You are tasked with building the API and datastore backend of a web forum for a small number

of users (< 100). The forum is a basic text system which has the capabilities to add posts,

retrieve posts, and like posts. Management does not believe in users editing or deleting existing

posts, for ethical reasons.

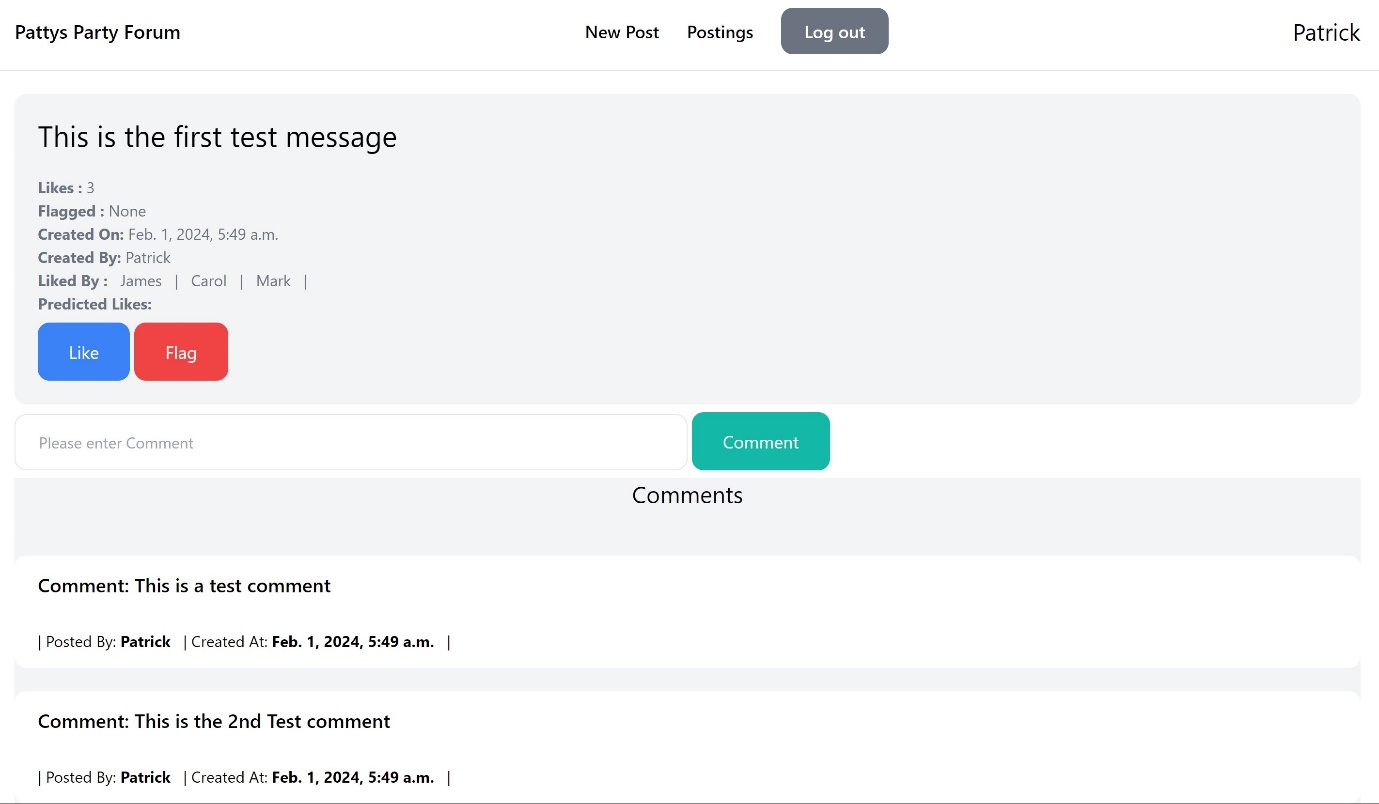
Task Specifications:

Functionality basics:

1. The forum is predominantly made up of posts created by users. Posts can be

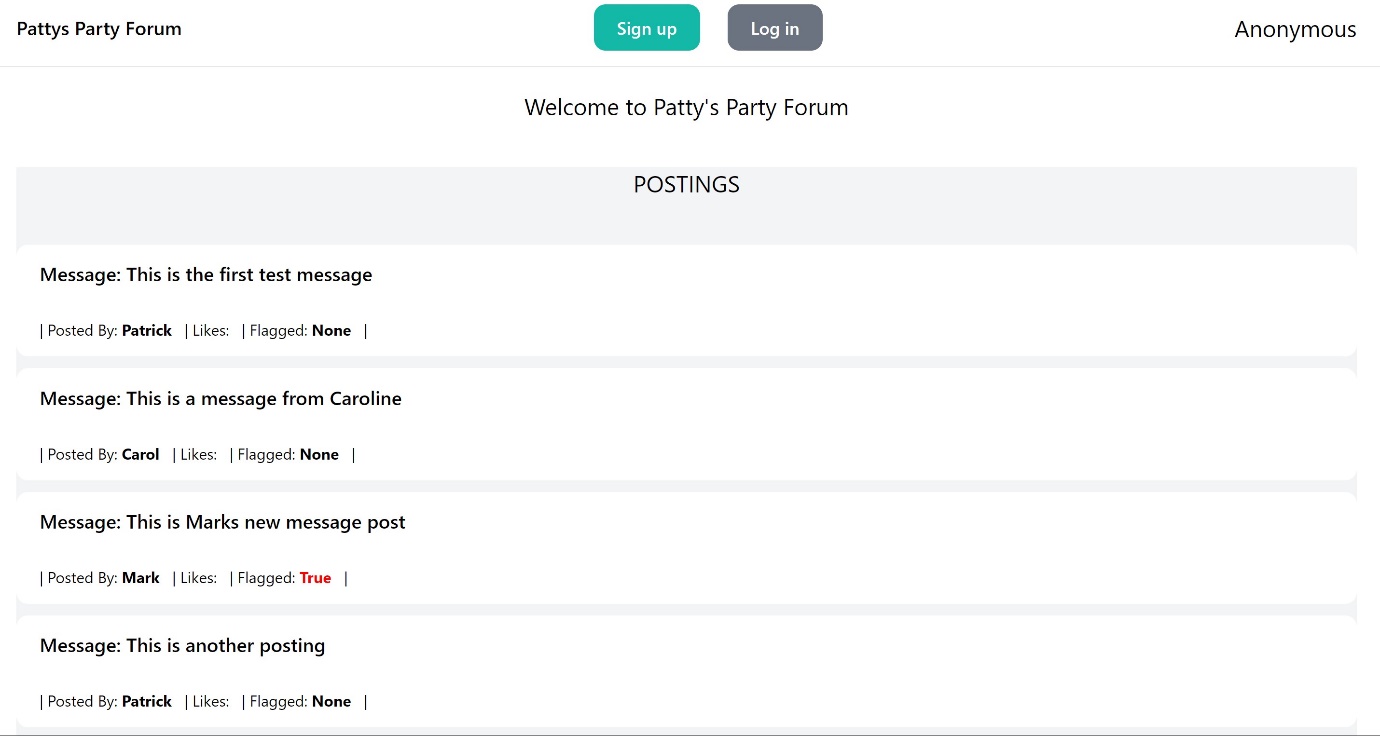
commented on. Posts can be assigned likes, each user can only like a post once, and

cannot like their own post.

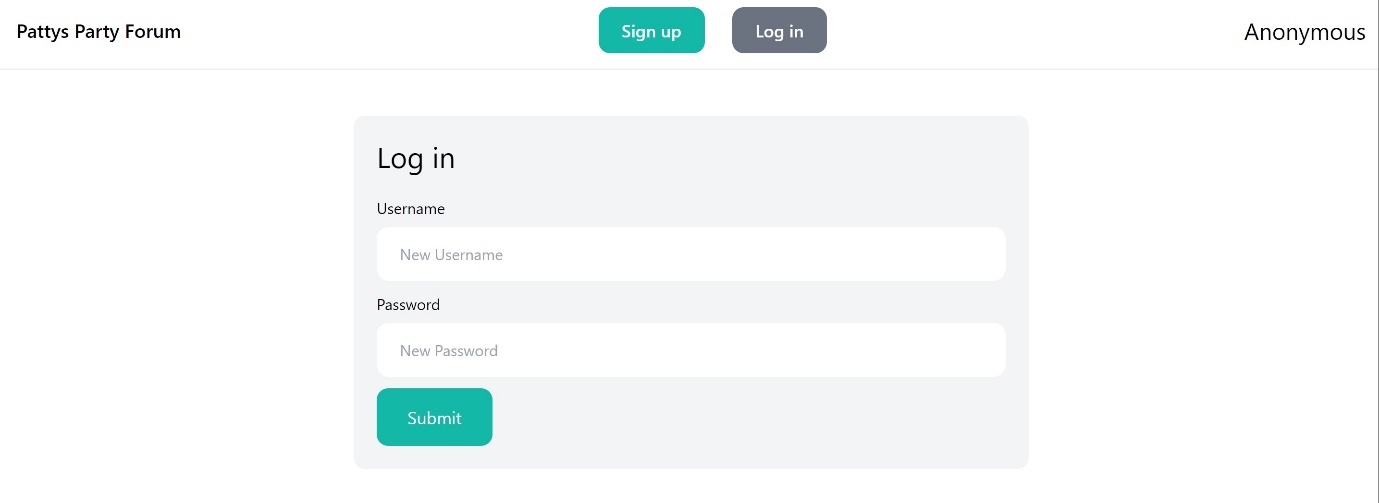


The Like button is available to all but validation done on execution in the view by checking the post is not made by the current logged in user, and also keeping track of all the previous likes done for the post

2. Users can view posts anonymously but must log in to post, comment, or like.



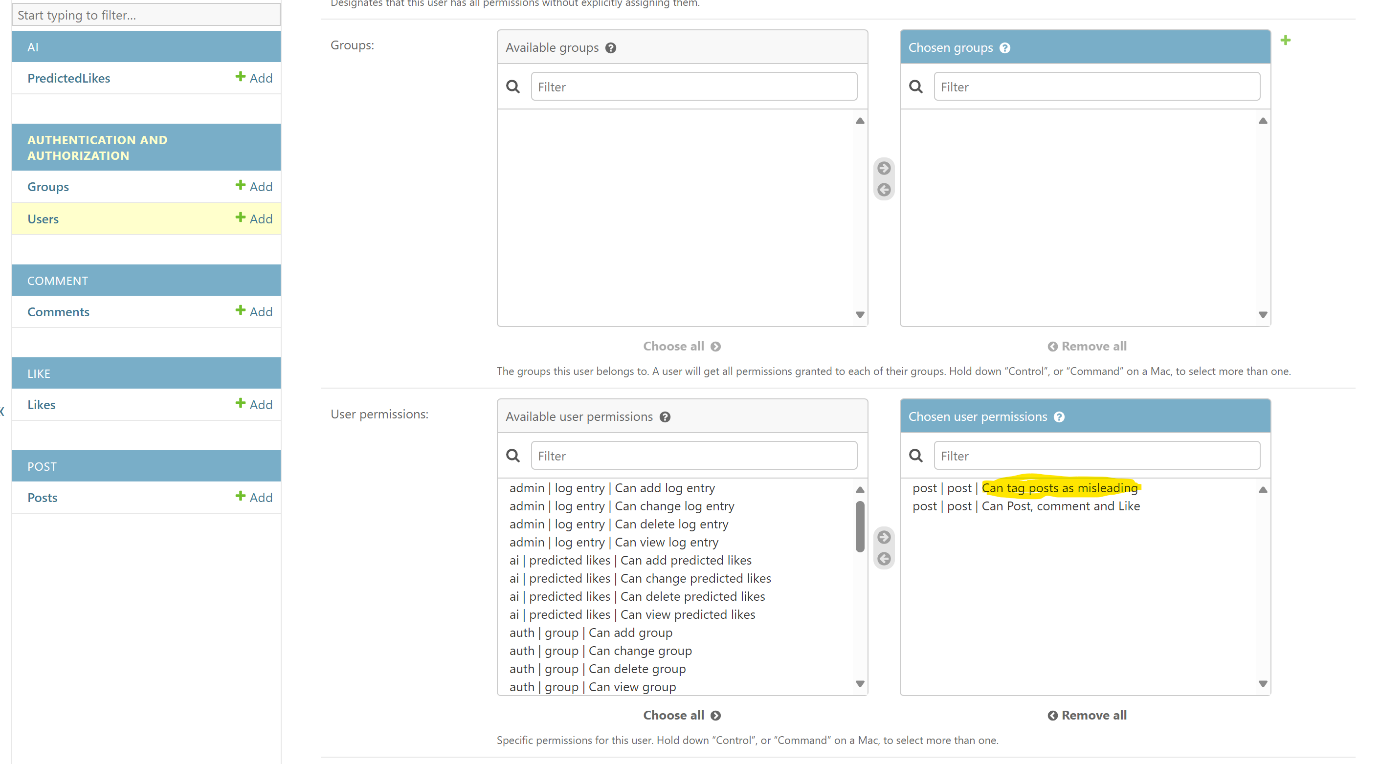
When the application starts, the front page shows the posts that are in the system. One can scroll and view, however, one is not able to see the details of the posts or create new posts without signing in. Once they do try, they will be confronted with the login page



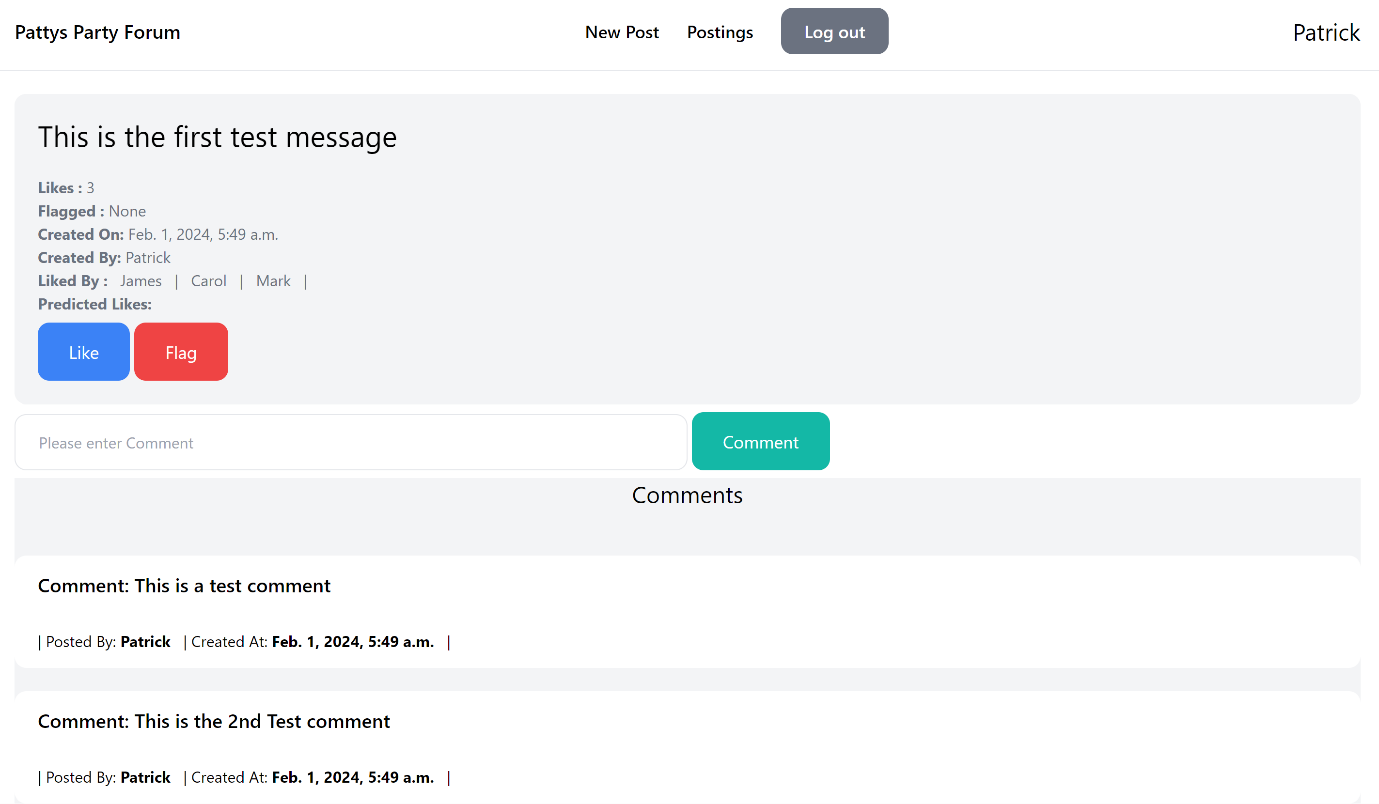
3. There are two categories of users, regular users and moderators. Regular users post,

comment, and like. Moderators can tag posts as “misleading or false information”, for

regulatory reasons.



In order for users for moderators to **FLAG** a post, the must have the above permission highlighted



If the user has the correct permission, the FLAG button is visible (handled in the template by looking at the current in-built user-permissions config. This feature is only managed on the front-end for now. This means that users can still flag posts even though they don’t have permissions from the back-end or through the API.   
This can be resolved by adding stronger permission modules in the code-base that look at permissions not only at a model level, but through groups as well.

Backend:

1. The system should be built in a common web framework, justify your choice, but do not

use a serverless architecture.

Used Python Django framework for the front-end interaction because it comes with the following:

* Enables rapid development of secure and manageable code
* Follows the Model View template design (which is a variation of the Model View Controller)
* Comes with in-build administrator portal
* In-built database management

Used the Django Rest Framework (DRF) for the external API interaction

2. The datastore backend may be whatever makes sense to you, again your choice should

be justified, your datastore should have some kind of out-of-the-box integration into

your chosen web framework, and efficient usage of the datastore is important.

Used the in-built SQLLite Database because

* it is light-weight
* comes built-in to the Django project.
* It is file based which allows for quick prototyping
* easy for testing
* Single file which can be transported and stored easily

3. User authentication needs to be handled in the application, do not use an external auth

provider. Users need to log in with a password, you are welcome to extend this with

2FA or social auth but this is not a requirement.

Used the existing Django user authentication built into the framework because:

* It handles accounts
* It handles groups
* It handles permissions
* It handles cookie-based user sessions
* Comes with the administrator portal built into the project

4. We want to enable regular users to use external third-party applications to interact with

our system in an automated way. Build an API for this.

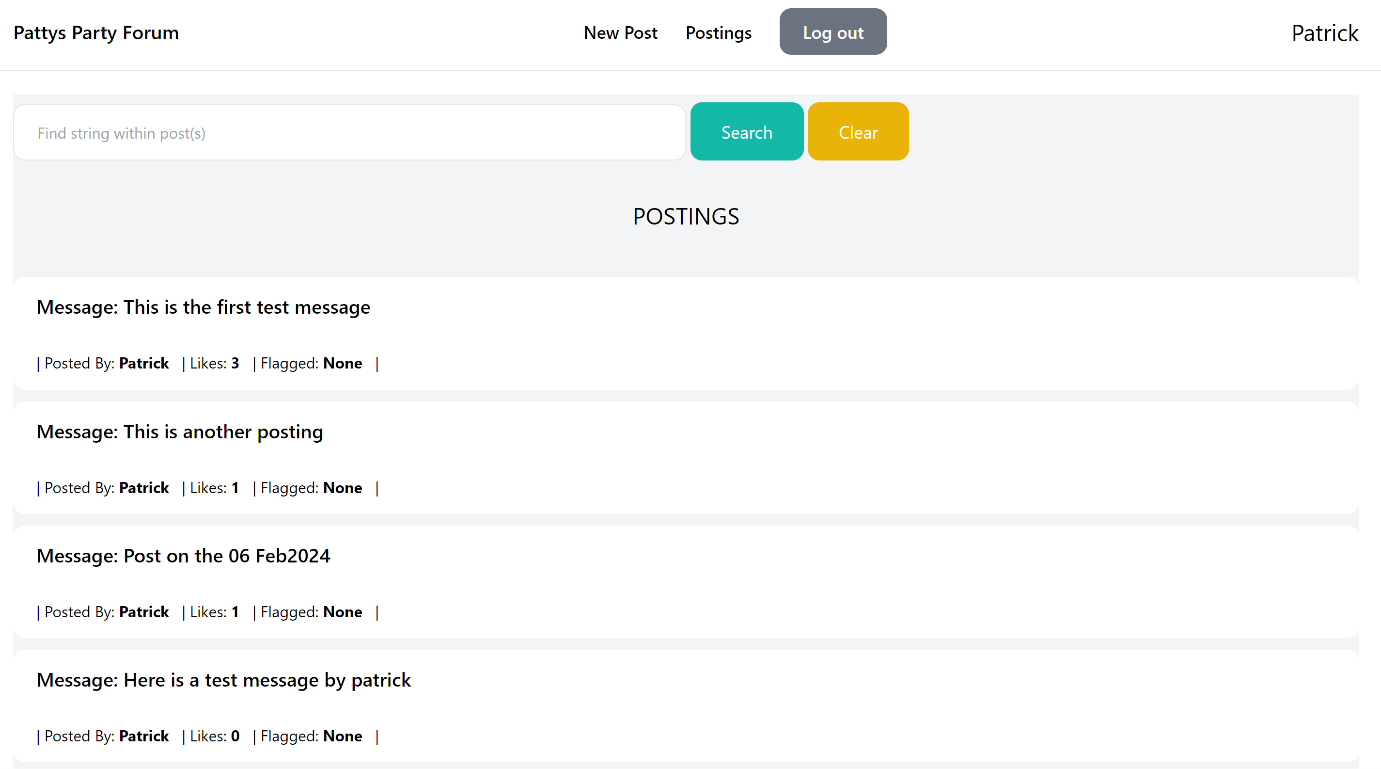
Used the Django-Rest-Framework as the foundation for the interaction

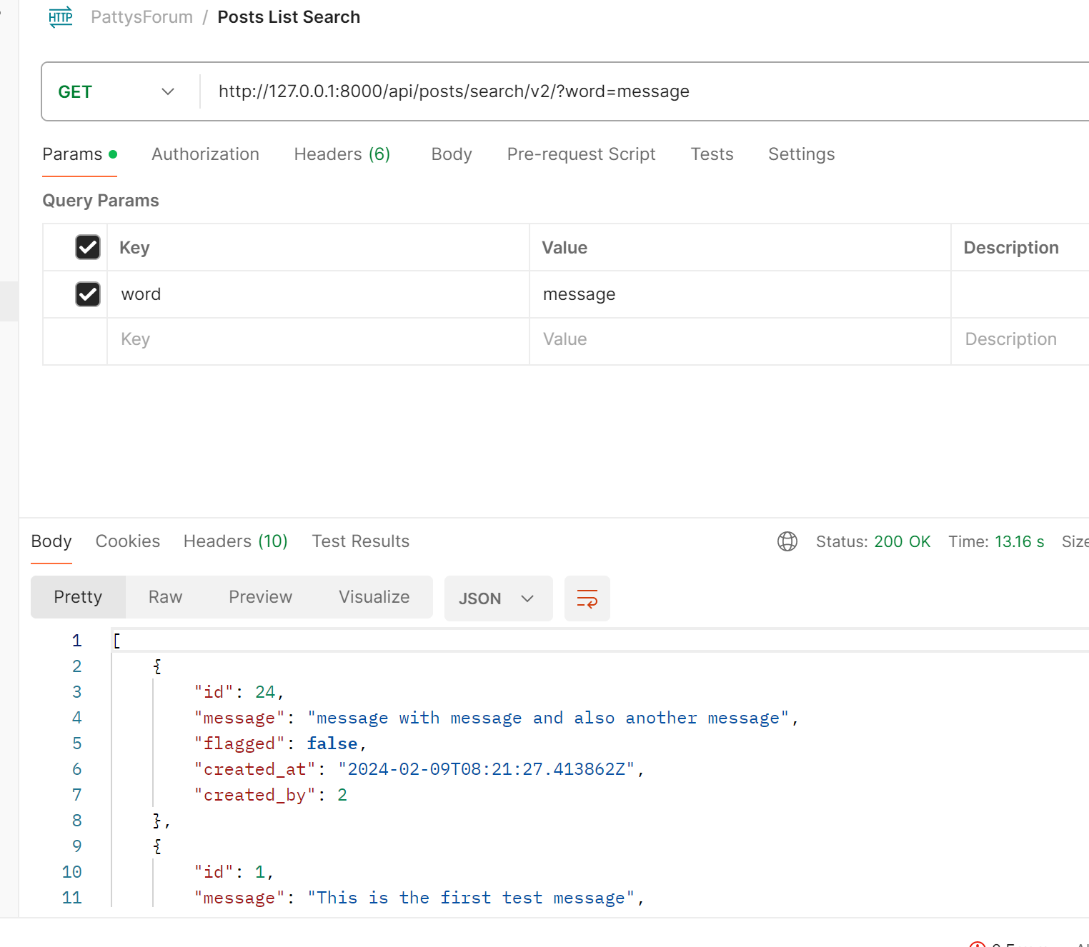
5. The API endpoint that returns posts needs to allow the requester to provide an optional

‘search word’ parameter which will return the posts sorted by the frequency of

occurrence of a specific word in the post, in descending order.

On the front-end interaction, the search word is processed through a form submission on the Posts page.

  
  
On the API request, it is handled through a search word query parameter

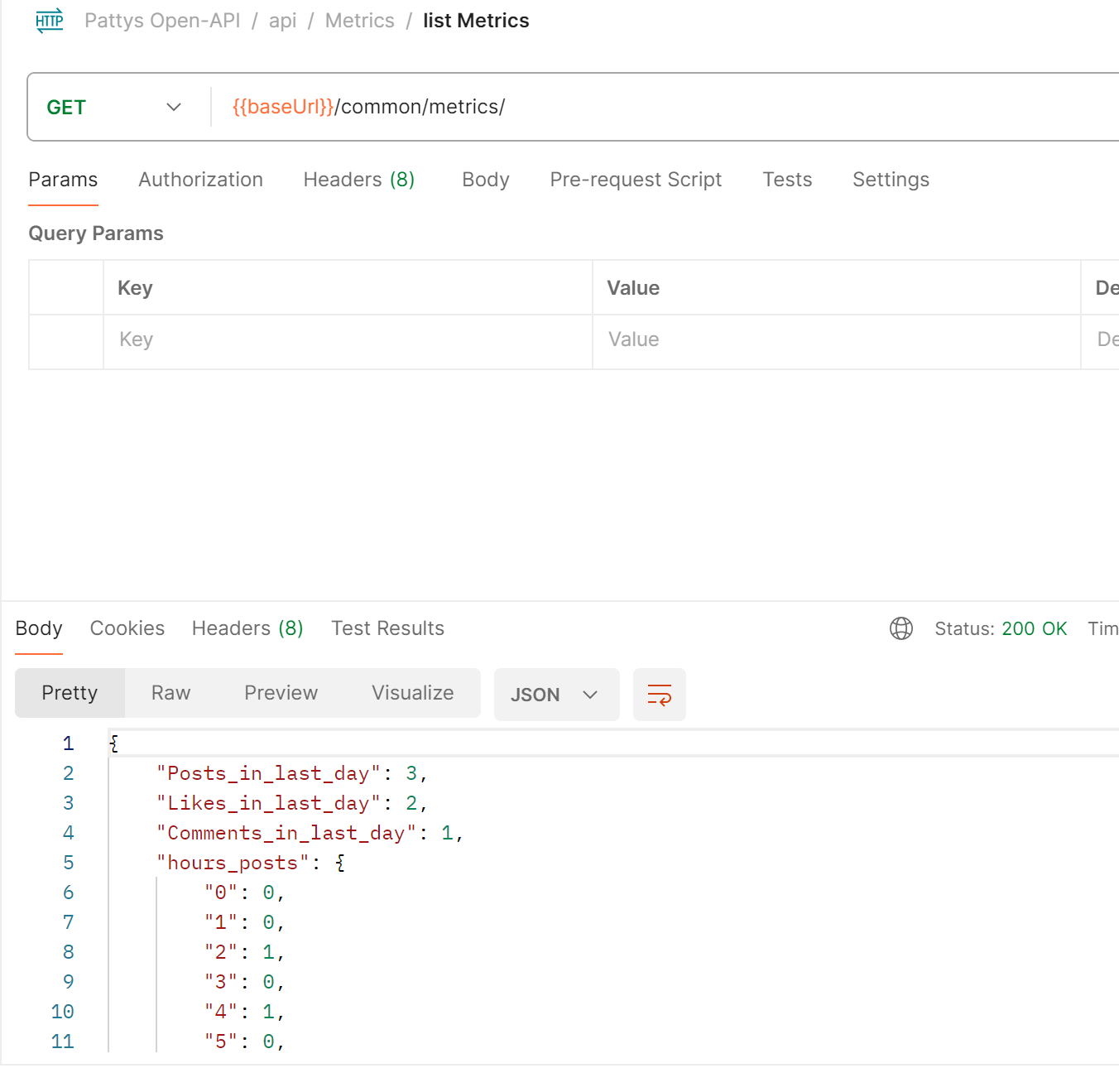


6. We want a way to get metrics from the system to publicly show off how popular our

forum is. Make an API endpoint which allows the requester to view the rate of posts and

comments created in the last day, per hour.

This was done on the front-end using the *metrics* class which is part of the common package in the source tree. Although this could be done on the API folder, I left it here as it returned the JSON structure which can be serialized at a later stage



7. Some users produce much better posts than others and we want to predict how many

likes a post will get based on its contents. When a user submits a new post, return the

predicted number of likes that that post will get based on the contents of the post and

the history of all posts on the system.

This is contained in the AI project and is resolved in the analytics function. The algorithm goes in the following manner:

*The algorithm used here is the following - claiming it as an proportionate like ratio  
- Get all the posts that have had likes before as the baseline  
- examine the contents of the posts and see overlapping of words  
- determine the ratio of common words for that post and assign that ratio to the likes  
- repeat for all the posts  
- apply that ratio to the number of users who liked the above because users can only like a post once*

8. We are very concerned about security. Suggest ways that we could harden this system

against common attacks and prevent bots from spamming forum posts on our system.

Notes:

*Currently the project is using Basic Authentication as the default setup for Django user auth. We can increase it by*

* *Use are 2FA as mentioned in this project spec*
* *Ensure that Debug Mode is disabled for production deployment*
* *Secure the Django Admin panel*
* *Change the user authentication from Basic to Token-based*
* *Use Django’s middleware to use X-Content-Type-Options to prevent content sniffing*
* *Use Django’s middleware to us X-Frome-Options to limit ClickJacking attacks*

Submission Guidelines:

● Code must be submitted using a public git repository in GitHub, with a full commit

history. Do not do the whole project in one go and commit it all at once, your use of Git

is an important part of this assessment

<https://github.com/Sabreme/PattysForum>

● Instructions to run the project need to be in the readme, the assessor needs to be able

to run it on a local machine using only the readme as instructions to do so (don’t forget

external dependencies!).

*Basic instructions in the readme whilst documentation contained in this documentation folder*

● Your submission should also contain a datastore of test/dummy data for the assessor to

Use

*Data store in the existing db.sqlite3 data file with the current data*

* *6 users*
* *Admin with password* ***AdminPassword***
* *No Groups, but Permissions assigned at Model Level for Posts*
  + *Can Tag Posts as misleading for User separation*
  + *Can Post, comment and like*

● Submit a public postman collection which allows easy testing of your API endpoints

*Postman-collection in the source folder*