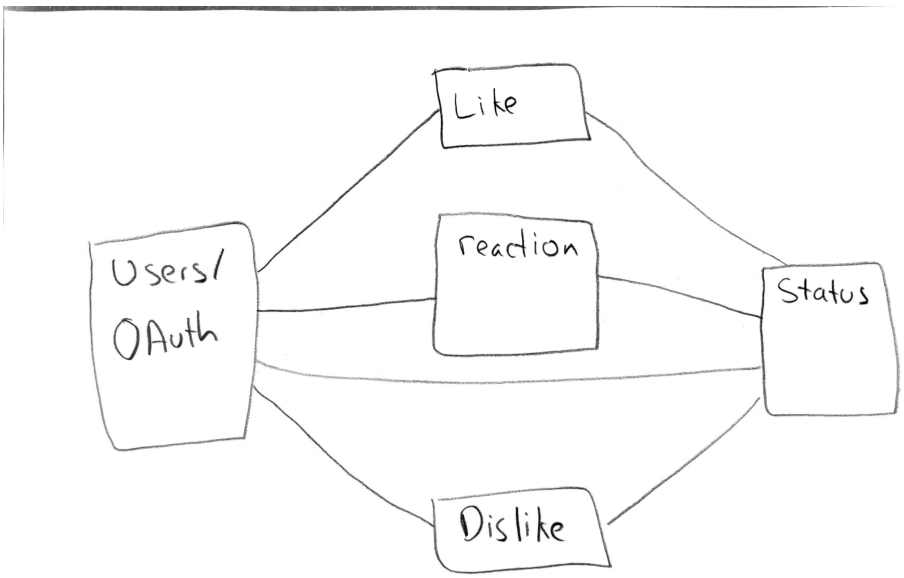
Reaction Table

This is when a user comments and is a Forgein Key table linked to status (and user). We want people to comment as many times as allowed when the post is live. I have overridden the save method to prevent people commenting on expired posts, but this has some ugly consequences. If I had been able to implement this through validators the error handling would have been much prettier.

Like and Dislike tables

These are two tables for like and dislikes. As mentioned earlier these started as many to many tables. I then made them a one to many relationship after reading a blog (medium,1). This worked the best of configurations I tried, but I could not prevent users liking their own post. This may be why facebook and twitter allow people to like their own posts.

Instead I went back to using Forgein key many to many, and prevented people liking their own post, expired posts and liking twice by overriding the save method. As with the reaction table this works but is ugly.



Endpoints

My main decision was having just one for POST that would include Status, Like and Reaction. This would have meant less endpoints, but a much messier POST. The JSON would have had to be heavily nested and it would have involved using PUT and Patch when in reality I was only updating one of the underlying SQL tables.
My final endpoints look like this, with specific links for POSTing each time and special endpoints for GET requests that included calculated fields such as number of likes that are not in the database but can easily calculated at the time of the GET request

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "status": "http://10.61.64.76:8000/v1/status/",
    "reaction": "http://10.61.64.76:8000/v1/reaction/",
    "user": "http://10.61.64.76:8000/v1/user/",
    "like": "http://10.61.64.76:8000/v1/like/",
    "dislike": "http://10.61.64.76:8000/v1/dislike/",
    "datadump": "http://10.61.64.76:8000/v1/datadump/",
    "StatusCommentsAndLikes": "http://10.61.64.76:8000/v1/StatusCommentsAndLikes/"
}
```

http://10.61.64.76:8000/v1/status/

This is intended for making a status a number after the link will show a status individually and allow puts if it corresponds to the PK of a status in the database

http://10.61.64.76:8000/v1/reaction/

This is for making comments on statuses. The parent status is not visible so it is designed for Post and Put requests but not useful for GET requests.

http://10.61.64.76:8000/v1/user/

This shows active user accounts and their PK. In a real application this would not be visible and would be replaced by a login system, but I need to to know which PK to use in my testing code

http://10.61.64.76:8000/v1/like/
http://10.61.64.76:8000/v1/dislike/

These are intended for POST or PUT on likes and dislikes. Like with the reaction endpoint they do not display the parent status or username, just FKs, so are not useful for GET requests

http://10.61.64.76:8000/v1/datadump/

This shows the whole database, with all the attributes of every table. I mainly used this to help me visualise when I was accessing attributes of a parent or child objects. It would be removed from view in any actual application.

http://10.61.64.76:8000/v1/StatusCommentsAndLikes/

This endpoint is for GET requests. It calls Status as well as their comments and various calculated fields based on @properties. These include expired, total likes and disliked. It can also be ordered by any database fields and uses topic as a search term, such as:

http://10.61.64.76:8000/v1/StatusCommentsAndLikes/?search=Tech

Further field could be added to the search functionality but adding the field to this line

```python
#These view determine what appears at the endpoints, who can view them (basically only admin)
class StatusViewSet(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]
    queryset = Status.objects.all()
    serializer_class = StatusSerializer
    filter_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend]
    search_fields = ['Topic', 'id']
```

However I left the minimum needed. Clashes with fields can cause unexpected behaviour and should be added after more testing. I noticed a similar oversight while using the morrisons website recently. I entered a search term in the food search that was also the name of a store and it took me to the store location page instead of saying no food item exists. Such clashes must be hard to remove once embedded in the system.

Test Script.

TC1 and TC2, the four users register and get tokens.
I just adapted the lab code here, here is the first return as JSON

```
print(olga_token)
```

```
{'access_token': 'ccYVGrxFhE0k6oef5yu6tWe8KtLBF6', 'expires_in': 36000, 'token_type': 'Bearer', 'scope': 'read write', 'refresh
_token': 'pS4tOi6TcIwDDhfyySt7QQFFZ2gpV2'}
ccYVGrxFhE0k6oef5yu6tWe8KtLBF6
```

The other four are added and their accounts can be see in admin or at the user endpoint, http://10.61.64.76:8000/v1/user/

TC. 3 Olga makes a call without her token which should fail

The response is

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

TC4 Olga posts a message to the Tech Topic with an expiration time
The response is
{'Title': 'Which Iphone', 'status': 'Got the new Iphone', 'author': 33, 'Topic': 'Tech', 'Expiration_Time': '00:05:00'}

And we can see in the status view area

http://10.61.64.76:8000/v1/StatusCommentsAndLikes/


```
Vary: Accept

[
    {
        "id": 77,
        "Title": "Which Iphone",
        "status": "Got the new Iphone",
        "Topic": "Tech",
        "author_username": "olga",
        "get_total_likes": 0,
        "get_total_dis_likes": 0,
        "overall_rating": 0,
        "created_at": "2021-04-04T17:26:59.533845Z",
        "Expiration_Time": "00:55:00",
        "Poststatus": [],
        "update_expired": false,
        "get_total_comments": 0
    }
]
```


TC5 Nick posts a message to the Tech Topic with an expiration time

As with Olga's post


```
{
    "id": 78,
    "Title": "Which android",
    "status": "Got the new 5G goglephone",
    "Topic": "Tech",
    "author_username": "nick",
    "get_total_likes": 0,
    "get_total_dis_likes": 0,
    "overall_rating": 0,
    "created_at": "2021-04-04T17:38:07.002100Z",
    "Expiration_Time": "00:35:00",
    "Poststatus": [],
    "update_expired": false,
    "get_total_comments": 0
}
```

TC6 Mary Posts in the Tech Topic


```
    {
        "id": 79,
        "Title": "Opera",
        "status": "has anyone tried opera as a web browser",
        "Topic": "Tech",
        "author_username": "mary",
        "get_total_likes": 0,
        "get_total_dis_likes": 0,
        "overall_rating": 0,
        "created_at": "2021-04-04T18:00:37.278530Z",
        "Expiration_Time": "00:15:00",
        "Poststatus": [],
        "update_expired": false,
        "get_total_comments": 0
    }
]
```


TC7 Nick and Olga look at the messages in the tech topic

The JSON returns looks like this,

```
response = requests.get(status_view_url_tech,headers=headers)
print(response.json())
```


```
[{'id': 70, 'Title': 'Which Iphone', 'status': 'Got the new Iphone', 'Topic': 'Tech', 'author_username': 'olga', 'created_at':
'2021-04-03T12:12:05.661492Z', 'Expiration_Time': '00:05:00', 'Poststatus': [], 'update_expired': True, 'get_total_comments':
0}, {'id': 71, 'Title': 'Which android', 'status': 'Got the new 5G goglephone', 'Topic': 'Tech', 'author_username': 'nick', 'cr
eated_at': '2021-04-03T12:17:11.300430Z', 'Expiration_Time': '00:15:00', 'Poststatus': [], 'update_expired': True, 'get_total_c
omments': 0}, {'id': 72, 'Title': 'Opera', 'status': 'is opera any good', 'Topic': 'Tech', 'author_username': 'mary', 'created_
at': '2021-04-03T12:46:18.728835Z', 'Expiration_Time': '00:15:00', 'Poststatus': [], 'update_expired': False, 'get_total_commen
ts': 0}]
```


Using the URL,which has a Tech ending (?search=Tech) to filter, naturally beautifies the results

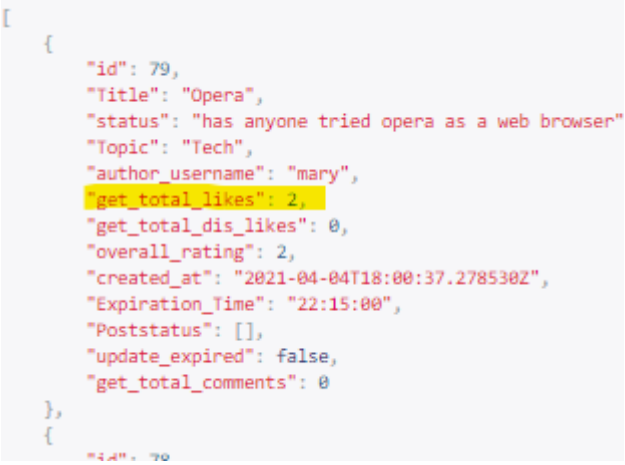
http://10.61.64.76:8000/v1/StatusCommentsAndLikes/?search=Tech

TC8 Nick and Olga Like Mary's Post

The returns are

{'id': 25, 'status': 79, 'users': 34}
{'id': 26, 'status': 79, 'users': 33}

Id is the like's pk, status is the status FK and users is the users FK. If we look at the status URL we can see the likes



The names are not visible here but could be seen on the database dump
http://10.61.64.76:8000/v1/datadump/

TC9 Nestor likes Nick's Post and Dislikes Mary's

Returns are in the same format as above

{'id': 27, 'status': 78, 'users': 36}
Is the like
{'status': 79, 'users': 36}
Is the dislike and they can be seen on the status view


```
},
{
    "id": 78,
    "Title": "Which android",
    "status": "Got the new 5G goglephone",
    "Topic": "Tech",
    "author_username": "nick",
    "get_total_likes": 1,
    "get_total_dis_likes": 0,
    "overall_rating": 1,
```

```
{
    "id": 79,
    "Title": "Opera",
    "status": "has anyone tried opera a
    "Topic": "Tech",
    "author_username": "mary",
    "get_total_likes": 2,
    "get_total_dis_likes": 1,
    "overall_rating": 1,
    "created_at": "2021-04-04T18:00:37.
    "Expiration_Time": "22:15:00",
```


TC10 Nick Browses the tech topic, there should be no comments, Mary's post should have 2 likes 1 dislike, Nick's should have 1 like

The call returns this which is hard to read

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

response = requests.get(status_view_url_tech,headers=headers)
print(response.json())
```


```
[{'id': 79, 'Title': 'Opera', 'status': 'has anyone tried opera as a web browser', 'Topic': 'Tech', 'author_username': 'mary',
'get_total_likes': 2, 'get_total_dis_likes': 1, 'overall_rating': 1, 'created_at': '2021-04-04T18:00:37.278530Z', 'Expiration_T
ime': '22:15:00', 'Poststatus': [], 'update_expired': False, 'get_total_comments': 0}, {'id': 78, 'Title': 'Which android', 'st
atus': 'Got the new 5G goglephone', 'Topic': 'Tech', 'author_username': 'nick', 'get_total_likes': 1, 'get_total_dis_likes': 0,
'overall_rating': 1, 'created_at': '2021-04-04T17:38:07.002100Z', 'Expiration_Time': '22:35:00', 'Poststatus': [], 'update_expi
red': False, 'get_total_comments': 0}, {'id': 77, 'Title': 'Which Iphone', 'status': 'Got the new Iphone', 'Topic': 'Tech', 'au
thor_username': 'olga', 'get_total_likes': 0, 'get_total_dis_likes': 0, 'overall_rating': 0, 'created_at': '2021-04-04T17:26:5
9.533845Z', 'Expiration_Time': '22:55:00', 'Poststatus': [], 'update_expired': False, 'get_total_comments': 0}]
```


From the URL it comes from its much more readable

```
[
    {
        "id": 79,
        "Title": "Opera",
        "status": "has anyone tried opera as a web brow
        "Topic": "Tech",
        "author_username": "mary",
        "get_total_likes": 2,
        "get_total_dis_likes": 1,
        "overall_rating": 1,
        "created_at": "2021-04-04T18:00:37.278530Z",
        "Expiration_Time": "22:15:00",
        "Poststatus": [],
        "update_expired": false,
        "get_total_comments": 0
    },
    {
        "id": 78,
        "Title": "Which android",
        "status": "Got the new 5G goglephone",
        "Topic": "Tech",
        "author_username": "nick",
        "get_total_likes": 1,
        "get_total_dis_likes": 0,
        "overall_rating": 1,
        "created_at": "2021-04-04T17:38:07.002100Z",
        "Expiration_Time": "22:35:00",
        "Poststatus": [],
        "update_expired": false,
        "get_total_comments": 0
    },
    {
        "id": 77,
        "Title": "Which Iphone",
        "status": "Got the new Iphone",
        "Topic": "Tech",
        "author_username": "olga",
        "get_total_likes": 0,
        "get_total_dis_likes": 0,
        "overall_rating": 0,
        "created_at": "2021-04-04T17:26:59.533845Z",
        "Expiration_Time": "22:55:00",
        "Poststatus": [],
        "update_expired": false,
        "get_total_comments": 0
    }
]
```


TC 11 Mary likes her own post this should fail

This is part of my code I am unhappy with. As I overode the save method in models the error handling is ugly. From my script post call I just get a JSON decode error, which says nothing about the problem.

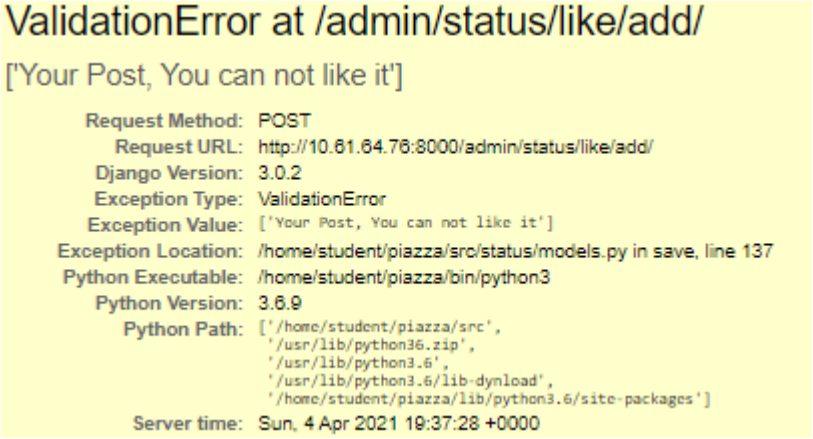
```
mary_response = requests.post(likes_url, headers=headers, data = record)
print(mary_response.json())
```

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
<ipython-input-55-c1ebf6f92c00> in <module>
      8
      9 mary_response = requests.post(likes_url, headers=headers, data = record)
---> 10 print(mary_response.json())

~\anaconda3\lib\site-packages\requests\models.py in json(self, **kwargs)
```

Doing the same call in admin mode gives a slightly but only slightly better error message. I think even this would be removed once DEBUG is turned off in settings

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```



Stelios showed examples using validators which are much better, but I could not work out how to prevent this by just comparing the value I needed to pass in the whole object. I do not know if this is poor coding from me, or if this is the only way to do it and custom error handling needs to be added either overriding on the save method in models or the create method on posting.

TC12 Nick and Olga comment on Mary's code in a Round Robin fashion at least 2 comments each

So the output is:

{'status': 79, 'comment': 'Not used it much, think mainly used in Africa', 'reaction': 33}
{'status': 79, 'comment': "yes Tim's working on something", 'reaction': 34}
{'status': 79, 'comment': 'Oh Really', 'reaction': 33}
{'status': 79, 'comment': 'interesting', 'reaction': 34}

And viewed from the status view end point

```
{
    "id": 79,
    "Title": "Opera",
    "status": "has anyone tried opera as a web browser",
    "Topic": "Tech",
    "author_username": "mary",
    "get_total_likes": 2,
    "get_total_dis_likes": 1,
    "overall_rating": 1,
    "created_at": "2021-04-04T18:00:37.278530Z",
    "Expiration_Time": "22:15:00",
    "Poststatus": [
        {
            "comment": "Not used it much, think mainly used in Africa",
            "author_username": "olga",
            "created_at": "2021-04-04T19:43:22.067784Z"
        },
        {
            "comment": "yes Tim's working on something",
            "author_username": "nick",
            "created_at": "2021-04-04T19:43:22.166963Z"
        },
        {
            "comment": "Oh Really",
            "author_username": "olga",
            "created_at": "2021-04-04T19:43:22.262917Z"
        },
        {
            "comment": "interesting",
            "author_username": "nick",
            "created_at": "2021-04-04T19:43:22.354933Z"
        }
    ],
    "update_expired": false,
    "get_total_comments": 4
}
```


TC 13 Nick looks at the comments in the Tech topic and can see the number of likes and dislikes

The call is made, to make it easier see from the end point:

http://10.61.64.76:8000/v1/StatusCommentsAndLikes/?search=Tech

Or a hard to read screenshot

TC 14 Nestor Posts in the Health Topic

The call returns

{'Title': 'Marathon ideas', 'status': 'Running a marathon in Autumn, any tips?', 'author': 36, 'Topic': 'Health', 'Expiration_Time': '00:15:00'}

TC15 Mary browses heath topics, should only be one

This is the return and the view from the URL

```
response = requests.get(status_view_url_health,headers=headers)
print(response.json())
```

```
[{'id': 80, 'Title': 'Marathon ideas', 'status': 'Running a marathon in Autumn, any tips?', 'Topic': 'Health', 'author_usernam
e': 'nestor', 'get_total_likes': 0, 'get_total_dis_likes': 0, 'overall_rating': 0, 'created_at': '2021-04-04T19:46:57.350479Z',
'Expiration_Time': '00:15:00', 'Poststatus': [], 'update_expired': False, 'get_total_comments': 0}]
```

And viewed from the endpoint

http://10.61.64.76:8000/v1/StatusCommentsAndLikes/?search=Health


```
{
    "id": 80,
    "Title": "Marathon ideas",
    "status": "Running a marathon in Autumn, any tips?",
    "Topic": "Health",
    "author_username": "nestor",
    "get_total_likes": 0,
    "get_total_dis_likes": 0,
    "overall_rating": 0,
    "created_at": "2021-04-04T19:46:57.350479Z",
    "Expiration_Time": "00:15:00",
    "Poststatus": [],
    "update_expired": false,
    "get_total_comments": 0
}
```


TC16 Mary comments on Nestor's post

This is the return and from the endpoint

{'status': 80, 'comment': 'Best to run almost everyday and increase your distance', 'reaction': 35}

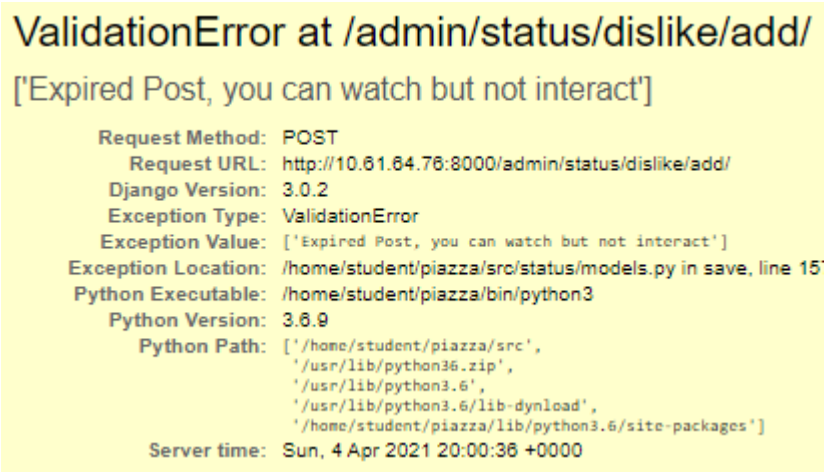

```
{
    "id": 80,
    "Title": "Marathon ideas",
    "status": "Running a marathon in Autumn, any tips?",
    "Topic": "Health",
    "author_username": "nestor",
    "get_total_likes": 0,
    "get_total_dis_likes": 0,
    "overall_rating": 0,
    "created_at": "2021-04-04T19:46:57.350479Z",
    "Expiration_Time": "00:15:00",
    "Poststatus": [
        {
            "comment": "Best to run almost everyday and increase your distance",
            "author_username": "mary",
            "created_at": "2021-04-04T19:57:22.453115Z"
        }
    ],
    "update_expired": false,
    "get_total_comments": 1
}
```

TC17 Mary dislikes Nestor's post but it fails as expired
I adjusted the expiry time (above the post is live as see from `"update_expired": false,`
). Again the call fails, but I have an ugly JSON decode error

```
mary_response = requests.post(dislikes_url, headers=headers, data = record)
print(mary_response.json())
```

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
<ipython-input-62-394f7e7e4e94> in <module>
      8
      9 mary_response = requests.post(dislikes_url, headers=headers, data = record)
---> 10 print(mary_response.json())
```

And doing the same in admin gives a better but still ugly rejection



TC18 Nestor looks at all the posts in the health topic, should be only his status and Mary's comment

The call returns

```
response = requests.get(status_view_url_health,headers=headers)
print(response.json())
```


```
[{'id': 80, 'Title': 'Marathon ideas', 'status': 'Running a marathon in Autumn, any tips?', 'Topic': 'Health', 'author_usernam
e': 'nestor', 'get_total_likes': 0, 'get_total_dis_likes': 0, 'overall_rating': 0, 'created_at': '2021-04-04T19:46:57.350479Z',
'Expiration_Time': '00:02:00', 'Poststatus': [{'comment': 'Best to run almost everyday and increase your distance', 'author_use
rname': 'mary', 'created_at': '2021-04-04T19:57:22.453115Z'}], 'update_expired': True, 'get_total_comments': 1}]
```


Which seen from the endpoint is
http://10.61.64.76:8000/v1/StatusCommentsAndLikes/?search=Health

```
{
    "id": 80,
    "Title": "Marathon ideas",
    "status": "Running a marathon in Autumn, any tips?",
    "Topic": "Health",
    "author_username": "nestor",
    "get_total_likes": 0,
    "get_total_dis_likes": 0,
    "overall_rating": 0,
    "created_at": "2021-04-04T19:46:57.350479Z",
    "Expiration_Time": "00:02:00",
    "Poststatus": [
        {
            "comment": "Best to run almost everyday and increase your distance",
            "author_username": "mary",
            "created_at": "2021-04-04T19:57:22.453115Z"
        }
    ],
    "update_expired": true,
    "get_total_comments": 1
}
```


TC19 Nick looks at expired messages in sport.

I get the correct response(nothing or []) but with a caveat:

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

response = requests.get(expired_sport_status_url,headers=headers)
print(response.json())
```

```
[]
```

My database does not allow only expired or live posts to be viewed. This is because I removed the field and instead have methods that calculate if the post is live from this line of code:
now >= (self.created_at + self.Expiration_Time)

When a search can be made in a URL such as ?search=Health in
http://10.61.64.76:8000/v1/StatusCommentsAndLikes/?search=Health
This can only be a database field not a @property. However this is better design as updating the boolean required an event to trigger the check. Before this event (most likely POST or save) the database is both wrong and contains redundant data.
It is better to lose this functionality and instead handle it client-side. You could easily remove the live status as the information is in the JSON in this calculated field


```
    ],
    "update_expired": true,
```


Which comes from the method
    @property
    def update_expired(self):

In the status model.

TC20 Nestor looks for the most interacted post highest overall score should be Mary's.

This also can not be done in my API for a similar reason. However in this case also the returned JSON gives the required data and this request could be handled client side.

The method

```
@property
def overall_rating(self):
```

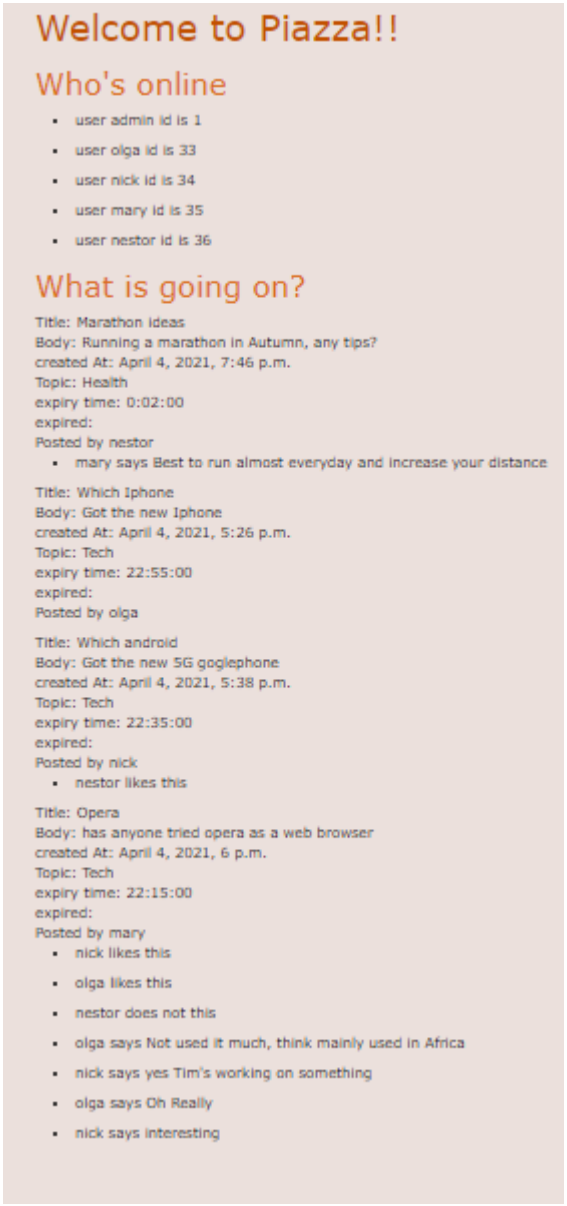
Gives an overall rating and a for loop clientside could find the highest post as long as piazza was a reasonable size. As piazza grew you would need to find a better solution as looping through every status clientside is not efficient. However to do this serverside would require making the database more complex and thus also less flexible in the future as the app develops.

So we can see Mary's post is the highest overall rating


```
"id": 79,
"Title": "Opera",
"status": "has anyone tried opera as a web browser",
"Topic": "Tech",
"author_username": "mary",
"get_total_likes": 2,
"get_total_dis_likes": 1,
"overall_rating": 1,
"created_at": "2021-04-04T18:00:37.278530Z",
"Expiration_Time": "22:15:00",
"Poststatus": [
    {
```


So at the end of the tests here is my front end site

http://10.61.64.76:8000/viewstatus/



Phase E

I have addressed some of the issues chronologically in the early part of the course. Overall I really enjoyed this exercise. There are some issues that need to be addressed that would significantly improve this app. With more time I think I could solve them but doing so now would mean less time to revise for the exams.

A login system.

A huge weakness is that people can post as other people. Once someone has a token they could use someone else's PK and the post.

The django documentation lists how to do this (django, 1), but it would be hard to test and imagine without a front end website.

Once implemented logged the permission classes could be updated in views

```
#These view determine what appears at the end
class StatusViewSet(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]
    queryset = Status.objects.all()
    serializer_class = StatusSerializer
    filter_backends = [filters.SearchFilter,
    search_fields = ['Topic', 'id']
```

Error handling.

I managed to stop the illegal post, but I did this by overriding the save method and the consequences are an ugly and unhelpful save method. It may be possible to use validators, but these seem better for handling input from forms (django, 2). There were a few posts of vaguely similar problems on stack overflow and they seemed to be more likely to favour preventing the post being made on the views and serializers, rather than preventing the database from saving. This would also give more flexibility for the future.

Record handling

The views I have made work so that the primary key after the default URL allows a single record to be looked at and changed via this URL. One I know is stable is

http://10.61.64.76:8000/v1/user/1/

This shows and allows the admin account to make PUT requests. Because the testing scripts didn't require much record manipulation there may be significant improvements that can be made to the way my API handles these.

Bibliography

Referenced in Text

django, 1, https://docs.djangoproject.com/en/3.1/topics/auth/default/
django, 2, https://docs.djangoproject.com/en/3.1/ref/validators/#
Medium,1
https://medium.com/djangotube/django-like-and-dislike-buttons-model-design-like-youtube-f152b95e7f21

Referenced in source code - I have left comments for which classes or functions have been influenced but here is a list here

#https://stackoverflow.com/questions/22677070/additional-field-while-serializing-django-rest-framework tips on nesting

https://medium.com/@sarit.r/djangofilterbackend-searchfilter-orderingfilter-example-dbe8e56ff1fc
-advice on filtering
https://stackoverflow.com/questions/8016412/in-django-do-models-have-a-default-timestamp-fied -advice on timestamp
https://stackoverflow.com/questions/34305805/django-foreignkeyuser-in-models
-forgeign key in models
https://stackoverflow.com/questions/39883950/str-returned-non-string-type-tuple
-corrected problem with def __str__(self):
https://docs.djangoproject.com/en/3.1/ref/models/fields/choicefield
-for the topic
https://stackoverflow.com/questions/15307623/cant-compare-naive-and-aware-datetime-now-challenge-datetime-end
-problems with expired post method