

Name: Viplavi Wade  
Student Id: 13922741  
Subject: Cloud Computing  
Course: MSC Advanced Computing  
Date: December 10, 2023

## Cloud Computing Coursework

**GitHub Repo Link:** <https://github.com/ViplaviWade/Piazza.git>

### 1.0 Introduction:

Piazza is a dynamic social media platform designed to provide users with a personalized space for sharing thoughts, ideas, and engaging with a community. Leveraging modern web technologies, Piazza offers features such as user registration, authentication, and a robust set of APIs for creating, updating, and interacting with posts.

**2.0 Architecture Overview:** The architecture of Piazza follows a server-client model:

Server: Implemented using Node.js and Express, the server handles HTTP requests and interacts with the MongoDB database.

Database: MongoDB is employed as the primary database for its flexibility and scalability. It stores user information in the 'users' collection and post-related data in the 'posts' collection.

Authentication: OAuth v2 is integrated to secure user authentication, ensuring a seamless and secure login process.

### 3.0 Technology Stack:

Server-Side: Node.js with Express for building a fast and efficient server.

Database: MongoDB for its NoSQL capabilities, facilitating flexibility in data storage.

Authentication: OAuth v2 ensures secure user authentication with support for third-party login providers.

## 4.0 Database Schema

### 4.1 Users Collection

The 'users' collection contains:

1. **username**: User's unique identifier.
2. **email**: User's email for communication and identification.
3. **password**: Encrypted password for secure authentication.
4. **date**: Registration date for tracking user activity.

### 4.2 Posts Collection

The 'posts' collection includes:

1. **post\_title**: Title of the post.
2. **post\_topic**: Topic or category to which the post belongs.
3. **message**: Content of the post.
4. **expiration\_time**: Time after which the post is marked as 'EXPIRED'.
5. **status**: Current status of the post ('LIVE' or 'EXPIRED').
6. **like\_users, dislike\_users**: Arrays storing users who liked or disliked the post.
7. **likes\_count, dislikes\_count**: Counts of likes and dislikes.
8. **comments**: Array storing comments on the post.
9. **comments\_count**: Count of comments on the post.
10. **post\_id**: Unique identifier for the post.
11. **timestamp**: Timestamp indicating when the post was created.
12. **postOwner**: Username of the post creator.

## 5.0 API Endpoints using test cases: (used Kubernetes deployment Endpoint IP address)

### 5.1 Test Case-1: Olga, Nick, Mary, and Nestor register and are ready to access the Piazza API.

Olga registers to Piazza API

```

POST 35.202.9.65/api/register
Body
{
  "username": "Olga",
  "email": "olga@gmail.com",
  "password": "olga123"
}

```

```

{
  "username": "Olga",
  "email": "olga@gmail.com",
  "password": "$2a$05$6/6IFrmV4zlWsVtl06Z300g.EK9G5K6KQgxFAXUHpoqazRjeidW76",
  "_id": "6574a54352572cbbd75f0aac",
  "date": "2023-12-09T17:34:59.081Z",
  "__v": 0
}

```

## Nick registers to Piazza API

POST 35.202.9.65/api/register

Params Auth Headers (8) Body ● Pre-req. Tests Settings Cookies

**Body**

Pretty Raw Preview Visualize JSON ✖ ✖

```

1
2   "username": "Nick",
3   "email": "nick@gmail.com",
4   "password": "nick123"
5

```

1
2 "username": "Nick",
3 "email": "nick@gmail.com",
4 "password": "\$2a\$05\$j67P3De92lqkzzTib/MY.
gfkOCI92y4mPiQG8PAdj24thEa0req",
5 "\_id": "6574a56a58a131ae2517bf2f",
6 "date": "2023-12-09T17:35:38.306Z",
7 "\_\_v": 0
8

## Mary registers to Piazza API

POST 35.202.9.65/api/register

Params Auth Headers (8) Body ● Pre-req. Tests Settings Cookies

**Body**

Pretty Raw Preview Visualize JSON ✖ ✖

```

1
2   "username": "Mary",
3   "email": "mary@gmail.com",
4   "password": "mary123"
5

```

1
2 "username": "Mary",
3 "email": "mary@gmail.com",
4 "password": "\$2a\$05\$A/GeaINaBDVYh0Siq10.
pJMaX3GvB5vm9p6zBiItsbNdc0cbwy",
5 "\_id": "6574a588f47ef65551c0b7ee",
6 "date": "2023-12-09T17:36:08.079Z",
7 "\_\_v": 0
8

## Nestor registers to Piazza API

POST 35.202.9.65/api/register

Params Auth Headers (8) Body ● Pre-req. Tests Settings Cookies

**Body**

Pretty Raw Preview Visualize JSON ✖ ✖

```

1
2   "username": "Nestor",
3   "email": "nestor@gmail.com",
4   "password": "nestor123"
5

```

1
2 "username": "Nestor",
3 "email": "nestor@gmail.com",
4 "password": "\$2a\$05\$J75uA55w9ii.u//.
VDFfze6WrxwacE6TJJPKqgv1q0By3N02whTk",
5 "\_id": "6574a56f47ef65551c0b7f1",
6 "date": "2023-12-09T17:36:54.924Z",
7 "\_\_v": 0
8

All the users: Olga, Nick, Mary and Nestor are registered in the **Piazza API**.

## Piazza.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 887B TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns

Aggregation

Search Indexes

Filter

Type a query: { field: 'value' }

```
_id: ObjectId('6574a54352572cbbd75f0aac')
username: "Olga"
email: "olga@gmail.com"
password: "$2a$05$a/61FrmV4zlWsVtlo6Z300g.EK9G5K6KQgxFAXURpoqazRjejDw76"
date: 2023-12-09T17:34:59.081+00:00
__v: 0
```

```
_id: ObjectId('6574a56a58a131ae2517bf2f')
username: "Nick"
email: "nick@gmail.com"
password: "$2a$05$j67P3De92lqjkzzTib/MY.gfKOCl92y4mPiQG8PAjxj24thEaOreq"
date: 2023-12-09T17:35:38.306+00:00
__v: 0
```

```
_id: ObjectId('6574a588f47ef65551c6b7ee')
username: "Mary"
email: "mary@gmail.com"
password: "$2a$05$A/GEaDINaBDVPYhq0Siql0.pJM4x3GvBSvm9p6ZBIitsbNdecrbWy"
date: 2023-12-09T17:36:08.079+00:00
__v: 0
```

```
_id: ObjectId('6574a5b6f47ef65551c6b7f1')
username: "Nestor"
email: "nestor@gmail.com"
password: "$2a$05$J75uA55w9iiL.u//VDffZee6WrhwacE6TJJPKqgv1qD0y3N82whTkm"
date: 2023-12-09T17:36:54.924+00:00
__v: 0
```

**5.2 Test Case 2:** Olga, Nick, Mary, and Nestor use the oAuth v2 authorisation service to register and get their tokens.

Olga receives a token for her login

POST 35.202.9.65/api/login

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON

```

1 {"email": "olga@gmail.com",
2 "password": "olga123"}
3
4

```

Body v

Pretty Raw Preview Visualize JSON

1 "auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJfaWQiOiI2NTc0YTYtMjUyNTcyZjIzZjBhYmlCJpYXQiojE3MDIxNDM2NTR9.  
x970PAacx\_1X0uY2XGSH\_sj1wQhwJn1oIScUpCU8kM"

Nick receives a token for his login

POST 35.202.9.65/api/login

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON

```

1 {"email": "nick@gmail.com",
2 "password": "nick123"}
3
4

```

Body v

Pretty Raw Preview Visualize JSON

1 "auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJfaWQiOiI2NTc0YTYtMjUyNTcyZjIzZjBhYmlCJpYXQiojE3MDIxNDM2NTR9.  
yCFXjhEr0xhJLg3U5o\_tVQPdjZhFFuGx6MM7mvw6Hw"

Mary receives a token for her login

POST 35.202.9.65/api/login

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON

```

1 {"email": "mary@gmail.com",
2 "password": "mary123"}
3
4

```

Body v

Pretty Raw Preview Visualize JSON

1 "auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJfaWQiOiI2NTc0YTYtMjUyNTcyZjIzZjBhYmlCJpYXQiojE3MDIxNDM3MTB9.  
TcA5PVKUWz4sr04xsIL32i0xScURsca8T4BIB5yvTA"

Nestor receives a token for her login

POST 35.202.9.65/api/login

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON

```

1 {"email": "nestor@gmail.com",
2 "password": "nestor123"}
3
4

```

Body v

Pretty Raw Preview Visualize JSON

1 "auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJfaWQiOiI2NTc0YTYtMjUyNTcyZjIzZjBhYmlCJpYXQiojE3MDIxNDM3NDB9.  
49mihB7YHfkmwqT9UGWJ4Q05Tx3CS7L0X-jh\_NtClvU"

All the users used the **oAuth v2 authorization** service to register and receive their tokens.

**5.3 Test Case 3:** Olga makes a call to the API without using her token. This call should be unsuccessful as the user is unauthorized.

POST 35.202.9.65/api/createPost

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```

1 {
2   "post_title": "Artificial Intelligence",
3   "post_topic": "Tech",
4   "message": "AI is on boom for 5 years",
5   "expiration_time": 90
6 }
7 
```

Body Pretty Raw Preview Visualize JSON

401 Unauthorized 295 ms 226 B Save as example

1 "message": "Access denied."

When Olga tried to make a call without her token, the API call failed as the user was not authorized. And the API returned the error message as "**Access Denied**".

**5.4 Test Case 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).

POST 35.202.9.65/api/createPost

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```

1 {
2   "post_title": "Artificial Intelligence",
3   "post_topic": "Tech",
4   "message": "AI is on boom for 5 years",
5   "expiration_time": 5
6 }
7 
```

Body Pretty Raw Preview Visualize JSON

200 OK 545 ms 573 B Save as example

```

1 {
2   "post_title": "Artificial Intelligence",
3   "post_topic": "Tech",
4   "message": "AI is on boom for 5 years",
5   "expiration_time": "2023-12-09T17:51:38.939Z",
6   "status": "LIVE",
7   "like_users": [],
8   "likes_count": 0,
9   "dislike_users": [],
10  "dislikes_count": 0,
11  "comments_count": 0,
12  "postId": "lIsSr0Bk4",
13  "postOwner": "Olga",
14  "_id": "6574a7fe751ebdc55a40a576",
15  "comments": [],
16  "timestamp": "2023-12-09T17:46:38.943Z",
17  "__v": 0
18 } 
```

After 5 minutes (expiration time) the post will not accept any further interactions (like/dislike/comments)

PUT 35.202.9.65/api/performAction?postId=lIsSr0Bk4&action=like

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Description	... Bulk Edit
<input checked="" type="checkbox"/> postId	lIsSr0Bk4		
<input checked="" type="checkbox"/> action	like		
Key	Value	Description	

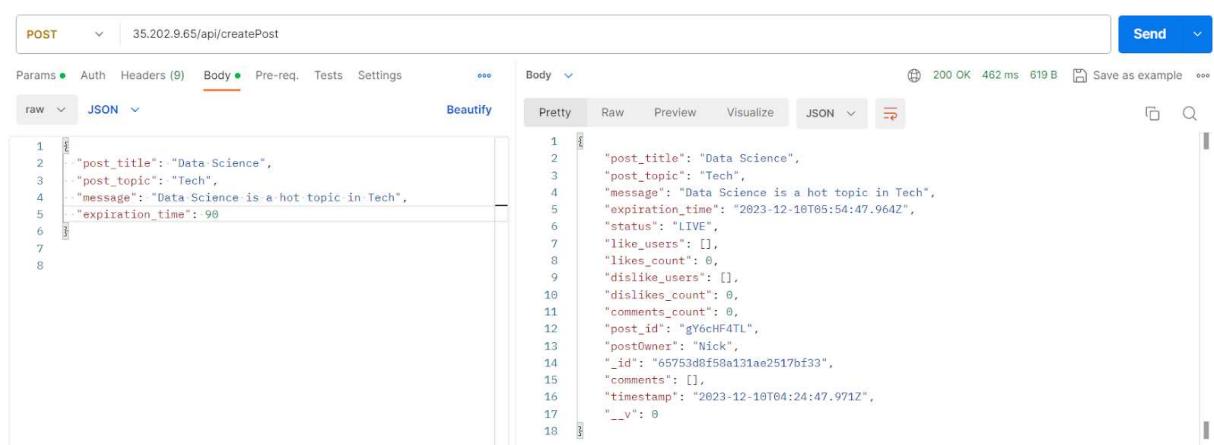
Body Pretty Raw Preview Visualize JSON

403 Forbidden 349 ms 245 B Save as example

1 "error": "Post expired for performing any action"

## 5.5 Test Case 5: Nick posts a message in the Tech topic with an expiration time using his token.

Nick posts a message in Tech topic with an expiration time of 90 minutes



The screenshot shows a POST request to the URL 35.202.9.65/api/createPost. The request is being made via a tool like Postman or cURL. The 'Body' tab is selected, showing the JSON payload:

```
POST 35.202.9.65/api/createPost
Params Auth Headers (9) Body Pre-req. Tests Settings ...
Body ...
Pretty Raw Preview Visualize JSON ...
1 "post_title": "Data Science",
2 "post_topic": "Tech",
3 "message": "Data Science is a hot topic in Tech",
4 "expiration_time": 90
5
6
7
8
```

The response status is 200 OK, with a response time of 462 ms and a response size of 619 B. The response body is displayed in a pretty-printed JSON format:

```
1 "post_title": "Data Science",
2 "post_topic": "Tech",
3 "message": "Data Science is a hot topic in Tech",
4 "expiration_time": "2023-12-10T05:54:47.964Z",
5 "status": "LIVE",
6 "like_users": [],
7 "like_count": 0,
8 "dislike_users": [],
9 "dislike_count": 0,
10 "comments_count": 0,
11 "post_id": "gY6cHF4TL",
12 "postOwner": "Nick",
13 "_id": "65753d0f50a131ae2517bf33",
14 "comments": [],
15 "timestamp": "2023-12-10T04:24:47.971Z",
16 "v": 0
17
18
```

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 687B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

Filter

Type a query: { field: 'value' }

```
_id: ObjectId('65753d8f58a131ae2517bf33')
post_title: "Data Science"
post_topic: "Tech"
message: "Data Science is a hot topic in Tech"
expiration_time: 2023-12-10T05:54:47.964+00:00
status: "LIVE"
▶ like_users: Array (empty)
  likes_count: 0
▶ dislike_users: Array (empty)
  dislikes_count: 0
  comments_count: 0
  post_id: "gY6cHF4TL"
  postOwner: "Nick"
▶ comments: Array (empty)
  timestamp: 2023-12-10T04:24:47.971+00:00
  __v: 0
```

### 5.6 Test Case 6: Mary posts a message in the Tech topic with an expiration time using her token.

POST [35.202.9.65/api/createPost](https://35.202.9.65/api/createPost) Send

Params • Auth Headers (9) Body • Pre-req. Tests Settings

raw JSON [Beautify](#) Pretty Raw Preview Visualize JSON

```
1 [ { 2   "post_title": "Cloud Computing", 3   "post_topic": "Tech", 4   "message": "This is Cloud project", 5   "expiration_time": 95 6 } ] 7 8 9 10 11 12 13 14 15 16 17 18 
```

200 OK 317 ms 608 B Save as example

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1019B TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB

**Find** Indexes Schema Anti-Patterns ⓘ Aggregation Search Indexes

Filter Type a query: { field: 'value' }

```
_id: ObjectId('65753e0858a131ae2517bf36')
post_title: "Cloud Computing"
post_topic: "Tech"
message: "This is Cloud project"
expiration_time: 2023-12-10T06:01:48.740+00:00
status: "LIVE"
  ▶ like_users: Array (empty)
  likes_count: 0
  ▶ dislike_users: Array (empty)
  dislikes_count: 0
  comments_count: 0
  post_id: "kzqNPnJ30"
  postOwner: "Mary"
  ▶ comments: Array (empty)
  timestamp: 2023-12-10T04:26:48.742+00:00
  __v: 0
```

These two messages are added in the database as shown above

**5.7 Test Case 7:** Nick and Olga browse all the available posts in the Tech topic; three posts should be available with zero likes, zero dislikes and no comments.

GET 35.202.9.65/api/getPostPerTopic?topic=Tech

Params • Auth Headers (8) Body Pre-req. Tests • Settings ...

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> topic	Tech			
Key	Value	Description		

Body ...

Pretty Raw Preview Visualize C

200 OK 305 ms 1.35 KB Save as example ...

Post Title	Post Topic	Message	Expiration Time	Status	Likes Count	Dislikes Count	Comments Count	P C
Artificial Intelligence	Tech	AI is on boom for 5 years	2023-12-09T17:51:38.939Z	LIVE	0	0	0	O
Data Science	Tech	Data Science is a hot topic in Tech	2023-12-10T05:54:47.964Z	LIVE	0	0	0	N
Cloud Computing	Tech	This is Cloud project	2023-12-10T06:01:48.740Z	LIVE	0	0	0	M

**5.8 Test Case 8:** Nick and Olga “like” Mary’s post on the Tech topic.

Olga likes Mary’s post on Tech topic

PUT 35.202.9.65/api/performAction?postId=kzqNPnJ30&action=like

Params • Auth Headers (9) Body • Pre-req. Tests Settings ...

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> postId	kzqNPnJ30			
<input checked="" type="checkbox"/> action	like			
Key	Value	Description		

Body ...

Pretty Raw Preview Visualize JSON

200 OK 422 ms 272 B Save as example ...

```

1
2 "message": "User has liked the post"
3

```

Nick likes Mary’s post on Tech topic

PUT 35.202.9.65/api/performAction?postId=kzqNPnJ30&action=like

Params • Auth Headers (9) Body • Pre-req. Tests Settings ...

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> postId	kzqNPnJ30			
<input checked="" type="checkbox"/> action	like			
Key	Value	Description		

Body ...

Pretty Raw Preview Visualize JSON

200 OK 340 ms 272 B Save as example ...

```

1
2 "message": "User has liked the post"
3

```

In the DB we can see likes\_count as 2 and like\_users are { Nick and Olga }

**Piazza.posts**

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.02KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

[Filter](#) Type a query: { field: 'value' }

```

_id: ObjectId('65753e0858a131ae2517bf36')
post_title: "Cloud Computing"
post_topic: "Tech"
message: "This is Cloud project"
expiration_time: 2023-12-10T06:01:48.740+00:00
status: "LIVE"
like_users: Array (2)
  0: "Olga"
  1: "Nick"
likes_count: 2
dislike_users: Array (empty)
dislikes_count: 0
comments_count: 0
post_id: "kzqNPnJ30"
postOwner: "Mary"
comments: Array (empty)
timestamp: 2023-12-10T04:26:48.742+00:00
__v: 0

```

### 5.9 Test Case 9: Nestor “likes” Nick’s post and “dislikes” Mary’s on the Tech topic

Nestor likes Nicks post on Tech

The screenshot shows a POST request to the URL `35.202.9.65/api/performAction?postId=gY6chF4TL&action=like`. The request body contains the following JSON:

```

{
  "postId": "gY6chF4TL",
  "action": "like"
}

```

The response status is 200 OK with a message: "User has liked the post".

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.05KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

Filter Type a query: { field: 'value' }

```

_id: ObjectId('65753d8f58a131ae2517bf33')
post_title: "Data Science"
post_topic: "Tech"
message: "Data Science is a hot topic in Tech"
expiration_time: 2023-12-10T05:54:47.964+00:00
status: "LIVE"
like_users: Array (1)
  0: "Nestor"
likes_count: 1
dislike_users: Array (empty)
dislikes_count: 0
comments_count: 0
post_id: "gY6cHF4TL"
postOwner: "Nick"
comments: Array (empty)
timestamp: 2023-12-10T04:24:47.971+00:00
--v: 0

```

Nestor dislikes Mary's topic on tech

PUT 35.202.9.65/api/performAction?postId=kzqNPrnJ30&action=dislike Send

Params • Auth Headers (9) Body • Pre-req. Tests Settings

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> postId	kzqNPrnJ30			
<input checked="" type="checkbox"/> action	dislike			
Key	Value	Description		

Body

Pretty Raw Preview Visualize JSON

```

1   "message": "User has disliked the post"
2
3

```

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.05KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

[Filter](#) Type a query: { field: 'value' }

```
_id: ObjectId('65753e0858a131ae2517bf36')
post_title: "Cloud Computing"
post_topic: "Tech"
message: "This is Cloud project"
expiration_time: 2023-12-10T06:01:48.740+00:00
status: "LIVE"
  ▶ like_users: Array (2)
    likes_count: 2
  ▶ dislike_users: Array (1)
    0: "Nestor"
    dislikes_count: 1
    comments_count: 0
    post_id: "kzqNPnJ30"
    postOwner: "Mary"
  ▶ comments: Array (empty)
    timestamp: 2023-12-10T04:26:48.742+00:00
    __v: 0
```

**5.10 Test Case 10:** Nick browses all the available posts on the Tech topic; at this stage, he can see the number of likes and dislikes for each post (Mary has two likes and one dislike, and Nick has one like). There are no comments made yet.

GET [35.202.9.65/api/getPostPerTopic?topic=Tech](https://35.202.9.65/api/getPostPerTopic?topic=Tech) Send

Params Auth Headers (8) Body Pre-req. Tests Settings

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> topic	Tech			
Key	Value	Description		

Body

Pretty Raw Preview Visualize

200 OK 313 ms 1.37 KB Save as example

Post Title	Post Topic	Message	Expiration Time	Status	Likes Count	Dislikes Count	Comments Count	Comments
Artificial Intelligence	Tech	AI is on boom for 5 years	2023-12-09T17:51:38.939Z	LIVE	0	0	0	O
Data Science	Tech	Data Science is a hot topic in Tech	2023-12-10T05:54:47.964Z	LIVE	1	0	0	N
Cloud Computing	Tech	This is Cloud project	2023-12-10T06:01:48.740Z	LIVE	2	1	0	M

**5.11 Test Case 11:** Mary likes her post on the Tech topic. This call should be unsuccessful; in Piazza, a post owner cannot like their messages.

The screenshot shows a POST request to `35.202.9.65/api/performAction?postId=kzqNPnJ30&action=like`. The request body contains parameters `postId` and `action`. The response status is 403 Forbidden, with the message "Post owner cannot perform any action on post".

Key	Value	Description
postId	kzqNPnJ30	
action	like	

```

1
2 "error": "Post owner cannot perform any action on post"
3

```

**5.12 Test Case 12:** Nick and Olga comment on Mary's post on the Tech topic in a round-robin fashion (one after the other, adding at least two comments each).

Nick comments on Mary's post

The screenshot shows a POST request to `35.202.9.65/api/performAction?postId=kzqNPnJ30&action=comment`. The request body is JSON with a comment from Nick. The response status is 200 OK, with the message "Comment posted successfully."

Key	Value	Description
comment	"Nick's first comment on Mary's post"	

```

1
2 ...
3

```

```

1
2 "message": "Comment posted successfully."
3

```

The screenshot shows a second POST request to `35.202.9.65/api/performAction?postId=kzqNPnJ30&action=comment`. The request body is JSON with a second comment from Nick. The response status is 200 OK, with the message "Comment posted successfully."

Key	Value	Description
comment	"Nick's second comment on Mary's post"	

```

1
2 ...
3

```

```

1
2 "message": "Comment posted successfully."
3

```

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.28KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

Filter Type a query: { field: 'value' }

```

_id: ObjectId('65753e0858a131ae2517bf36')
post_title: "Cloud Computing"
post_topic: "Tech"
message: "This is Cloud project"
expiration_time: 2023-12-10T06:01:48.740+00:00
status: "LIVE"
  like_users: Array (2)
    likes_count: 2
  dislike_users: Array (1)
    dislikes_count: 1
    comments_count: 2
    post_id: "kzqNPnJ30"
    postOwner: "Mary"
  comments: Array (2)
    0: Object
      commentId: "bj6pxZoD0"
      username: "Nick"
      comment: "Nick's first comment on Mary's post"
      _id: ObjectId('6575412c6d5978f1cc8ff221')
    1: Object
      commentId: "__VWtXMbV"
      username: "Nick"
      comment: "Nick's second comment on Mary's post"
      _id: ObjectId('6575419b6d5978f1cc8ff226')
timestamp: 2023-12-10T04:26:48.742+00:00
v: 0

```

Olga comments on Mary's post

PUT [35.202.9.65/api/performAction?postId=kzqNPnJ30&action=comment](https://35.202.9.65/api/performAction?postId=kzqNPnJ30&action=comment) Send

Params • Auth Headers (9) Body • Pre-req. Tests Settings [...](#)

Param	Type	Value
postId	String	kzqNPnJ30
action	String	comment
comment	String	"comment": "Olga's first comment on Mary's post"

Body [...](#)

raw [JSON](#) [Beautify](#)

```

1   {
2     "comment": "Olga's first comment on Mary's post"
3   }

```

Pretty Raw Preview Visualize JSON [...](#)

```

1   {
2     "comment": "Olga's first comment on Mary's post"
3   }

```

200 OK 376 ms 277 B [Save as example](#) [...](#)

PUT | 35.202.9.65/api/performAction?postId=kzqNPnJ30&action=comment

Params • Auth Headers (9) Body • Pre-req. Tests Settings ⚙️

raw JSON Beautify Body ⚙️ Pretty Raw Preview Visualize JSON ⚙️

Send ↗ 200 OK 377 ms 277 B Save as example ⚙️

```

1   ...
2   ... "comment": "Olga's second comment on Mary's post"
3

```

```

1   ...
2   ... "message": "Comment posted successfully."
3

```

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.51KB TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

[Filter](#) Type a query: { field: 'value' }

```

_id: ObjectId('65753e0858a131ae2517bf36')
post_title: "Cloud Computing"
post_topic: "Tech"
message: "This is Cloud project"
expiration_time: 2023-12-10T06:01:48.740+00:00
status: "LIVE"
  + like_users: Array (2)
    likes_count: 2
  + dislike_users: Array (1)
    dislikes_count: 1
    comments_count: 4
    post_id: "kzqNPnJ30"
    postOwner: "Mary"
  + comments: Array (4)
    * 0: Object
      commentId: "bj6pxZoDOO"
      username: "Nick"
      comment: "Nick's first comment on Mary's post"
      _id: ObjectId('6575412c6d5978f1cc8ff221')
    * 1: Object
      commentId: "__VWtxNbvb"
      username: "Nick"
      comment: "Nick's second comment on Mary's post"
      _id: ObjectId('6575419b6d5978f1cc8ff226')
    * 2: Object
      commentId: "QKBCWujNX"
      username: "Olga"
      comment: "Olga's first comment on Mary's post"
      _id: ObjectId('65754217a134b52b9325f89c')
    * 3: Object
      commentId: "fbSTgmySn"
      username: "Olga"
      comment: "Olga's second comment on Mary's post"
      _id: ObjectId('6575423683fc258ffeb71c02')
timestamp: 2023-12-10T04:26:48.742+00:00
__v: 0

```

**5.13 Test Case 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.

Post Title	Post Topic	Message	Expiration Time	Status	Likes Count	Dislikes Count	Comments Count	P C
Artificial Intelligence	Tech	AI is on boom for 5 years	2023-12-09T17:51:38.939Z	LIVE	0	0	0	O
Data Science	Tech	Data Science is a hot topic in Tech	2023-12-10T05:54:47.964Z	LIVE	1	0	0	N
Cloud Computing	Tech	This is Cloud project	2023-12-10T06:01:48.740Z	LIVE	2	1	4	M

**5.14 Test Case 14:** Nestor posts a message in the Health topic with an expiration time using her token.

```

1
2   "post_title": "Healthy Habbits",
3   "post_topic": "Health",
4   "message": "Health is wealth",
5   "expiration_time": "00
6
7
8
9
10
11
12
13
14
15
16
17
18
  
```

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.83KB TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

Filter

Type a query: { field: 'value' }

```
_id: ObjectId('65754305751ebdc55a40a57d')
post_title: "Healthy Habbits"
post_topic: "Health"
message: "Health is wealth"
expiration_time: 2023-12-10T06:08:05.188+00:00
status: "LIVE"
▶ like_users: Array (empty)
  likes_count: 0
▶ dislike_users: Array (empty)
  dislikes_count: 0
  comments_count: 0
  post_id: "T87skkgQU"
  postOwner: "Nestor"
▶ comments: Array (empty)
  timestamp: 2023-12-10T04:48:05.192+00:00
  __v: 0
```

**5.15 Test Case 15:** Mary browses all the available posts on the Health topic; at this stage, she can see only Nestor's post.

GET <35.202.9.65/api/getPostPerTopic?topic=Health> Send

Params • Auth Headers (8) Body Pre-req. Tests • Settings ⚙

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> topic	Health			
Key	Value	Description		

Body

Pretty Raw Preview Visualize ⚙

Post Title	Post Topic	Message	Expiration Time	Status	Likes Count	Dislikes Count	Comments Count	Pos Owr
Healthy Habbits	Health	Health is wealth	2023-12-10T06:08:05.188Z	LIVE	0	0	0	Nest

### 5.16 Test Case 16: Mary posts a comment in Nestor's message on the Health topic.

PUT 35.202.9.65/api/performAction?postId=T87skkgQU&action=comment

Params Auth Headers (9) Body Pre-req. Tests Settings

raw JSON Beautify

Pretty Raw Preview Visualize JSON

200 OK 389 ms 277 B Save as example

```

1 {
2   ...
3   "comment": "Mary's first comment on Nestor's post"
4 }
5
6   "message": "Comment posted successfully."
7 }
```

## Piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.95KB TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 72KB

[Find](#) [Indexes](#) [Schema Anti-Patterns ⓘ](#) [Aggregation](#) [Search Indexes](#)

[Filter](#) Type a query: { field: 'value' }

```

_id: ObjectId('65754305751ebdc55a40a57d')
post_title: "Healthy Habbits"
post_topic: "Health"
message: "Health is wealth"
expiration_time: 2023-12-10T06:08:05.188+00:00
status: "LIVE"
  like_users: Array (empty)
  likes_count: 0
  dislike_users: Array (empty)
  dislikes_count: 0
  comments_count: 1
  post_id: "T87skkgQU"
  postOwner: "Nestor"
  comments: Array (1)
    0: Object
      commentId: "Io0Wop4Gx"
      username: "Mary"
      comment: "Mary's first comment on Nestor's post"
      _id: ObjectId('657543b48c71afed61edfcba')
      timestamp: 2023-12-10T04:48:05.192+00:00
      v: 0

```

**5.17 Test Case 17:** Mary dislikes Nestor's message on the Health topic after the end of post-expiration time. This should fail

The screenshot shows a POSTMAN interface with the following details:

- Method: PUT
- URL: 35.202.9.65/api/performAction?postId=T87skkgQU&action=dislike
- Body tab selected
- Params section shows two checked items: postId (T87skkgQU) and action (dislike)
- Body content: "error": "Post expired for performing any action"
- Status: 403 Forbidden
- Time: 319 ms
- Size: 292 B

Mary dislikes Nestor's message on Health topic after the end of post-expiration, and this API call fails as the post is expired for performing any actions such as (likes/ dislikes/ comments)

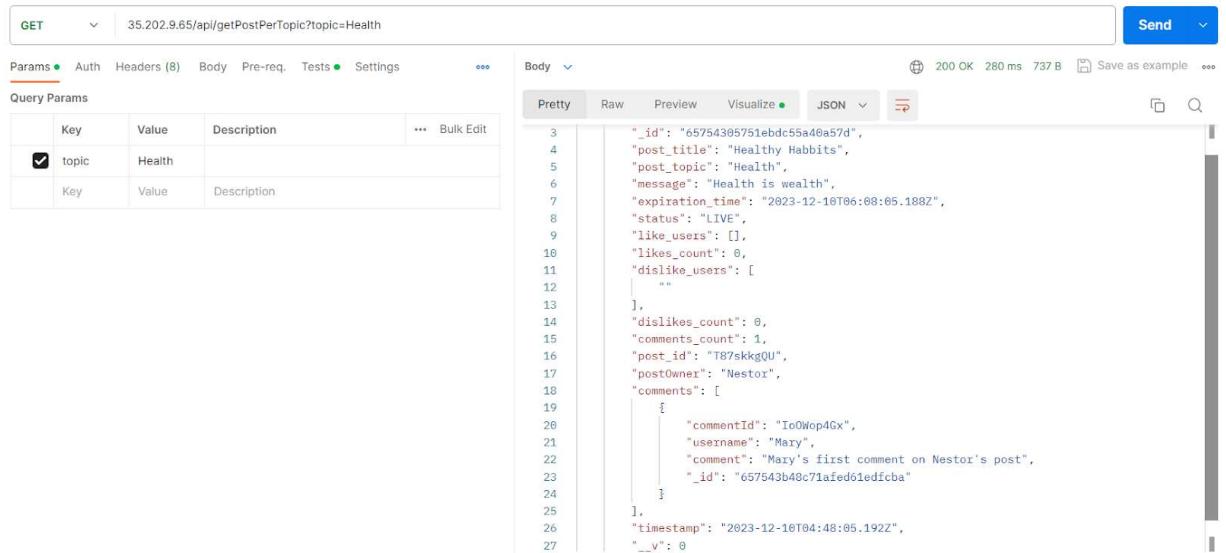
**5.18 Test Case 18:** Nestor browses all the messages on the Health topic. There should be only one post (his own) with one comment (Mary's).

The screenshot shows a GET request to 35.202.9.65/api/getPostPerTopic?topic=Health. The response body is as follows:

Post Title	Post Topic	Message	Expiration Time	Status	Likes Count	Dislikes Count	Comments Count	Pos Owr
Healthy Habbits	Health	Health is wealth	2023-12-10T06:08:05.188Z	LIVE	0	0	1	Nestc

Response details: 200 OK, 280 ms, 737 B.

Json view of the response:



```

GET 35.202.9.65/api/getPostPerTopic?topic=Health
Params Auth Headers (8) Body Pre-req. Tests Settings ...
Query Params


|                                     | Key   | Value  | Description |
|-------------------------------------|-------|--------|-------------|
| <input checked="" type="checkbox"/> | topic | Health |             |
|                                     | Key   | Value  | Description |


Body
Pretty Raw Preview Visualize JSON ...
200 OK 280 ms 737 B Save as example ...
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
{
  "_id": "65754385751ebdc55a40a57d",
  "post_title": "Healthy Habbits",
  "post_topic": "Health",
  "message": "Health is wealth",
  "expiration_time": "2023-12-10T06:08:05.188Z",
  "status": "LIVE",
  "like_users": [],
  "likes_count": 0,
  "dislike_users": [
    ""
  ],
  "dislikes_count": 0,
  "comments_count": 1,
  "post_id": "TB7skkgQU",
  "postOwner": "Nestor",
  "comments": [
    {
      "commentId": "Io0Wop4Gx",
      "username": "Mary",
      "comment": "Mary's first comment on Nestor's post",
      "_id": "657543b48c7iafed6iedfcba"
    }
  ],
  "timestamp": "2023-12-10T04:48:05.192Z",
  "__v": 0
}

```

**5.19 Test Case 19:** Nick browses all the expired messages on the Sports topic. These should be empty.



```

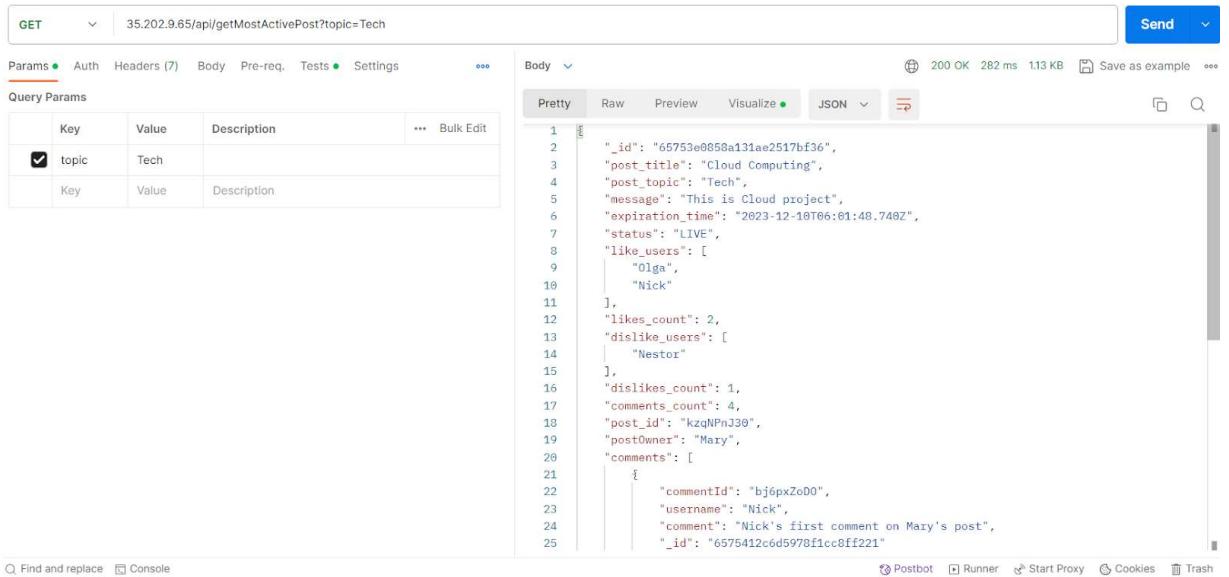
GET 35.202.9.65/api/getExpiredPosts?topic=Sports
Params Auth Headers (7) Body Pre-req. Tests Settings ...
Query Params


|                                     | Key   | Value  | Description |
|-------------------------------------|-------|--------|-------------|
| <input checked="" type="checkbox"/> | topic | Sports |             |
|                                     | Key   | Value  | Description |


Body
Pretty Raw Preview Visualize JSON ...
404 Not Found 281 ms 296 B Save as example ...
1
2
3
{
  "error": "No expired posts found for the topic: Sports"
}

```

**5.20 Test Case 20:** Nestor queries for an active post with the highest interest (maximum number of likes and dislikes) in the Tech topic. This should be Mary's post.



GET 35.202.9.65/api/getMostActivePost?topic=Tech

Params • Auth Headers (7) Body Pre-req. Tests • Settings

Query Params

Key	Value	Description
<input checked="" type="checkbox"/> topic	Tech	

Body

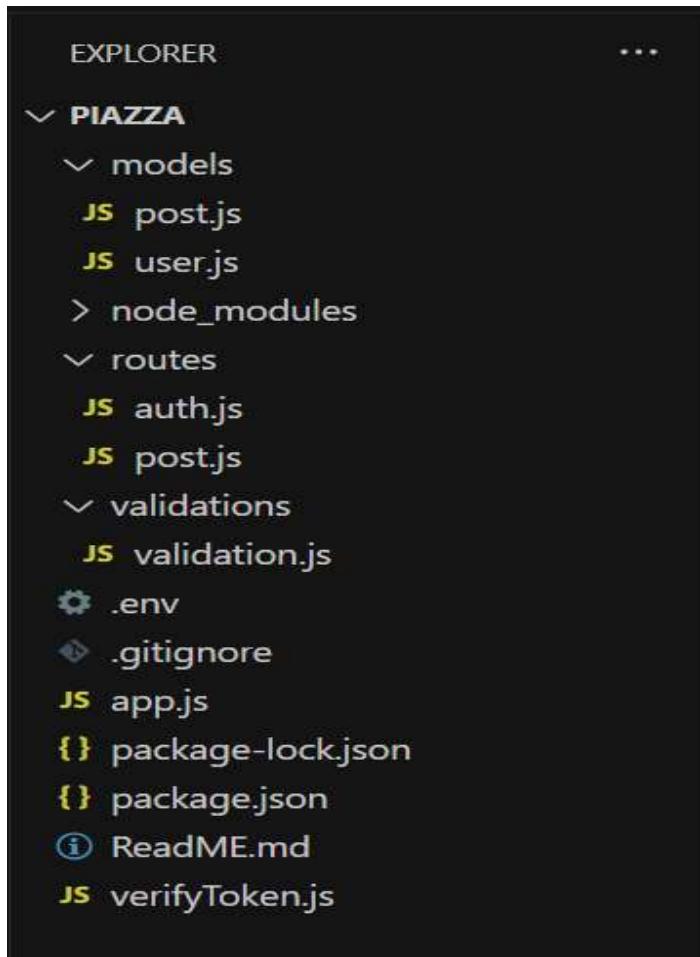
Pretty Raw Preview Visualize • JSON

```
1   "id": "65753e0858a131ae2517bf36",
2   "post_title": "Cloud Computing",
3   "post_topic": "Tech",
4   "message": "This is Cloud project",
5   "expiration_time": "2023-12-10T06:01:48.740Z",
6   "status": "LIVE",
7   "like_users": [
8     "Olga",
9     "Nick"
10   ],
11   "likes_count": 2,
12   "dislike_users": [
13     "Nestor"
14   ],
15   "dislikes_count": 1,
16   "comments_count": 4,
17   "post_id": "kzqNPnJ30",
18   "postOwner": "Mary",
19   "comments": [
20     {
21       "commentId": "bj6pxZo00",
22       "username": "Nick",
23       "comment": "Nick's first comment on Mary's post",
24       "_id": "6575412c6d5970f1cc8ff221"
25     }
26   ]
27 }
```

200 OK 282 ms 1.13 KB Save as example

Find and replace Console Postbot Runner Start Proxy Cookies Trash

## 6.0 File Structure:



**The Models folder:** has the schema models for posts and users schema.

**The Node Modules:** has the imported node modules required for node project

**The Routes folder:** has two routers Auth and Post where Auth Router handles the registration of a new user to the Piazza API and login for the registered users.

**The Validation folder:** has a validation.js file which incorporates the validation of the user inputs using the JOI module, which is popular for data validation.

**The .env file:** has the Database connector URI (`DB_CONNECTOR`) for connection for the database to the Mongoose and the (`TOKEN_SECRET`) for the user.

**The .gitignore file:** To ignore the node\_modules, npm-debug.logs and .DS\_Store while committing the codebase to GitHub.

**The app.js file:** has the router path for both Auth and Post users, the database connection and the port localhost:3000 server configurations.

**The package-lock.json and package.json:** has the necessary packages required for implementation.

**The ReadME.md file:** has all the commands used for implementing the project

**The verifyToken.js:** handles the verification and authentication of users in Piazza API

## 7.0 Cloud Deployment

### 7.1 Docker

Commands to setup docker in GCP Virtual Machine

```
sudo apt-get install docker.io
sudo adduser docker-user
sudo usermod -aG sudo docker-user
```

Upload piazza application on github repository

```
`git init
git remote add origin
https://YOUR\_GIT\_USERNAME:YOUR\_GIT\_TOKEN@github.com/ViplaviWade/Piazza.git
git add .
git commit -m "code push"
git push -f origin master
```

Switched to vm and used the following steps

```
su - docker-user
git clone
```

[https://YOUR\\_GIT\\_USERNAME:YOUR\\_GIT\\_TOKEN@github.com/heenal66/Piazza.git](https://YOUR_GIT_USERNAME:YOUR_GIT_TOKEN@github.com/heenal66/Piazza.git)

```
cd Piazza
```

Created the below Dockerfile:

```
FROM alpine
RUN apk add --update nodejs npm
COPY . /src
WORKDIR /src
EXPOSE 3000
ENTRYPOINT ["node", "./app.js"]
```

Commands to build docker image

```
docker image build -t piazza-image .
docker container run -d --name piazza --publish 80:3000 piazza-image
```

```
docker-user@docker-vm:~/Piazza$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
NAMES
f6e7dc2f1f6b piazza-image "node ./app.js" 21 minutes ago Up 21 minutes
00/tcp piazza
docker-user@docker-vm:~/Piazza$ []
```

POST | 34.171.53.65/api/login

Params Auth Headers (9) **Body** Pre-req. Tests Settings **Beautify**

raw JSON

```

1 {
2   "email": "olga@gmail.com",
3   "password": "olga123"
4 }

```

## 7.2Kubernetes

Commands to build and push piazza images to DockerHub:

```
docker image build -t heenalj6520/piazza-deployment
docker push wadeviplavi/piazza-app
```

Opened the GCP Cloud Terminal and connected to GCloud Cluster Link:

```
kubectl run piazza-app --image=wadeviplavi/piazza-app
```

Create a deployment yaml file to create Pods:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: Piazza
spec:
  replicas: 10
  selector:
    matchLabels:
      app: piazza-app
  template:
    metadata:
      labels:
        app: piazza-app
    spec:
      containers:
```

```

- name: piazza-backend
  image: wadeviplavi/piazza
  ports:
    - containerPort: 3000

```

---

```

apiVersion: v1
kind: Service
metadata:
  name: piazza-service
spec:
  selector:
    app: piazza-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer

```

The screenshot shows the Google Cloud Platform Kubernetes interface for a deployment named 'piazza'. The top navigation bar includes 'REFRESH', 'EDIT', 'DELETE', 'ACTIONS', 'KUBECTL', 'SHOW INFO PANEL', 'OPERATIONS', and a gear icon.

The main view displays the deployment details for 'piazza' in the 'cloudcomputing' cluster, default namespace, created on Dec 9, 2023, at 5:04:45 PM. It shows no labels set and annotations related to autopilot and deployment.kubernetes.io.

Replica status: 10 updated, 10 ready, 10 available, 0 unavailable. Label selector: app = piazza-app. Update strategy: Rolling update, Max unavailable: 25%, Max surge: 25%. Min time ready before available: 0 s. Progress deadline: 600 s. Revision history limit: 10.

[← Deployment d...](#) [REFRESH](#) [EDIT](#) [DELETE](#) [ACTIONS ▾](#) [KUBECTL ▾](#) [SHOW INFO PANEL](#) [OPERATIONS](#) [?](#)

10GiB 10GiB

**Cluster** [cloudcomputing](#)  
**Namespace** default  
**Labels** No labels set  
**Logs** [Container logs](#), [Audit logs](#)  
**Replicas** 10 updated, 10 ready, 10 available, 0 unavailable  
**Pod specification** Revision 1, containers: [piazza-backend](#)  
**Horizontal Pod** Not configured [CONFIGURE](#)  
**Autoscaler** [?](#) Not configured [CONFIGURE](#)  
**Vertical Pod Autoscaler** Not configured [?](#) [CONFIGURE](#)

**Active revisions**

Revision	Name	Status	Summary	Created on	Pods running/Pods total
1	<a href="#">piazza-5b7f8476d</a>	<span style="color: green;">✓</span> OK	piazza-backend: wadeviplavi/piazza	Dec 9, 2023, 5:04:45 PM	10/10

[← Deployment details](#) [REFRESH](#) [EDIT](#) [DELETE](#) [ACTIONS ▾](#) [KUBECTL ▾](#) [SHOW INFO PANEL](#) [OPERATIONS](#) [?](#) [LEARN](#)

**Active revisions**

Revision	Name	Status	Summary	Created on	Pods running/Pods total
1	<a href="#">piazza-5b7f8476d</a>	<span style="color: green;">✓</span> OK	piazza-backend: wadeviplavi/piazza	Dec 9, 2023, 5:04:45 PM	10/10

**Managed pods**

Revision	Name	Status	Restarts	Created on
1	<a href="#">piazza-5b7f8476d-k5fcf</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:46 PM
1	<a href="#">piazza-5b7f8476d-hpz9t</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:46 PM
1	<a href="#">piazza-5b7f8476d-jf8fq</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-4wkzn</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-zsgjn</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-9pxz5</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-xtzhr</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-bw459</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-l5mh8</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM
1	<a href="#">piazza-5b7f8476d-zvsl_</a>	<span style="color: green;">✓</span> Running	0	Dec 9, 2023, 5:04:47 PM

**Exposing services** [?](#)

Name	Type	Endpoints
<a href="#">piazza-service</a>	Load balancer	<a href="#">35.202.9.65:80</a>

The screenshot shows the Postman application interface. At the top, it displays a POST request to the URL `35.202.9.65/api/login`. Below the URL, there are tabs for Params, Auth, Headers (9), Body (selected), Pre-req., Tests, Settings, and a more options menu. Under the Body tab, there are two dropdowns: 'raw' and 'JSON'. The 'JSON' dropdown is currently selected. To the right of the dropdowns is a 'Beautify' button. The main area contains the JSON payload:

```
1 {  
2   "email": "olga@gmail.com",  
3   "password": "olga123"  
4 }
```

## 7.0 Validation and Authentication

Piazza implements rigorous data validation to ensure data integrity and user security. OAuth v2 is employed for secure user authentication, safeguarding user accounts from unauthorized access.

JWTs are a good way of securely transmitting information between parties because they can be signed, which means the users are identified with the unique tokens.

The async auth function covers the authentication part which authenticates the users with a unique token and maintains the user.id and user.username for the signed in user.

```
JS verifyToken.js > ...
1  const jwt = require('jsonwebtoken')
2
3  const User = require('../models/user')
4
5  async function auth(req, res, next) {
6
7      const token = req.header('auth-token')
8
9      if (!token) {
10          return res.status(401).send({message: "Access denied."})
11      }
12
13      try {
14
15          const verifiedToken = jwt.verify(token, process.env.TOKEN_SECRET)
16
17          const user = await User.findById(verifiedToken._id)
18
19          if(!user) {
20              return res.status(401).json({ error: "Unauthorized. Invalid User"})
21          }
22
23          req.user = {
24              _id: user._id,
25              username: user.username
26          }
27
28          next()
29
30      } catch {
31          res.status(401).json({error: "Unauthorized. Invalid token"})
32      }
33  }
34
35  module.exports = auth
```

## 8.0 API Endpoints:

### 1. CreatePost:

```
router.post('/createPost', verifyToken, async (req, res) => {

  const currentTime = Date.now();
  const expiration_time = req.body.expiration_time;
  const expirationTimeInMilliseconds = currentTime + expiration_time * 60 * 1000;

  const newPost = new Post ({
    post_id:shortid.generate(),
    post_title: req.body.post_title,
    post_topic: req.body.post_topic,
    message: req.body.message,
    status: 'LIVE',
    likes_count: 0,
    dislikes_count: 0,
    comments_count: 0,
    postOwner: req.user.username,
    expiration_time: new Date(expirationTimeInMilliseconds)
  })

  try {
    const savedRecord = await newPost.save()
    res.json(savedRecord)
  } catch (error) {
    res.status(500).json({error: error.message})
  }
})
```

## 2. Update Post:

```
router.put('/updatePost', verifyToken,async (req, res) => {
  const {postId} = req.query
  const {postTitle, postTopic, message} = req.body
  const loggedUser = req.user.username;

  try {
    const postExists = await Post.findOne({post_id: postId})

    if(!postExists) {
      return res.status(404).json({error: "Post not found"})
    }

    if(postExists.postOwner != loggedUser) {
      return res
        .status(403)
        .json({error: "Access Denied. You do not have the permission to modify this post"})
    }

    if(postExists.expiration_time < Date.now) {
      return res.status(403).json({error: "Post expired for update"})
    }

    postExists.title = postTitle || postExists.title
    postExists.post_topic = postTopic || postExists.post_topic
    postExists.message = message || postExists.message

    await postExists.save();
    res.status(200).json({message: "Post updated successfully."})
  }
  catch {
    console.error(error);
    res.status(500).json({ error: "Internal server error"})
  }
})
```

## 3. Perform Actions (Like/ Dislike/ Comment):

```
router.put('/performAction', verifyToken, async (req, res) => {

    const {postId, action} = req.query
    const loggedUser = req.user.username;
    const commentBody = req.body.comment
    const currentTime = Date.now();

    let userAlreadyPerformedAction = false;

    try {
        const postExists = await Post.findOne({post_id: postId})

        if(!postExists) {
            return res.status(404).json({error: "Post not found"})
        }

        if(postExists.postOwner == loggedUser) {
            return res
                .status(403)
                .json({error: "Post owner cannot perform any action on post"})
        }

        if(postExists.expiration_time < currentTime) {
            return res.status(403).json({error: "Post expired for performing any action"})
        }

        if(action == 'like') {
            if (!postExists.like_users || !postExists.like_users.includes(loggedUser)) {
                await Post.findByIdAndUpdate(
                    postExists._id,
                    { $push: { like_users: loggedUser }, $inc: { likes_count: 1 } }
                );
            } else {
                userAlreadyPerformedAction = true;
            }
        }
    }
})
```

```

    if(action == 'dislike') {
      if (!postExists.dislike_users || !postExists.dislike_users.includes(loggedUser)) {
        await Post.findByIdAndUpdate(
          postExists._id,
          { $push: { dislike_users: loggedUser }, $inc: { dislikes_count: 1 } }
        );
      } else {
        userAlreadyPerformedAction = true;
      }
    }

    if(action == 'comment') {
      const newComment = {
        commentId: shortid.generate(),
        username: loggedUser,
        comment: commentBody
      }
      await Post.findByIdAndUpdate(
        postExists._id,
        { $push: { comments: newComment }, $inc: { comments_count: 1 } }
      );
      return res.status(200).json({ message: "Comment posted successfully." })
    }

    if (userAlreadyPerformedAction) {
      return res.status(400).json({ message: `User has already ${action === 'like' ? 'liked' : 'disliked'} the post` });
    }

    res.json({ message: `User has ${action === 'like' ? 'liked' : 'disliked'} the post` });
    if (userAlreadyPerformedAction) {
      return res.status(400).json({ error: `User has already ${action === 'like' ? 'liked' : 'disliked'} the post` });
    }
  }
} catch (error) {
  console.error(error);
}

```

#### 4. Get a Post by ID:

```

router.get('/getpostById', verifyToken, async (req, res) => {

  const {postId} = req.query

  try {
    const postExists = await Post.findOne({post_id: postId})

    if(!postExists) {
      return res.status(404).json({error: "Post not found"})
    }
    res.send(postExists)
  } catch (error) {
    res.status(400).send({message: error})
  }
})

```

## 5. Get Most Active Post:

```
router.get('/getMostActivePost', verifyToken, async (req, res) => {
    const {topic} = req.query

    try {
        const postExists = await Post.aggregate([
            { $match: { post_topic: topic } },
            {
                $addFields: {
                    totalInteractions: { $add: ['$likes_count', '$dislikes_count'] }
                },
            },
            { $sort: { totalInteractions: -1 } },
            { $limit: 1 },
        ]);

        if (!postExists || postExists.length === 0) {
            return res.status(404).json({error: `No active posts found for the topic: ${topic}`})
        }

        res.json(postExists[0]);
    } catch (err) {
        res.status(500).json({ error: err });
    }
})
```

## 6. Get Post Per Topic:

```
router.get('/getPostPerTopic', verifyToken, async (req, res) => {
    const {topic} = req.query

    try {
        const postExists = await Post.find({post_topic: topic})
        console.log(postExists)
        if(!postExists) {
            return res.status(404).json({error: "Post not found"})
        }

        res.send(postExists)
    } catch(error) {
        res.status(400).send({message: error})
    }
})
```

## 7. Get Expired Posts:

```
router.get('/getExpiredPosts', verifyToken, async (req, res) => {
    const { topic } = req.query;
    const currentTime = Date.now();

    try {
        const postExists = await Post.find({
            post_topic: topic,
            expiration_time: { $lt: currentTime }
        }).sort({ timestamp: -1 }).exec();

        if (!postExists || postExists.length === 0) {
            return res.status(404).json({ error: `No expired posts found for the topic: ${topic}` });
        }

        res.send(postExists)
    } catch (error) {
        res.status(500).json({ error: "Internal server error" })
    }
})
```

## 8. Get All Posts:

```
router.get('/getAllPosts', verifyToken, async (req, res) => {
    try {
        const posts = await Post.find()
        console.log("Posts are : ", posts)
        res.send(posts)
    } catch (err) {
        res.status(400).send({ message: err })
    }
})
```

## 9.0 Conclusion

In conclusion, Piazza offers a feature-rich social media experience with secure user authentication, interactive post functionalities, and efficient data storage using MongoDB. The implementation of OAuth v2 ensures a secure environment for users to engage and share content.