



Cloud Computing Coursework

Development of a Cloud Software as a Service

by Oboni Anower (13923163)

MSc Advanced Computing

School of Computing and Mathematical Sciences

Birkbeck, University of London

10th December 2023

Contents

Student Declaration	3
Introduction	4
1. Installation and Deployment.....	4
1.1. Virtual Machine Installation	4
1.2. Software Packages Installations	4
1.3. Utilization of GitHub.....	5
2. Folder Structure	6
3. Database Design.....	8
3.1. User	8
3.2. Post.....	8
4. API Endpoints	9
5. User Authentication and Verification Functionalities	10
5.1. User Validations	10
5.2. User Registration.....	10
5.3. Request Access on Client-Side: Sign-in.....	11
5.4. Access Protected Resource on Server Side	12
6. Deployment of Piazza into Docker	13
7. Testing Piazza	14
8. Deploying Piazza to Kubernetes.....	32
9. Area of Improvements & Future Work	34
Conclusion.....	34
References.....	35
Appendices.....	36
Appendix A.....	36
Appendix B.....	36
Appendix C	37
Appendix D.....	38
Appendix E	39
Appendix F	40

Student Declaration

This Report is documented and completed solely by the student.

Any information from outside source has been referenced.

Grammarly has been used for spelling inspections ONLY.

Introduction

This technical report documents all the configurations, installation process, code workflow, and test scenarios of the working application.

The project revolves around building software named Piazza as a service, which is an interactive platform similar to Twitter. It is built with NodeJS, a JavaScript run-time environment. In Piazza, a user can register an account, and interact with other users over the platform. The users must go through an identity verification process before accessing sensitive information and Piazza's services and resources. To do that, an authorization mechanism like the OAuth2.0 protocol is used.

The software is then hosted on a cloud infrastructure to enable access to end users over the internet. A third-party cloud provider like Google Cloud Platform is used in this case. Inside GCP, Piazza and all its dependencies are containerized to run reliably and efficiently from the virtual machine's environment to the local browser. An Ubuntu virtual machine is used for deploying the workloads in the public cloud because of its performance quality and consistency (Mavrogiannopoulos, 2021).

1. Installation and Deployment

1.1. Virtual Machine Installation

This project is initiated by setting up a virtual machine on the **Google Cloud Platform** named "*coursework-vm*", to replicate an Ubuntu environment. It is a popular open-source Linux OS which is known for its user-friendliness. More details about this VM configuration can be found in Appendix A and B. More detail about this VM utilization is discussed in section 7.

Configuration of the Virtual machine:

Name: coursework-vm

Working External IP: 34.113.238.130

Zone: us-central1-a

Machine type: e2-medium (2 vCPU, 1 core, 4 GB memory)

Image: ubuntu-2004-focal-v20231101

Storage Size: 25 GB

HTTP traffic: On

HTTPS traffic: On

1.2. Software Packages Installations

On the other hand, a project directory "*coursework-app*" is created in the local machine's Visual Studio Code IDE. Inside the project terminal, npm is initiated to install its corresponding packages using "*npm install*". These packages are utilized throughout this project.

NPM is a default package manager for Node.js which offers thousands of packages to reuse for web application development (Goswami, Gupta, Li, Meng, & Yao, 2020). All these dependencies are stored in an auto-generated file named "*package.json*". Refer to Appendix C for a detailed overview of the packages and versions. Here are the main packages used for designing Piazza:

```
>npm install express nodemon mongoose body-parser dotenv bcryptjs joi jsonwebtoken
```

npm packages include:

Package	Purpose
express	Web application development framework to build secured web applications.
nodemon	monitors changes in the source and restarts the server automatically when required.
mongoose	A driver to connect to the MongoDB cluster.
dotenv	allows to separate sensitive data from the source code.
body-parser	To parse body data as JSON objects.
joi	To enforce user validations.
bcryptjs	for encrypting/decrypting passwords.
jsonwebtoken	to generate authentication tokens.

Table 1: Node Package Management (NPM)

1.3. Utilization of GitHub

After the workspace in the local machine is ready, a new private repository named “[coursework-SaaS](#)” is created on GitHub. The https URL under this repository is used to clone this into the local computer. This cloning establishes a synchronization between the local project workspace and the GitHub repo.

coursework-SaaS Repository URL:

https://0anower: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@github.com/0anower/coursework-SaaS.git



This URL comprises of the user’s git username, git access token, and repository address. The purpose of the git access token is to identify users and permit access to GitHub Enterprise Server while interacting with GitHub's API or via command-line operations.

Once the codebase of the Piazza app is ready and tested in localhost server, it is then cloned with GitHub. The following commands are used to deploy the project:

```
$ git -version
$ git init
$ git remote add origin
https://0anower:ghp\_50zdfSMQpMb0GQudK7S2aD188j18dm3jY2wJ@github.com/0anower/coursework-SaaS.git
$ git add .
$ git commit -m "Push Piazza to GitHub!"
$ git push -f origin main
```

```

anowe@OBONI-LAPTOP MINGW64 ~/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSEWORK/coursework 2023-SaaS-0
app (main)
$ git --version
git version 2.42.0.windows.2

anowe@OBONI-LAPTOP MINGW64 ~/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSEWORK/coursework 2023-SaaS-0
app (main)
$ git init
Reinitialized existing Git repository in C:/Users/anowe/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSE
as-Oboni Anower/coursework-app/.git/

anowe@OBONI-LAPTOP MINGW64 ~/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSEWORK/coursework 2023-SaaS-0
app (main)
$ git remote add origin https://0anower:ghp_50zdfSMQpMb0GQudK7S2aD188j18dm3jY2wD@github.com/0anower/coursework-SaaS.git
error: remote origin already exists.

anowe@OBONI-LAPTOP MINGW64 ~/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSEWORK/coursework 2023-SaaS-0
app (main)
$ git add .

anowe@OBONI-LAPTOP MINGW64 ~/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSEWORK/coursework 2023-SaaS-0
app (main)
$ git commit -m "Push piazza to Github!"
[main 30752f8] Push piazza to Github!
10 files changed, 467 insertions(+), 54 deletions(-)
create mode 100644 routes/browseRoute.js
create mode 100644 routes/highestRoute.js
create mode 100644 routes/home.js
create mode 100644 routes/interactions.js
anowe@OBONI-LAPTOP MINGW64 ~/OneDrive/Documents/STUDY MATERIAL/BBK_UoL/Cloud Computing/COURSEWORK/coursework 2023-SaaS-0
app (main)
$ git push -f origin main
Enumerating objects: 2433, done.
Counting objects: 100% (2433/2433), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2347/2347), done.
Writing objects: 100% (2433/2433), 3.74 MiB | 1.25 MiB/s, done.
Total 2433 (delta 344), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (344/344), done.
To https://github.com/0anower/coursework-SaaS.git
+ 82a1a6d...30752f8 main -> main (forced update)

```

Image 1: GitHub cloning

Now this project successfully cloned GitHub's server, it can be deployed into the virtual machine's docker to containerise with some simple git commands. This will be discussed further in section 7.

2. Folder Structure

The structure of this SaaS project is crafted with a logical hierarchy to make it easier to navigate the codebases. Having different files for different purpose facilitate readability, scalability, and collaboration within the project. The main working directory is divided into various sub-directories, including source code files for models, different routes, and user validations.

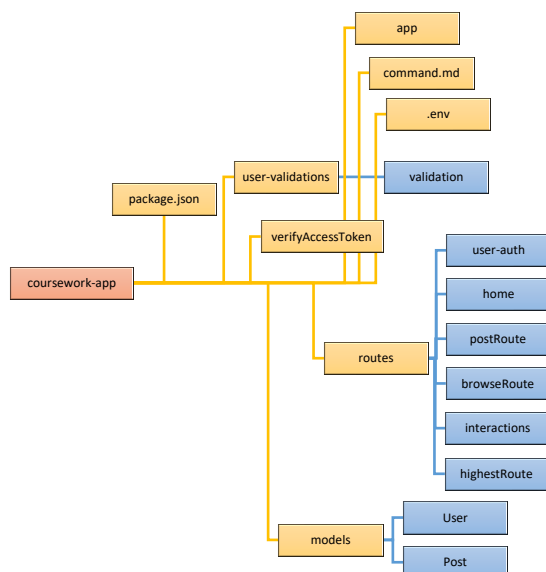


Figure 1: Folder Structure

Figure 1 illustrates a concise layout of the project folder, showcasing the mother folder and all her surrounding resources to represent different functionalities. Here is a breakdown of the purposes they serve, starting with the main directory:

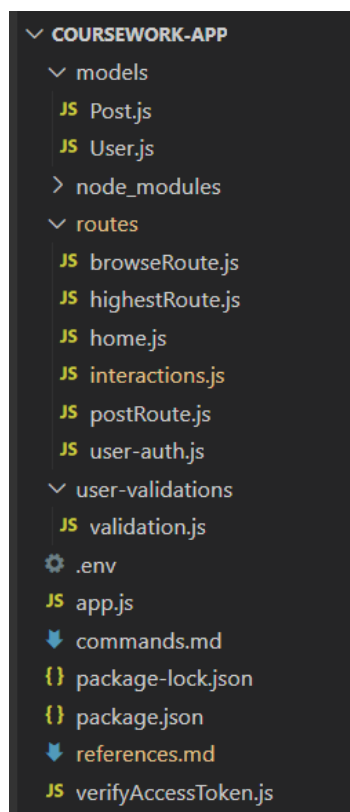


Image 2: Folders

coursework-app: This is the main workspace for Piazza development. All the necessary configurations and packages are only applied to this specific folder.

package. json: It is an auto-generated JSON file when `<npm init>` command is run in the terminal. It lists all the package names and their versions that this project depends on. Since it is a dynamic process, it automatically updates any new installations or modifications to the packages.

app.js: It is the main server file for Piazza and Node will use this entry point (Kris, 2021) to start the app. All the major API end-point routes and main database connectivity are also listed in this file.

command.md: A .md file is used to keep important information about the development process. For example, the package installation commands, git repository URLs, and Git access tokens are archived in this file.

.env: The environment variable file is used to store the main database connection string and the user access token in variables. These sensitive data are recorded outside the source code area to keep them secret and secure.

verifyAccessToken: It is used for the authentication process. It verifies the access token before allowing the user to access the platform.

models: This folder contains database models that allow to establish an interface between a mongoose schema and MongoDB collection. A mongoose schema is used to define attributes and set their data types inside a database collection. For this project, two types of models are utilized: User model and Post Model

routes: This folder is to organise and manage different routes or endpoints of Piazza application. All the REST API end-point routes such as posting, browsing, and interaction are listed in this folder. Making this file helps to group routes by functionality. For example, all the liking, disliking, and commenting endpoints are listed in one location. Whereas endpoints to browse posts are listed in a separate route. Having a separate file makes it easier to scale by adding or modifying routes without making the workflow too complicated.

3. Database Design

Inside mongoDB, a database is created and named Piazza. Two collections are created inside this database named 'users' and 'posts'. Once a model for each of these collections is created, data can be inserted into them with a POST HTTP method.

```
// connects to mongoDB cluster
mongoose.connect(process.env.DB_CONNECTOR, {
  useNewUrlParser: true
})
.then(() => {
  console.log('mongoDB is now connected')
})
.catch((error) => {
  console.error('Error connecting to MongoDB:', error)
})
```

The code above helps to establish a connectivity between the workspace and the MongoDB cluster using the Mongoose library and its connect() method. The connection string named *DB_CONNECTOR* is stored in a .env file to keep it secured.

3.1. User

This model is designed to store the user details in a Mongoose schema. The attributes include the name of a user, email address, and password. Each of these attributes records information provided by the user during an account registration using HTTP POST request. This Mongoose schema is exported to the user's collection in piazza database. When the user tries to sign-into the platform, the entered data are compared with data inside the database to authenticate the user.

User collections: userSchema

- **_id**: auto generated; user identifier.
- **username**: stores username during sign-up.
- **email**: stores during sign-up; used for user authentication at sign-in.
- **password**: stores encrypted during sign-up; decrypts during sign-in process.

3.2. Post

Similar to a User model, a Post model stores details for posting and commenting. In this model, two different schemas are created: one is for posting in the news feed. This includes attributes like post owner's name, title, body of the post, registration and expiration time, duration, and status. It also contains a topic attribute that specifies four topics and users are restricted to post from only four of them. The user can choose more than one of these topics too. These details are inserted into the MongoDB database during a message posting request or interactive requests such as liking or disliking a post.

Post Collection: postSchema

- **_id**: auto generated; post identifier.
- **postOwnerName**: generated from authorised user's username during sign-in.
- **postTopic**: stores fixed array values; only returns from them.
- **postTitle**: title of post in text.
- **messageBody**: post content in text.
- **postRegistrationTime**: auto generates system time of post request.
- **postDuration**: in minutes; user sets this time.
- **postExpirationTime**: autogenerates expiration time based on reg-time and time-duration.

- **status:** Live until expiration time; changed to Expired after duration over.
- **likes:** number of likes.
- **likedBy:** array to store userid who liked.
- **dislikes:** number of dislikes.
- **dislikedBy:** array to store userid who disliked.

The other schema is for commenting. This includes the name of the commenter, the comment body, and the commenting time. Data are inserted into the database when a comment request is called.

Comment collection: commentSchema

- **commenter:** generated from authorized user's username during sign-in.
- **comment:** comment body text.
- **commentDate:** auto-generates from system time.

4. API Endpoints

File: user-auth.js

User registration: POST /api/user/sign-up

User-access: POST /api/user/sign-in

File: home.js

Piazza homepage: GET /api/home

File: postRoute.js

Message posting: POST /api/home/post

File: browseRoute.js

Browse posts: GET /api/home/allposts

Retrieve post: GET /api/home/allposts/:postId

Query: GET /api/home/allposts?key1=value1&key2=value2&..

Edit post: PATCH /api/home/allposts/:postId

Delete post: DELETE /api/home/allposts/:postId

File: interactions.js

Like: POST /api/home/allposts/:postId/like

Dislike: POST /api/home/allposts/:postId/dislike

Comment: POST /api/home/allposts/:postId/comment

File: highestRoute.js

post with max interaction, filter topic and status: GET api/home/allposts/maxint/:postTopic/:status

5. User Authentication and Verification Functionalities

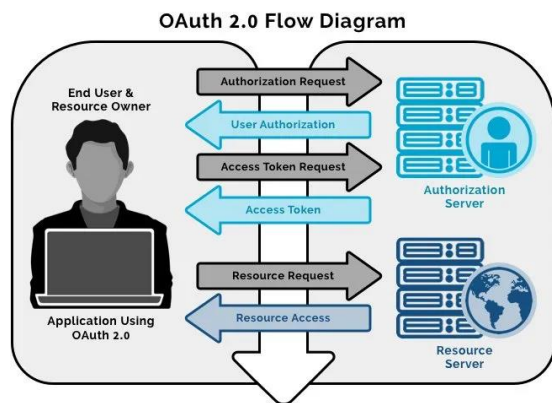


Image 3: OAuth2 Flow diagram

To gain authorised access to the API, the OAuth2 protocol has been implemented. This procedure is adapted so a server can access resources on behalf of users without getting access to their credentials. The most common and widely used mechanism for token generation is JWT (JSON web token), which holds authorisation information. Image 3 shows a work process of OAuth2. (Dineshchandgr, 2022)

5.1. User Validations

File Location: coursework-app/validations/validation.js

Before creating the functionalities, it is important to implement a validation constraint that follows a specific schema format. This will prevent users from passing invalid or incorrect data during the registration and sign-in process. To enforce this validation, a *joi library* is imported. Next, a validation schema object is designed to describe the composition of the working data and is used to validate the data against that schema object (Banimahd, 2023).

In this project, two different functions are created with the same constraints as the User model. However, the sign-up validation contains a username, whereas the sign-in object only has email and password. These functions are then exported to the 'user-auth' route.

5.2. User Registration

File Location: coursework-app/routes/user-auth.js

The previously created *signupValidation* function is imported to verify the registration info entered by the user, including an email address and password. If all the info matches the specification, it then checks if the email address already exists in the system. In case if it does, the process will be terminated.

If the email is unique, the system will proceed to the next stage and hide the password with an encryption, so it will only be known to the user but hidden from the database. To do that, a *bcrypt* library is used, which is basically a hashing algorithm that mixes random strings called *salt* with the password (Mohsin, 2023).

Finally, the username, email address, and hashed password are inserted and saved in MongoDB's 'users' database.

API end-point: '[api/user/sign-up](#)'

5.3. Request Access on Client-Side: Sign-in

File Location: coursework-app/routes/user-auth.js

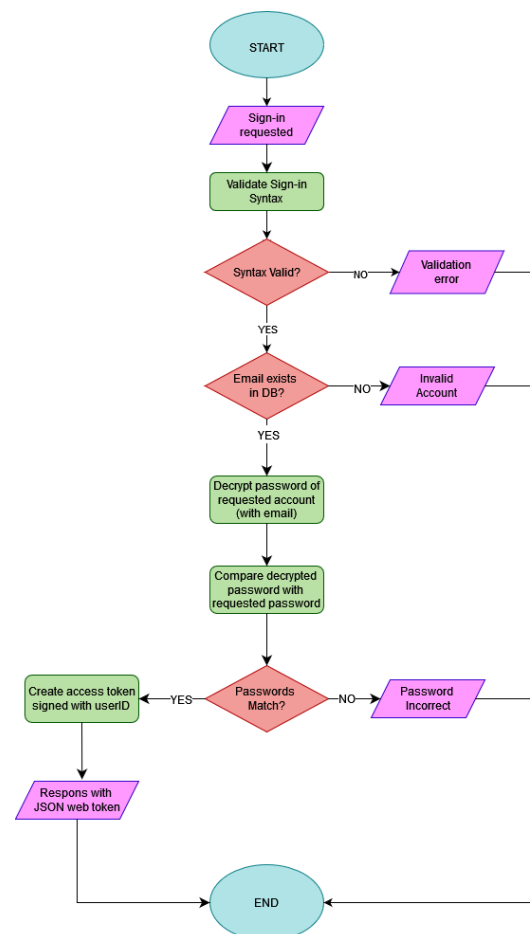


Figure 2: Workflow of the Sign-in process

Figure 2 (Anower, 2023) shows a complete workflow of the process that allows users to sign into the Piazza application. This step is crucial because it is the gateway to Piazza's interactive platform. To validate the account-holder's information, the *signinValidation* function created in 5.1.1 is called in. It shows that the client requests a validity check to the server. If data is inserted with correct syntax, i.e., at least 6 characters in the password, it compares the entered email address with the email addresses recorded in the database. Once an email match is found, it decrypts the password of that email in the database and compares it with the password entered by the user. If both passwords are an exact match, then the user is proven to be authentic. Now the authorisation is granted by the server to the client.

Now, the client requests for an access token from the server. Since the request is coming from an authorised user, the server generates access token using *jsonwebtoken* library and SECRET_ACCESS_TOKEN (stored away in .env file). This token is base64url encoded and digitally signed with the authentic user's ID (Barros, 2019). The purpose of this token will be discussed further in the next section.

API end-point: `/user/ sign-in'`

5.4. Access Protected Resource on Server Side

File Location: coursework-app/verifyAccessToken.js

Here, the client requests access to the resource using the token.

An authentication function is designed to control the authentication process for the JSON web token using *jsonwebtoken* library. Next, a key called “auth-token” is created and the token retrieved during sign-in process is entered as the value. Then, a condition applied to check if the header key is active with correct token value. The library compares the header value with the Secret Access Token in .env file using a `verify()` method. If token is absent or incorrect, access to Piazza will be denied. An access will be granted to the client for the correct token.

This JSON web token is passed every time a user makes an API call (Barros, 2019), and the application will verify the token every time before processing the API calls.

The work process and code implementations from sections 5.1-5.4 are adapted from Cloud Computing Lab 4 (Stelios, 2022).

6. Deployment of Piazza into Docker

Now that the app is ready, it can be deployed into GCP's "coursework-vm" from GitHub repository. To do this, docker is installed in the Ubuntu virtual machine. Then, a super user named docker-u is created, where the project repository from GitHub can be cloned.

```
docker-u@coursework-vm:~$ pwd
/home/docker-u
docker-u@coursework-vm:~$ git clone --branch main https://0anower:ghp_50zdfSMQpMb0GQudK7S2aD188j18dm3
jY2wJ@github.com:0anower/coursework-SaaS.git
Cloning into 'coursework-SaaS'...
remote: Enumerating objects: 2433, done.
remote: Counting objects: 100% (2433/2433), done.
remote: Compressing objects: 100% (2003/2003), done.
remote: Total 2433 (delta 344), reused 2433 (delta 344), pack-reused 0
Receiving objects: 100% (2433/2433), 3.74 MiB | 9.57 MiB/s, done.
Resolving deltas: 100% (344/344), done.
docker-u@coursework-vm:~$ ls
coursework-SaaS
docker-u@coursework-vm:~$ cd coursework-SaaS
docker-u@coursework-vm:~/coursework-SaaS$ ls
app.js      models      package-lock.json  routes      verifyAccessToken.js
commands.md node_modules package.json        user-validations
```

Image 4: Cloning GitHub Repo into the virtual machine

Inside this repository, a Dockerfile is created, where a set of instructions (Sotiriadis, 2023) are added that are used to create a Docker image to run a Node.js application. This file uses Alpine Linux as the base image, which includes source code, libraries, tools, and other core dependencies (Charboneau, 2022). For example, to utilize Piazza, will set up a Node.js environment with NPM installed.

Content inside Dockerfile:

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

This Dockerfile is saved in the repository inside the virtual machine. The next step is to create a docker image called "piazza-app:1", which is the first version of this image. Once the image is built, a detached container called "my-piazza-web" is created and run. This container maps the VM's port 8080 to the newly created container's port 3000. Now this container can host the Piazza Node.js application using the previously built piazza-app:1 image.

```
docker-u@coursework-vm:~/coursework-SaaS$ docker ps -all
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
docker-u@coursework-vm:~/coursework-SaaS$ docker container run -d --name my-piazza-web --publish 8080:3000 piazza-app:1
e3b42103cc58a0b5e9d5e12b05b6742043322741bed6b872060ab650a43af3ea
docker-u@coursework-vm:~/coursework-SaaS$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
piazza-app    1         f56d4d973174   2 minutes ago  94MB
alpine        latest    b541f2080109   5 days ago    7.34MB
docker-u@coursework-vm:~/coursework-SaaS$ docker ps -all
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
e3b42103cc58   piazza-app:1 "node ./app.js"         25 seconds ago Up 24 seconds 0.0.0.0:8080->3000/tcp, :::8080->3000/tcp my-piazza-web
```

Image 5: Containerisation to host Piazza app

Now that this container is active, the Piazza app can be accessed as a service with a web browser, or POSTMAN using this virtual machine's external IP Address.

The entire process is adapted from Cloud Computing labs 5.1 and 5.2 (Sotiriadis, 2023).

7. Testing Piazza

The test cases for the Piazza app as a service is done with the VM's external IP address: 34.113.238.130.

Each step will be explained with screenshots, HTTP calls, API end-points, and Action points from the specification. For the user details, the same names are used from the coursework specification: *Olga*, *Nick*, *Mary*, and *Nestor*.

TC1: This scenario shows the user account registrations after using the HTTP GET method. Here, the passwords are encrypted using bcrypt library.

File: routes/user-auth.js

API end-point: '/api/user/sign-up'

HTTP method: POST

The image displays four sequential screenshots of a REST client interface, each showing a POST request to the endpoint `34.133.238.130:8080/api/user/sign-up` and its corresponding JSON response. The responses show user registration details for four different users: Olga, Nick, Mary, and Nestor. Each response includes a unique ID, a timestamp for the date joined, and a bcrypt-hashed password.

Request 1 (Olga):

```
POST 34.133.238.130:8080/api/user/sign-up
Body: {
  "username": "Olga",
  "email": "olga@cloud.com",
  "password": "123456"
}
```

Response 1 (Olga):

```
{
  "username": "Olga",
  "email": "olga@cloud.com",
  "password": "$2a$05$2EKLCE7CINQVh1KN3Kh2gu1A8QkTawZ11R08kPAVt2Ke8h1tGzba",
  "id": "6574d6416ad5de0bda5bdfc9",
  "date_joined": "2023-12-09T21:04:01.511Z",
  "v": 0
}
```

Request 2 (Nick):

```
POST 34.133.238.130:8080/api/user/sign-up
Body: {
  "username": "Nick",
  "email": "nick@cloud.com",
  "password": "1234567"
}
```

Response 2 (Nick):

```
{
  "username": "Nick",
  "email": "nick@cloud.com",
  "password": "$2a$05$KbkveamewzaDNZfKbDceacJTsePA3P1N1z1yJ3c1HMLtb5L6yz5",
  "id": "6574d6746ad5de0bda5bdfc3",
  "date_joined": "2023-12-09T21:04:52.009Z",
  "v": 0
}
```

Request 3 (Mary):

```
POST 34.133.238.130:8080/api/user/sign-up
Body: {
  "username": "Mary",
  "email": "mary@cloud.com",
  "password": "1234566"
}
```

Response 3 (Mary):

```
{
  "username": "Mary",
  "email": "mary@cloud.com",
  "password": "$2a$05$N5Sc1VeyLECUzmx691ZhXu1V6wKjVvGU.1q5o4gf1m2.TK/h3oIRi",
  "id": "6574d6a76ad5de0bda5bdfc6",
  "date_joined": "2023-12-09T21:05:43.714Z",
  "v": 0
}
```

Request 4 (Nestor):

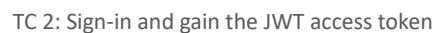
```
POST 34.133.238.130:8080/api/user/sign-up
Body: {
  "username": "Nestor",
  "email": "nestor@cloud.com",
  "password": "123456"
}
```

Response 4 (Nestor):

```
{
  "username": "Nestor",
  "email": "nestor@cloud.com",
  "password": "$2a$05$N3nJqvz1SCWN0nM4yVvjd11jhAKNaL7Vqj3ZC1Q8CvouD2uA.W",
  "id": "6574d6ca6ad5de0bda5bdfc9",
  "date_joined": "2023-12-09T21:06:18.122Z",
  "v": 0
}
```

TC 1: User registrations

Action point: 1
File: routes/user-auth.js
API Route: '/api/user/sign-in'
HTTP method: POST



Action point: 1
File: routes/home.js
API end-point: 'api/home'
HTTP method: GET

HTTP **GET** http://34.133.238.130:8080/api/home

Headers **8 hidden**

Key	Value	...	Bulk Edit	Presets
auth-token	eyJhb...			
Key	Value	Description		

Body **JSON**

```

1 {
2   "message": "Access denied"
3 }

```

TC 3: Unauthorised request by Olga

TC4: Here, Olga posts a new message on the platform with POST method. For this, the auth-token key at the header is turned ON. This token proves that the user is valid and is authorised to access the Piazza API services. A time has been set for this post's duration, which is 500 minutes. The registration time will be generated automatically from the system/s time and the expiration time will be 500 min after the registration time. Once the expiration time crosses the time limit, it will not accept any further interactions from users, such as likes, dislikes, and comments.

Action point: 2.

File: routes/postRoute.js

API end-point: 'api/home/post'

HTTP method: POST

POST 34.133.238.130:8080 POST http://34.133.238.130

HTTP **POST** http://34.133.238.130:8080/api/home/post

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

raw **JSON** Beautify

```

1 {
2   "postOwnerName": "Olga",
3   "postTopic": "Tech",
4   "postTitle": "Post about tech",
5   "messageBody": "This is Olga's first post on Tech topic",
6   "postRegistrationTime": "2023-12-09T21:20:24.881Z",
7   "postDuration": 500
8 }

```

Body **JSON**

```

1 {
2   "postOwnerName": "Olga",
3   "postTopic": "Tech",
4   "postTitle": "Post about tech",
5   "messageBody": "This is Olga's first post on Tech topic",
6   "postRegistrationTime": "2023-12-09T21:20:24.881Z",
7   "postDuration": 500,
8   "postExpirationTime": "2023-12-10T05:40:24.881Z",
9   "status": "Live",
10  "likes": 0,
11  "likedBy": [],
12  "dislikes": 0,
13  "dislikedBy": [],
14  "_id": "6574da186ad5de0bda5bdf1",
15  "comments": [],
16  "__v": 0
17 }

```

TC 4: Olga posts with a token and set time

TC5: Same process as TC4 for this test case. In Nick's case, the duration is set to 600 minutes.

Action point: 2.

File: routes/postRoute.js

API end-point: 'api/home/post'

HTTP method: POST

TC6: Same process as TC4 and TC 5. For this user, the duration is set to 400 min.

Action point: 2.

File: routes/postRoute.js

API end-point: 'api/home/post'

HTTP method: POST

The screenshot displays a REST client interface with a POST request to `http://34.133.238.130:8080/api/home/post`. The request body is a JSON object with the following fields: `postOwnerName` (empty string), `postTopic` ("Tech"), `postTitle` ("a post about tech"), `messageBody` ("This is Mary's first post on Tech topic"), and `postDuration` (400). The response body is a JSON object with the following fields: `postOwnerName` ("Mary"), `postTopic` ("Tech"), `postTitle` ("a post about tech"), `messageBody` ("This is Mary's first post on Tech topic"), `postRegistrationTime` ("2023-12-09T21:40:23.080Z"), `postDuration` (400), `postExpirationTime` ("2023-12-10T04:20:23.080Z"), `status` ("Live"), `likes` (0), `likedBy` ([]), `dislikes` (0), `dislikedBy` ([]), `_id` ("6574dec76ad5de0bda5bdf7"), `comments` ([]), and `__v` (0).

```
POST http://34.133.238.130:8080/api/home/post

{
  "postOwnerName": "",
  "postTopic": "Tech",
  "postTitle": "a post about tech",
  "messageBody": "This is Mary's first post on Tech topic",
  "postDuration": 400
}
```

```
{
  "postOwnerName": "Mary",
  "postTopic": "Tech",
  "postTitle": "a post about tech",
  "messageBody": "This is Mary's first post on Tech topic",
  "postRegistrationTime": "2023-12-09T21:40:23.080Z",
  "postDuration": 400,
  "postExpirationTime": "2023-12-10T04:20:23.080Z",
  "status": "Live",
  "likes": 0,
  "likedBy": [],
  "dislikes": 0,
  "dislikedBy": [],
  "_id": "6574dec76ad5de0bda5bdf7",
  "comments": [],
  "__v": 0
}
```

TC 6: Mary posts with a token and set time

TC7: in this case, Both Nick and Olga will browse all the posts that are currently up on the platform. Here, at the top, the authorised user's name is generated, and the posts by active users afterwards. To do that, both users request Get method. At this point, there is no likes, dislikes, or comments present in any post. Both the users can see only 3 posts at the moment.

Action point: 3

File: routes/browseRoute.js

API end-point: 'api/home/allposts'

HTTP method: GET

The first screenshot shows the REST client interface with the URL `http://34.133.238.130:8080/api/home/allposts` and the method `GET`. The response body is displayed in JSON format, showing the user `Nick` and a list of posts. The response is as follows:

```
1 {
2   "loggedInUser": "Nick",
3   "totalPosts": 3,
4   "posts": [
5     {
6       "_id": "6574dec76ad5de8bda5bdff7",
7       "postOwnerName": "Mary",
8       "postTopic": "Tech",
9       "postTitle": "a post about tech",
10      "messageBody": "This is Mary's first post on Tech topic",
11      "postRegistrationTime": "2023-12-09T21:40:23.080Z",
12      "postDuration": 400,
13      "postExpirationTime": "2023-12-10T04:20:23.080Z",
14      "status": "Live",
15      "likes": 0,
16      "likedBy": [],
17      "dislikes": 0,
18      "dislikedBy": [],
19      "comments": [],
20      "__v": 0
21    },
22    {
23      "_id": "6574daa46ad5de8bda5bdff5",
24      "postOwnerName": "Nick",
25      "postTopic": "Tech",
26      "postTitle": "another post about tech",
27      "messageBody": "This is Nick's first post on Tech topic",
28      "postRegistrationTime": "2023-12-09T21:22:44.164Z",
29      "postDuration": 600,
30      "postExpirationTime": "2023-12-10T07:22:44.164Z",
31      "status": "Live",
32      "likes": 0,
33      "likedBy": [],
34      "dislikes": 0,
35      "dislikedBy": [],
36      "comments": [],
37      "__v": 0
38    },
39    {
40      "_id": "6574de1b6ad5de8bda5bdff1",
41      "postOwnerName": "Olga",
42      "postTopic": "Tech",
43      "postTitle": "Post about tech",
44      "messageBody": "This is Olga's first post on Tech topic",
45      "postRegistrationTime": "2023-12-09T21:20:24.881Z",
46      "postDuration": 600,
47      "postExpirationTime": "2023-12-10T05:40:24.881Z",
48      "status": "Live",
49      "likes": 0,
50      "likedBy": [],
51      "dislikes": 0,
52      "dislikedBy": [],
53      "comments": [],
54      "__v": 0
55    }
56  ]
57 }
```

The second screenshot shows the REST client interface with the same URL and method. The response body is displayed in JSON format, showing the user `Olga` and a list of posts. The response is as follows:

```
27 {
28   "messageBody": "This is Nick's first post on Tech topic",
29   "postRegistrationTime": "2023-12-09T21:22:44.164Z",
30   "postDuration": 600,
31   "postExpirationTime": "2023-12-10T07:22:44.164Z",
32   "status": "Live",
33   "likes": 0,
34   "likedBy": [],
35   "dislikes": 0,
36   "dislikedBy": [],
37   "comments": [],
38   "__v": 0
39 },
40 {
41   "_id": "6574de1b6ad5de8bda5bdff1",
42   "postOwnerName": "Olga",
43   "postTopic": "Tech",
44   "postTitle": "Post about tech",
45   "messageBody": "This is Olga's first post on Tech topic",
46   "postRegistrationTime": "2023-12-09T21:20:24.881Z",
47   "postDuration": 600,
48   "postExpirationTime": "2023-12-10T05:40:24.881Z",
49   "status": "Live",
50   "likes": 0,
51   "likedBy": [],
52   "dislikes": 0,
53   "dislikedBy": [],
54   "comments": [],
55   "__v": 0
56 }
```

TC 7.1: Nick browsing all the posts

http://34.133.238.130:8080/api/home/allposts

GET http://34.133.238.130:8080/api/home/allposts

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

Headers 6 hidden

Key	Value	Description
auth-token	eyJhbGciOiJIUzI1NiIsInR5cGEiOi...	

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "loggedInUser": "Olga",
3   "totalPosts": 3,
4   "posts": [
5     {
6       "_id": "6574dec76ad5de9bda5bd1f7",
7       "postOwnerName": "Mary",
8       "postTopic": "Tech",
9       "postTitle": "a post about tech",
10      "messageBody": "This is Mary's first post on Tech topic",
11      "postRegistrationTime": "2023-12-09T21:40:23.686Z",
12      "postDuration": 400,
13      "postExpirationTime": "2023-12-10T04:20:23.686Z",
14      "status": "Live",
15      "likes": 0,
16      "likedBy": [],
17      "dislikes": 0,
18      "dislikedBy": [],
19      "comments": [],
20      "_v": 0
21    },
22    {
23      "_id": "6574daa46ad5de9bda5bd1d5",
24      "postOwnerName": "Nick",
25      "postTopic": "Tech",
26      "postTitle": "another post about tech",
27      "messageBody": "This is Nick's first post on Tech topic",
28      "postRegistrationTime": "2023-12-09T21:22:44.164Z",
29      "postDuration": 600
30    }
31  ]
32 }
```

http://34.133.238.130:8080/api/home/allposts

GET http://34.133.238.130:8080/api/home/allposts

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

Headers 6 hidden

Key	Value	Description
auth-token	eyJhbGciOiJIUzI1NiIsInR5cGEiOi...	

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
29 "postDuration": 600,
30 "postExpirationTime": "2023-12-10T07:22:44.164Z",
31 "status": "Live",
32 "likes": 0,
33 "likedBy": [],
34 "dislikes": 0,
35 "dislikedBy": [],
36 "comments": [],
37 "_v": 0
38 },
39 {
40   "_id": "6574da16ad5de9bda5bd1d1",
41   "postOwnerName": "Olga",
42   "postTopic": "Tech",
43   "postTitle": "Post about tech",
44   "messageBody": "This is Olga's first post on Tech topic",
45   "postRegistrationTime": "2023-12-09T21:20:24.881Z",
46   "postDuration": 500,
47   "postExpirationTime": "2023-12-10T05:40:24.881Z",
48   "status": "Live",
49   "likes": 0,
50   "likedBy": [],
51   "dislikes": 0,
52   "dislikedBy": [],
53   "comments": [],
54   "_v": 0
55 }
56 ]
```

TC 7.2: Olga browsing all the posts available

TC8: Nick and Olga both get Mary's post on Tech topic and adds a like. The post can be retrieved by its ID with a GET method. However, to like the post, a POST method is used.

Action point: 4

File: routes/interactions.js

API end-point: 'api/home/allposts/:postId/like'

HTTP method: POST

HTTP://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/like

POST

Headers

Key	Value	Description
auth-t...	eyJhb...	

Body

```
1 {
2   "loggedInUser": "Nick",
3   "likedPost": {
4     "_id": "6574dec76ad5de0bda5bdf7",
5     "postOwnerName": "Mary",
6     "postTopic": "Tech",
7     "postTitle": "a post about tech",
8     "messageBody": "This is Mary's first post on Tech topic",
9     "postRegistrationTime": "2023-12-09T21:40:23.080Z",
10    "postDuration": 400,
11    "postExpirationTime": "2023-12-10T04:20:23.080Z",
12    "status": "Live",
13    "likes": 1,
14    "likedBy": [
15      "6574d6746ad5de0bda5bdfc3"
16    ],
17    "dislikes": 0,
18    "dislikedBy": [],
19    "comments": [],
20    "__v": 1
21  }
22 }
```

TC 8.1: Nick likes Mary's post which is on Tech topic

HTTP://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/like

POST

Headers

Key	Value	Description
auth-token	eyJhbGciOiJIUzI1NiIsInR5cGE6YWV0Ijoi...	

Body

```
1 {
2   "loggedInUser": "Olga",
3   "likedPost": {
4     "_id": "6574dec76ad5de0bda5bdf7",
5     "postOwnerName": "Mary",
6     "postTopic": "Tech",
7     "postTitle": "a post about tech",
8     "messageBody": "This is Mary's first post on Tech topic",
9     "postRegistrationTime": "2023-12-09T21:40:23.080Z",
10    "postDuration": 400,
11    "postExpirationTime": "2023-12-10T04:20:23.080Z",
12    "status": "Live",
13    "likes": 2,
14    "likedBy": [
15      "6574d6746ad5de0bda5bdfc3",
16      "6574d6416ad5de0bda5bdfc0"
17    ],
18    "dislikes": 0,
19    "dislikedBy": [],
20    "comments": [],
21    "__v": 2
22  }
23 }
```

TC 8.2: Olga likes Mary's post on Tech Topic

TC9: Nestor first likes Nick's post that is on Tech topic. This is done by a POST method. 1 like is added to the post after the request. Here, the auth-token is turned ON just like every other case.

Then Nestor dislikes Mary's post with a /dislike call in the same way, thus a dislike count is added.

Action point: 4

File: routes/interactions.js

API end-points: 'api/home/allposts/:postId/like'

HTTP method: POST

The screenshot shows a REST client interface with the following details:

- URL:** http://34.133.238.130:8080/api/home/allposts/6574daa46ad5de0bda5bdfd5/like
- Method:** POST
- Headers (8):**

Key	Value	Description
auth-token	eyJhbG...	
- Body:**

```
{  "loggedInUser": "Nestor",  "currentPost": {    "postOwnerName": "Nick",    "postTopic": "Tech",    "postTitle": "another post about tech",    "messageBody": "This is Nick's first post on Tech topic",    "postRegistrationTime": "2023-12-09T21:22:44.164Z",    "postExpirationTime": "2023-12-10T07:22:44.164Z",    "status": "Live",    "likes": 1,    "likedBy": [      "6574d6ca6ad5de0bda5bdfc9"    ],    "dislikes": 0,    "dislikedBy": [],    "comments": []  }  }
```

TC 9.1: Nestor likes Nick's post

Action point: 4

File: routes/interactions.js

API end-points: 'api/home/allposts/:postId/dislike'

HTTP method: POST

The screenshot shows a REST client interface with the following details:

- URL:** http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/dislike
- Method:** POST
- Headers (8):**

Key	Value	Description
auth-token	eyJhbG...	
- Body:**

```
{  "loggedInUser": "Nestor",  "dislikedPost": {    "_id": "6574dec76ad5de0bda5bdf7",    "postOwnerName": "Mary",    "postTopic": "Tech",    "postTitle": "a post about tech",    "messageBody": "This is Mary's first post on Tech topic",    "postRegistrationTime": "2023-12-09T21:48:23.080Z",    "postDuration": 400,    "postExpirationTime": "2023-12-10T04:20:23.080Z",    "status": "Live",    "likes": 2,    "likedBy": [      "6574d6746ad5de0bda5bdfc3",      "6574d6416ad5de0bda5bdfc0"    ],    "dislikes": 1,    "dislikedBy": [      "6574d6ca6ad5de0bda5bdfc9"    ],    "comments": [],    "_v": 3  }  }
```

TC9.2: Nestor dislikes Mary's post

TC10: Here, Nick searches all the posts that are on Tech topic. A topic query is done here at the URL with *?postTopic=Tech* at the end of the API call. He can see all the user interactions listed here under every post.

Action point: 3

File: routes/interactions.js

API end-points: 'api/home/allposts?Query'

HTTP method: GET

The following JSON responses are shown in the screenshots:

```

1
2
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
28
29
{
  "loggedInUser": "Nick",
  "totalPosts": 3,
  "posts": [
    {
      "_id": "6574dec7aad5de8bda5bdf7",
      "postOwnerName": "Mary",
      "postTopic": "Tech",
      "postTitle": "a post about Tech",
      "messageBody": "This is Mary's first post on Tech topic",
      "postRegistrationTime": "2023-12-09T21:46:23.088Z",
      "postDuration": 480,
      "postExpirationTime": "2023-12-10T04:20:23.088Z",
      "status": "Live",
      "likes": 2,
      "likedBy": [
        "6574de74aad5de8bda5bdfc3",
        "6574d641aad5de8bda5bdfc9"
      ],
      "dislikes": 1,
      "dislikedBy": [
        "6574d6caaad5de8bda5bdfc9"
      ],
      "comments": [],
      "_v": 3
    },
    {
      "_id": "6574daa46ad5de8bda5bdf5",
      "postOwnerName": "Nick",

```

```

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
{
  "postOwnerName": "Nick",
  "postTopic": "Tech",
  "postTitle": "another post about Tech",
  "messageBody": "This is Nick's first post on tech topic",
  "postRegistrationTime": "2023-12-09T21:22:44.164Z",
  "postDuration": 600,
  "postExpirationTime": "2023-12-10T07:22:44.164Z",
  "status": "Live",
  "likes": 1,
  "likedBy": [
    "6574d6caaad5de8bda5bdfc9"
  ],
  "dislikes": 0,
  "dislikedBy": [],
  "comments": [],
  "_v": 1
},
{
  "_id": "6574da18aad5de8bda5bdf1",
  "postOwnerName": "Olga",
  "postTopic": "Tech",
  "postTitle": "Post about tech",
  "messageBody": "This is Olga's first post on Tech topic",
  "postRegistrationTime": "2023-12-09T21:20:24.881Z",
  "postDuration": 500,
  "postExpirationTime": "2023-12-10T05:40:24.881Z",
  "status": "Live",
  "likes": 0,
  "likedBy": [],

```

```

36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
{
  "status": "Live",
  "likes": 1,
  "likedBy": [
    "6574d6caaad5de8bda5bdfc9"
  ],
  "dislikes": 0,
  "dislikedBy": [],
  "comments": [],
  "_v": 1
},
{
  "_id": "6574da18aad5de8bda5bdf1",
  "postOwnerName": "Olga",
  "postTopic": "Tech",
  "postTitle": "Post about tech",
  "messageBody": "This is Olga's first post on Tech topic",
  "postRegistrationTime": "2023-12-09T21:20:24.881Z",
  "postDuration": 500,
  "postExpirationTime": "2023-12-10T05:40:24.881Z",
  "status": "Live",
  "likes": 0,
  "likedBy": [],
  "dislikes": 0,
  "dislikedBy": [],
  "comments": [],
  "_v": 0
}
]
}

```

TC 10: Nick browses all the post that belong to Tech topic

TC11: Here, Mary tries to like her post. However it is not allowed in Piazza Platform to like or dislike a user's own post. The like and dislike route is designed with action restrictions. Refer to Appendix D to Learn more about validations of the like route. The post is retrieved with GET and a like was requested with POST method. This action was denied.

Action point: 4

File: routes/interactions.js

API end-points: 'api/home/allposts/:postId/like

HTTP method: POST

The screenshot displays two REST client requests. The first is a GET request to `http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7`. The response body is a JSON object representing a post:

```
1 {
2   "loggedInUser": "Mary",
3   "currentPost": {
4     "postOwnerName": "Mary",
5     "postTopic": "Tech",
6     "postTitle": "a post about tech",
7     "messageBody": "This is Mary's first post on Tech topic",
8     "postRegistrationTime": "2023-12-09T21:40:23.080Z",
9     "postExpirationTime": "2023-12-10T04:20:23.080Z",
10    "status": "Live",
11    "likes": 2,
12    "likedBy": [
13      "6574d6746ad5de0bda5bdfc3",
14      "6574d6416ad5de0bda5bdfc0"
15    ],
16    "dislikes": 1,
17    "dislikedBy": [
18      "6574d6ca6ad5de0bda5bdfc9"
19    ],
20    "comments": []
21  }
22 }
```

The second request is a POST to `http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/like`. The response body is a JSON object with an error message:

```
1 {
2   "message": "You are not allowed to like your own post!"
3 }
```

TC 11: Like action denied in Mary's own post

TC12: Here, Nick and Olga posts in a round-robin fashion. In this case, an error occurred because a wrong attribute was passed, which clashed with the database. Thus, in TC12.1, the comment body can't be seen. However, this issue was resolved in TC12.2. Refer to Appendix E to learn about the comment workflow used in comment route.

Action point: 4

File: routes/interactions.js

API end-points: 'api/home/allposts/:postId/comment

HTTP method: POST

The screenshot shows a REST client interface with a POST request to `http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/comment`. The request body is a JSON object with the following structure:

```
{  "text": "This is Nick's 2nd comment on Mary's post on tech topic",  "user": ""}
```

The response body is a JSON object with the following structure:

```
{  "status": "Live",  "likes": 2,  "likedBy": [    "6574d6746ad5de0bda5bdfc3",    "6574d6416ad5de0bda5bdfc0"  ],  "dislikes": 1,  "dislikedBy": [    "6574d6ca6ad5de0bda5bdfc9"  ],  "comments": [    {      "commenter": "Nick",      "commentDate": "2023-12-09T22:10:51.323Z",      "_id": "6574e5eb42615aad838656b1"    },    {      "commenter": "Olga",      "commentDate": "2023-12-09T22:11:00.717Z",      "_id": "6574e5f442615aad838656b6"    },    {      "commenter": "Nick",      "commentDate": "2023-12-09T22:11:14.230Z",      "_id": "6574e60242615aad838656bc"    }  ],  "_v": 6}
```

TC 12.1: Comments with Wrong data; in round-robin fashion

http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/comment

POST http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/comment

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

raw JSON Beautify

```

1  {
2    "commenter": "",
3    "comment": "This is Olga's 3rd comment on Mary's post
4    on tech topic"
  }

```

Body Cookies Headers (7) Test Results 200 OK 350 ms

Pretty Raw Preview Visualize JSON

```

21  {
22    "commenter": "Nick",
23    "commentDate": "2023-12-09T22:10:51.323Z",
24    "_id": "6574e5eb42615aad838656b1"
25  },
26  {
27    "commenter": "Olga",
28    "commentDate": "2023-12-09T22:11:00.717Z",
29    "_id": "6574e5f442615aad838656b6"
30  },
31  {
32    "commenter": "Nick",
33    "commentDate": "2023-12-09T22:11:14.230Z",
34    "_id": "6574e60242615aad838656bc"
35  },
36  {
37    "commenter": "Olga",
38    "commentDate": "2023-12-09T22:11:23.742Z",
39    "_id": "6574e60b42615aad838656c3"
40  },
41  {
42    "commenter": "Olga",
43    "comment": "This is Olga's 3rd comment on Mary's post on tech topic",
44    "commentDate": "2023-12-09T22:20:18.555Z",
45    "_id": "6574e82242615aad838656d3"
46  }
47  ],
48  "__v": 8
49

```

http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/comment

POST http://34.133.238.130:8080/api/home/allposts/6574dec76ad5de0bda5bdf7/comment

Body Cookies Headers (7) Test Results Status: 200 OK

raw JSON Beautify

```

1  {
2    "commenter": "",
3    "comment": "This is Nick's 3rd comment on
4    Mary's post on tech topic"
  }

```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

27  {
28    "commenter": "Olga",
29    "commentDate": "2023-12-09T22:11:00.717Z",
30    "_id": "6574e5f442615aad838656b6"
31  },
32  {
33    "commenter": "Nick",
34    "commentDate": "2023-12-09T22:11:14.230Z",
35    "_id": "6574e60242615aad838656bc"
36  },
37  {
38    "commenter": "Olga",
39    "commentDate": "2023-12-09T22:11:23.742Z",
40    "_id": "6574e60b42615aad838656c3"
41  },
42  {
43    "commenter": "Olga",
44    "comment": "This is Olga's 3rd comment on Mary's post on tech topic",
45    "commentDate": "2023-12-09T22:20:18.555Z",
46    "_id": "6574e82242615aad838656d3"
47  },
48  {
49    "commenter": "Nick",
50    "comment": "This is Nick's 3rd comment on Mary's post on tech topic",
51    "commentDate": "2023-12-09T22:21:13.654Z",
52    "_id": "6574e85942615aad838656e4"
53  }
54  ],
55  "__v": 9

```

TC 12.2: After correcting the data Keys

TC13: Nick again browses all the posts on Tech topic. Now he can see the interactions added. He queries with *?postTopic=Tech* and send a GET request.

Action point: 4

File: routes/browseRoute.js

API end-points: 'api/home/allposts?Query'

HTTP method: GET

HTTP GET http://34.133.238.130:8080/api/home/allposts?postTopic=Tech

Status: 200 OK

Headers (7):

Key	Value	Description
auth-t...	eyJhb...	
Key	Value	Description

Body (JSON):

```
{
  "loggedInUser": "Nick",
  "totalPosts": 3,
  "posts": [
    {
      "_id": "6574dec76ad5de0bda5bdf7",
      "postOwnerName": "Mary",
      "postTopic": "Tech",
      "postTitle": "a post about tech",
      "messageBody": "This is Mary's first post on Tech topic",
      "postRegistrationTime": "2023-12-09T21:40:23.080Z",
      "postDuration": 400,
      "postExpirationTime": "2023-12-10T04:20:23.080Z",
      "status": "Live",
      "likes": 2,
      "likedBy": [
        "6574d6746ad5de0bda5bdfc3",
        "6574d6416ad5de0bda5bdfc8"
      ],
      "dislikes": 1,
      "dislikedBy": [
        "6574d6ca6ad5de0bda5bdfc9"
      ],
      "comments": [
        {
          "commenter": "Nick",
          "commentDate": "2023-12-09T22:10:51.323Z",
          "_id": "6574e5eb42615aad838656b1"
        }
      ]
    }
  ]
}
```

Headers (7):

Key	Value	Description
auth-t...	eyJhb...	
Key	Value	Description

Body (JSON):

```
{
  "commenter": "Olga",
  "commentDate": "2023-12-09T22:11:00.717Z",
  "_id": "6574e5f442615aad838656b6"
},
{
  "commenter": "Nick",
  "commentDate": "2023-12-09T22:11:14.230Z",
  "_id": "6574e60242615aad838656bc"
},
{
  "commenter": "Olga",
  "commentDate": "2023-12-09T22:11:23.742Z",
  "_id": "6574e60b42615aad838656c3"
},
{
  "commenter": "Olga",
  "comment": "This is Olga's 3rd comment on Mary's post on tech topic",
  "commentDate": "2023-12-09T22:18:18.555Z",
  "_id": "6574e82242615aad838656d3"
},
{
  "commenter": "Nick",
  "comment": "This is Nick's 3rd comment on Mary's post on tech topic",
  "commentDate": "2023-12-09T22:21:13.654Z",
  "_id": "6574e85942615aad838656e4"
}
]
```

TC 13.1: Browse posts p1

HTTP <http://34.133.238.130:8080/api/home/allposts?postTopic=Tech>

GET <http://34.133.238.130:8080/api/home/allposts?postTopic=Tech>

Headers 8 hidden

	Key	Value	Description	Bulk Edit	Presets
<input checked="" type="checkbox"/>	auth-t...	eyJhb...			
	Key	Value	Description		

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
61 {
62   "_id": "6574daa46ad5de0bda5bdfd5",
63   "postOwnerName": "Nick",
64   "postTopic": "Tech",
65   "postTitle": "another post about tech",
66   "messageBody": "This is Nick's first post on Tech topic",
67   "postRegistrationTime": "2023-12-09T21:22:44.164Z",
68   "postDuration": 600,
69   "postExpirationTime": "2023-12-10T07:22:44.164Z",
70   "status": "Live",
71   "likes": 1,
72   "likedBy": [
73     "6574d6ca6ad5de0bda5bdfc9"
74   ],
75   "dislikes": 0,
76   "dislikedBy": [],
77   "comments": [],
78   "__v": 1
79 },
80 {
81   "_id": "6574da186ad5de0bda5bdfd1",
82   "postOwnerName": "Olga",
83   "postTopic": "Tech",
84   "postTitle": "Post about tech",
85   "messageBody": "This is Olga's first post on Tech topic",
86   "postRegistrationTime": "2023-12-09T21:20:24.881Z",
87   "postDuration": 500,
88   "postExpirationTime": "2023-12-10T05:40:24.881Z",
89   "status": "Live",
90   "likes": 0,
91   "likedBy": [],
92   "dislikes": 0,
93   "dislikedBy": [],
94   "comments": [],
95   "__v": 0
96 }
97 ]
```

TC13.2: Browse posts p2

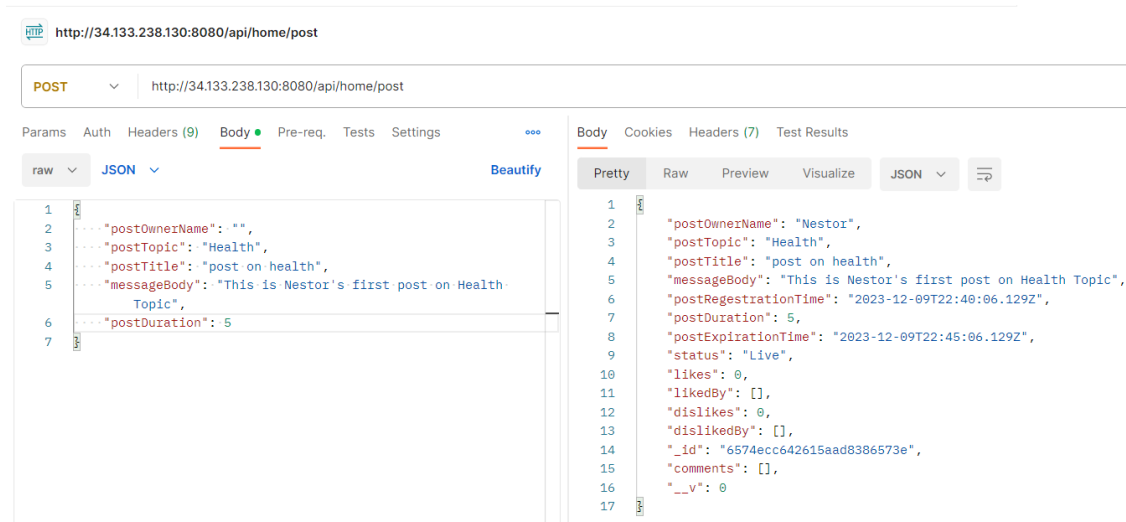
TC14: Now, Nestor posts a message on Health topic. The time is set to 5 min now. Time time is kept short this time so the upcoming tasks can be executed. Ofcourse, the post is done with the auth-token.

Action point: 2

File: routes/postRoute.js

API end-points: 'api/home/post'

HTTP method: POST



TC 14: Nestor posts a message

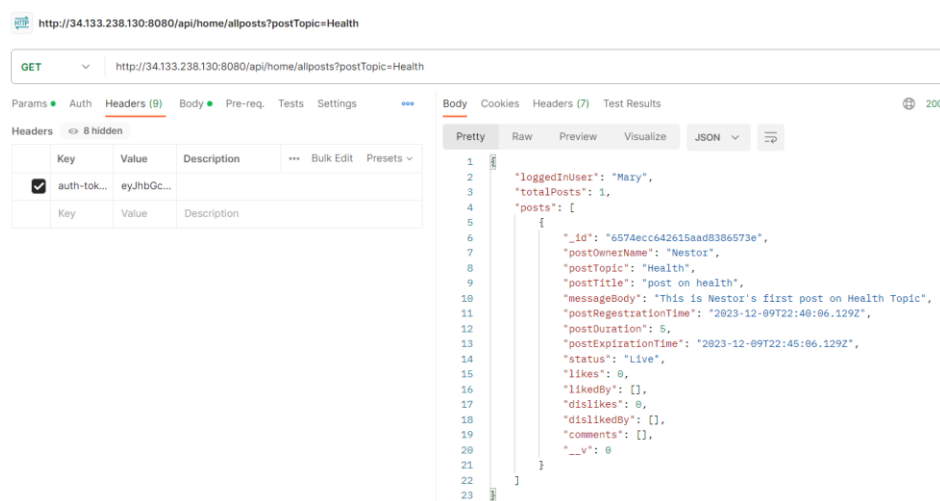
TC15: Mary queries to browse all the post with Health topic. She can only see 1 post which is done by Nestor previously.

Action point: 3

File: routes/browseRoute.js

API end-points: 'api/home/allposts?Query'

HTTP method: GET



TC 15: Mary browses all posts on Health topic

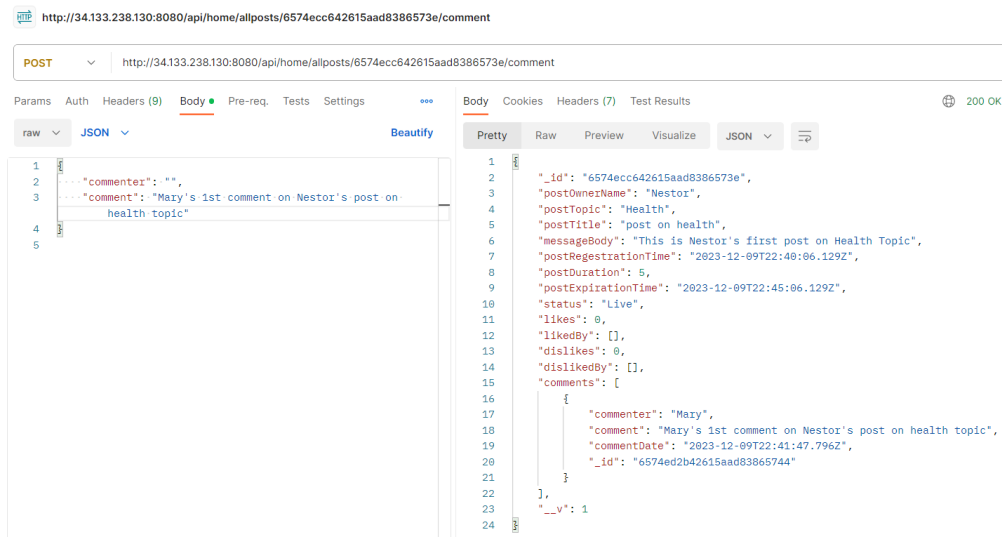
TC16: Now, Mary posts a comment on Nestor's post on health topic. The post is still active because the time duration is not over yet. Therefore, a comment POST request is accepted. This time, the comment can be seen correctly, unlike TC12.1.

Action point: 4

File: routes/interactions.js

API end-points: 'api/home/allposts/:postId/comment'

HTTP method: POST



TC 16: Mary posts comment on Nestor's post

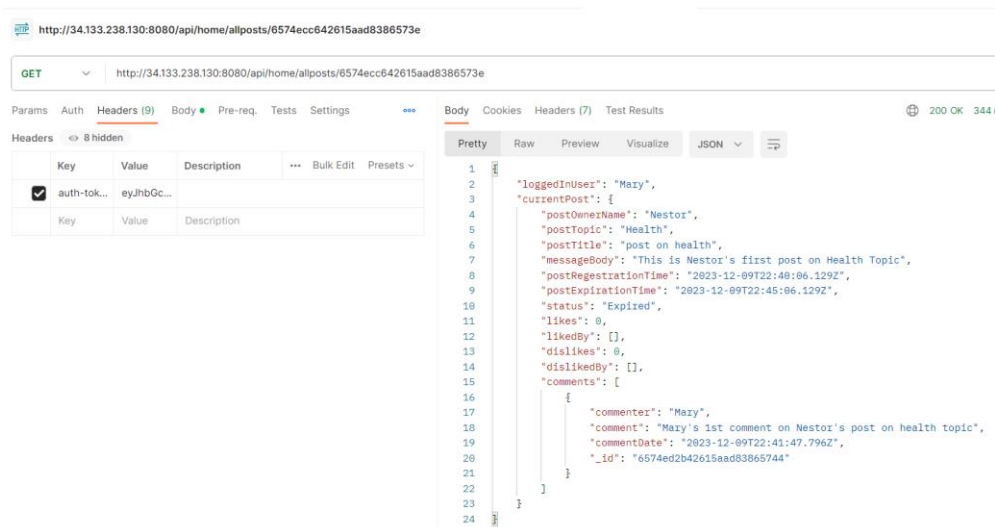
TC17: Mary attempts to dislike Nestor's post, but the action will be denied because the post is expired now. The status has now changed from Live to Expired too.

Action point: 4

File: routes/interactions.js

API end-points: 'api/home/allposts/:postId/dislike'

HTTP method: POST



TC 17.1: Mary's dislike attempt

http://34.133.238.130:8080/api/home/allposts/6574ecc642615aad8386573e/dislike

POST http://34.133.238.130:8080/api/home/allposts/6574ecc642615aad8386573e/dislike

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

Headers 8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	auth-tok...	eyJhbGc...				
	Key	Value	Description			

Body Cookies Headers (7) Test Results 400 B

Pretty Raw Preview Visualize JSON

```

1
2
3
"message": "Action can not be executed: post is already Expired."

```

TC 17.2: Dislike attempt denied

TC18: Now Nestor queries for all posts on Health topic. He only sees 1 post, which belongs to him. Mary's comment is also listed under.

Action point: 3

File: routes/browseRoute.js

API end-points: 'api/home/allposts?Query'

HTTP method: GET

http://34.133.238.130:8080/api/home/allposts?postTopic=Health

GET http://34.133.238.130:8080/api/home/allposts?postTopic=Health

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

Headers 8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	auth-token	eyJhbGc...				
	Key	Value	Description			

Body Cookies Headers (7) Test Results 200 OK 284 ms 82

Pretty Raw Preview Visualize JSON

```

1
2
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
28
29
{
  "loggedInUser": "Nestor",
  "totalPosts": 1,
  "posts": [
    {
      "_id": "6574ecc642615aad8386573e",
      "postOwnerName": "Nestor",
      "postTopic": "Health",
      "postTitle": "post on health",
      "messageBody": "This is Nestor's first post on Health Topic",
      "postRegistrationTime": "2023-12-09T22:40:06.129Z",
      "postDuration": 5,
      "postExpirationTime": "2023-12-09T22:45:06.129Z",
      "status": "Expired",
      "likes": 0,
      "likedBy": [],
      "dislikes": 0,
      "dislikedBy": [],
      "comments": [
        {
          "commenter": "Mary",
          "comment": "Mary's 1st comment on Nestor's post on health topic",
          "commentDate": "2023-12-09T22:41:47.796Z",
          "_id": "6574ed2b42615aad83865744"
        }
      ]
    }
  ],
  "_v": 1
}

```

TC18: Nestor browses posts on Health topic

TC 20: Nestor queries for Tech posts with the highest interactions and still Active

8. Deploying Piazza to Kubernetes

Kubernetes, an open-source platform which is used to automate deployment, scale, and administer software applications. It is used to easily replicate large numbers of containers and control them all as a unified platform. It is helpful because by working as a whole, each container will not require individual management.

In this coursework, a cluster is created using Kubernetes Engine. But before that, Piazza needs to be dockerised with a docker image. This image is different from the one created in Section 7. This image is then pushed into DockerHub. Kubernetes will access this image from DockerHub to containerize. Refer to Appendix F to see details of the image in DockerHub.

```
Successfully built 8ea412430107
Successfully tagged oanower/app-piazza:1
docker-u@coursework-vm:~/coursework-SaaS$ docker login -u oanower
Password:
WARNING! Your password will be stored unencrypted in /home/docker-u/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
docker-u@coursework-vm:~/coursework-SaaS$ docker push oanower/app-piazza:1
The push refers to repository [docker.io/oanower/app-piazza]
4b2a36e8b872: Pushed
f2a742512446: Pushed
5af4f8f59b76: Mounted from library/alpine
1: digest: sha256:bf61d41945b30f6d156da313c442764ae8f0ac7b29c59ad4e0b045ba81a416bc size: 951
docker-u@coursework-vm:~/coursework-SaaS$ docker search oanower
NAME                DESCRIPTION          STARS     OFFICIAL   AUTOMATED
oanower/app-piazza  Container for Piazza App: Cloud Computing Co...  0
```

Image 6: Image pushed into DockerHub

Then, in GCP, a Kubernetes cluster is created and it's credential is retrieved. This credential allows you to connect to the cluster from Google Cloud Shell. It has 3 nodes running by default. A '*kubectl*' comment is used to perform any operation inside a Kubernetes cluster.

Now, in a .yaml configuration file, some data are added that allows to replicate Piazza app. For this assignment, 5 pods are created to host the Piazza application container using the docker image. Image 6 shows all the running pods of the application with their respective IP addresses.

piazzaapp-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: piazzaapp-deployment
  labels:
    app: piazzaapp
spec:
  replicas: 5
  selector:
    matchLabels:
      app: piazzaapp
  template:
    metadata:
      labels:
        app: piazzaapp
    spec:
      containers:
        - name: piazzaapp
          image: oanower/app-piazza:1
          imagePullPolicy: Always
          ports:
            - containerPort: 3000
```



```
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ pico piazzaapp-deployment.yaml
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl apply -f piazzaapp-deployment.yaml
deployment.apps/piazzaapp-deployment created
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
piazzaapp-deployment-798c487785-b9dk5 1/1     Running   0           18s
piazzaapp-deployment-798c487785-btnehj 1/1     Running   0           18s
piazzaapp-deployment-798c487785-fdqtq 1/1     Running   0           18s
piazzaapp-deployment-798c487785-mkzrj 1/1     Running   0           18s
piazzaapp-deployment-798c487785-w8tpz 1/1     Running   0           18s
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE                                NOMINATED NODE
piazzaapp-deployment-798c487785-b9dk5 1/1     Running   0           31s   10.16.1.14    gke-kubernetes-cluster-p-default-pool-edbeec0-vrsr <none>
piazzaapp-deployment-798c487785-btnehj 1/1     Running   0           31s   10.16.2.8     gke-kubernetes-cluster-p-default-pool-edbeec0-q5vn <none>
piazzaapp-deployment-798c487785-fdqtq 1/1     Running   0           31s   10.16.2.7     gke-kubernetes-cluster-p-default-pool-edbeec0-q5vn <none>
piazzaapp-deployment-798c487785-mkzrj 1/1     Running   0           31s   10.16.0.7     gke-kubernetes-cluster-p-default-pool-edbeec0-30r6 <none>
piazzaapp-deployment-798c487785-w8tpz 1/1     Running   0           31s   10.16.0.6     gke-kubernetes-cluster-p-default-pool-edbeec0-30r6 <none>
```

Image 7: 5 running Pods of Piazza

The next step involves connecting the cluster to the internet using a Load Balancer. A load balancer helps to balance traffic and increase resilience (Gestat, 23) . To add a load balancer, a service is created using another .yaml file. This service maps the port 8080 of the Kubernetes cluster with port 3000 of the Piazza server that is running inside Docker. Load Balancer updates automatically if there is a configuration change in a Kubernetes cluster.

```
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ pico piazzaapp-service.yaml
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl apply -f piazzaapp-service.yaml
service/piazzaapp-service created
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.20.0.1    <none>         443/TCP          11m
piazzaapp-service    LoadBalancer 10.20.14.3    <pending>      8080:30401/TCP   11s
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.20.0.1    <none>         443/TCP          11m
piazzaapp-service    LoadBalancer 10.20.14.3    <pending>      8080:30401/TCP   20s
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.20.0.1    <none>         443/TCP          11m
piazzaapp-service    LoadBalancer 10.20.14.3    <pending>      8080:30401/TCP   33s
oanowe01@cloudshell:~ (pelagic-firefly-401119)$ kubectl get services
NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes           ClusterIP   10.20.0.1    <none>         443/TCP          11m
piazzaapp-service    LoadBalancer 10.20.14.3    35.192.72.2   8080:30401/TCP   51s
```

piazzaapp-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: piazzaapp-service
  labels:
    app: piazzaapp-service
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 8080
      protocol: TCP
      targetPort: 3000
  selector:
    app: piazzaapp
  sessionAffinity: None
```

Image 8: Load balancer running

The External IP retrieved from the load balancing service is used to run the first API point of the Piazza app (inside the app.js file). Now that the Piazza Software is deployed in Kubernetes successfully, it responds to the HTTP call!

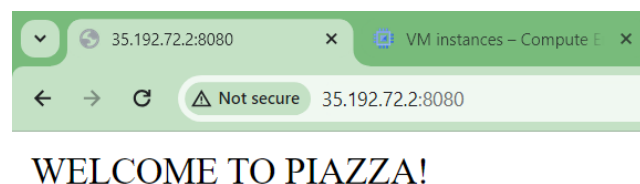


Image 9: Piazza Software deployed in Kubernetes.

The process in this section is adapted from Cloud Computing Lab 6 (S, 2023).

9. Area of Improvements & Future Work

Name in the URL during API calls: While running the test cases, it was seen that while testing an interaction by an authorised user, the user's name is not mentioned in the URL. This could possibly be fixed by redirecting to a desired URL containing user's first name and last name after the user passes authentication.

OAuth2 using Service providers: It provides a secure protocol for authorization between APIs and third-party service so the provider doesn't require access to user credentials (Bello, 2023). While creating this project, an attempt was made to access tokens via Google sign-in verification process. Using Google client credentials. Some codes were also attempted using online sources. However, this approach was not implemented in the main project at the end.

Use CI/CD Pipelines: By making a CI/CD pipeline in Terraform, it will allow to manage the Piazza software infrastructure and services automatically. The infrastructure can be created by spinning up different servers and deploying Piazza's docker containers with a MongoDB database container. The infrastructure can be built in the Google Cloud Platform. If the software is updated or new functionalities are added, the infrastructure can be managed using Terraform. By adding some automation tools, the changes can be handles easily by Terraform. (TechWorldwithNana, 2020)

Conclusion

In conclusion, the development of NodeJS-based software Piazza as a service with NPM uses the flexibility of JavaScript to build a well-performing scalable application. The application efficiently uses an OAuth2 workflow to verify the authenticity of a user in the system and ensures secure access to the Piazza resources. The software successfully integrates its resources with the MongoDB database and utilizes data accurately for different HTTP methods and API calls. With different test cases, the working function of the RESTful API are visualised in the report. While testing the software, it was discovered that a query function can be implemented to redirect to a URL that displays the current user's name in the URL, so it is clear to viewers who is currently signed-in.

By using docker, the software is easily containerized to make it accessible to the internet using a virtual machine's IP address. This same IP is used to test the application. Lastly, with the help of Kubernetes, the app was containerised into five replicas to allow scalability and efficiency.

References

- Anower, O. (2023). Workflow of Sign-in process. *N/A*. BBK, London.
- Banimahd, M. (2023, 01 13). *Joi: The Ultimate Solution for Data Validation in Node.js*. Retrieved from Medium: <https://masoudx.medium.com/joi-the-ultimate-solution-for-data-validation-in-node-js-3b72f9695931>
- Barros, S. (2019, 06 21). *Authentication: how to create a NodeJS application using JWT*. Retrieved from Medium: <https://medium.com/swlh/authentication-how-to-create-a-nodejs-application-using-jwt-cee8bc5a89fe>
- Bello, G. (2023, 13 12). *How to access Google APIs using OAuth 2.0 in Postman*. Retrieved from Postman Blog: <https://blog.postman.com/how-to-access-google-apis-using-oauth-in-postman/>
- Charboneau, T. (2022, 09 08). *How to Use the Alpine Docker Official Image*. Retrieved from docker: <https://www.docker.com/blog/how-to-use-the-alpine-docker-official-image/>
- Dineshchandgr. (2022, 11 24). *Do you know about OAuth2 Protocol and its different flows?* Retrieved from Medium: <https://medium.com/javarevisited/do-you-know-about-oauth2-protocol-and-its-different-flows-9d849cec45b0>
- Gestat, A. (23, 03 10). *Bringing traffic to your pods with Load Balancer*. Retrieved from Scaleway Blog: <https://www.scaleway.com/en/blog/kubernetes-load-balancer/>
- Goswami, P., Gupta, S., Li, Z., Meng, N., & Yao, D. (2020). Investigating The Reproducibility of NPM Packages. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (p. 677). Adelaide, SA, Australia. doi:10.1109/ICSME46990.2020.00071
- Kris. (2021, 02 22). *Set Up and Run a Simple Node Server Project*. Retrieved 12 03, 2023, from Medium: <https://levelup.gitconnected.com/set-up-and-run-a-simple-node-server-project-38b403a3dc09>
- Mavrogiannopoulos, N. (2021, 11 23). *Cloud-optimized Linux kernels – what makes Ubuntu the top OS across the clouds*. Retrieved 11 20, 2023, from Ubuntu: <https://ubuntu.com/blog/cloud-optimized-linux-kernels>
- Mohsin, Z. (2023). *How to hash passwords using bcrypt in Node.js*. Retrieved from educative: <https://www.educative.io/answers/how-to-hash-passwords-using-bcrypt-in-nodejs>
- S, S. (2023, 11 08). *Lab6: Deploying on Kubernetes*. Retrieved from GitHub: <https://github.com/warestack/cc/blob/master/Class-6/README.md>
- Sotiriadis, S. (2023, 11 02). *Lab 5*. Retrieved from Github: <https://github.com/warestack/cc/tree/master/Class-5>
- Stelios, S. (2022, 02 05). *Lab 4*. Retrieved from GitHub: <https://github.com/warestack/cc/commits/master/Class-4/Lab4.1-Building-the-MiniFilm-application.md>
- TechWorldwithNana. (2020). *Terraform explained in 15 mins | Terraform Tutorial for Beginners*. Retrieved from Youtube: https://www.youtube.com/watch?v=I5k1ai_GBDE

Appendices

Appendix A

Boot disk

Select an image or snapshot to create a boot disk, or attach an existing disk. Can't find what you're looking for? Explore hundreds of VM solutions in [Marketplace](#)

PUBLIC IMAGES

CUSTOM IMAGES

SNAPSHOTS

ARCHIVE SNAP

Operating system

Ubuntu

Version *

Ubuntu 20.04 LTS

x86/64, amd64 focal image built on 2023-11-01

Boot disk type *

Balanced persistent disk







COMPARE DISK TYPES

Size (GB) *

25

Provision between 10 and 65536 GB

Appendix B

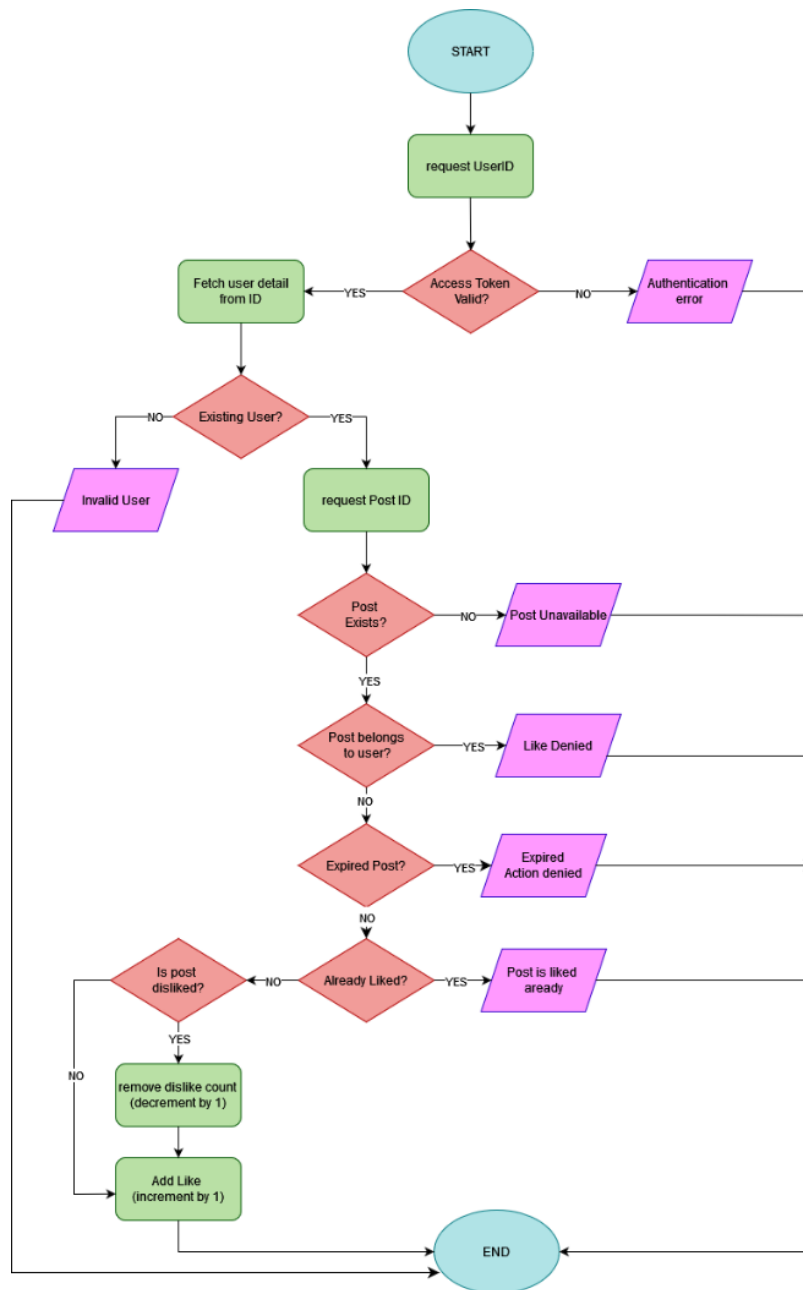
Basic information	
Name	coursework-vm
Instance ID	1366829003863050318
Description	None
Type	Instance
Status	 Stopped
Creation time	Nov 18, 2023, 5:38:42 pm UTCZ
Zone	us-central1-a
Instance template	None
In use by	None
Reservations	Automatically choose
Labels	None
Tags 	— 
Deletion protection	Disabled
Confidential VM service 	Disabled
Preserved state size	0 GB
Machine configuration	
Machine type	e2-medium
CPU platform	Unknown CPU Platform
Minimum CPU platform	None
Architecture	x86/64
vCPUs to core ratio 	—
Custom visible cores 	—
Display device	Disabled Enable to use screen capturing and recording tools
GPUs	None

Appendix C

```
{ } package.json > ...  
1  {  
2    "name": "coursework-app",  
3    "version": "1.0.0",  
4    "main": "index.js",  
    ▶ Debug  
5    "scripts": {  
6      "start": "nodemon app.js"  
7    },  
8    "author": "",  
9    "license": "ISC",  
10   "dependencies": {  
11     "bcryptjs": "^2.4.3",  
12     "body-parser": "^1.20.2",  
13     "dotenv": "^16.3.1",  
14     "express": "^4.18.2",  
15     "express-session": "^1.17.3",  
16     "joi": "^17.11.0",  
17     "jsonwebtoken": "^9.0.2",  
18     "mongodb": "^6.3.0",  
19     "mongoose": "^8.0.1",  
20     "nodemon": "^3.0.1",  
21     "passport": "^0.7.0",  
22     "passport-google-oauth20": "^2.0.0",  
23     "request-promise": "^4.2.6"  
24   },  
25   "description": "",  
26   "devDependencies": {  
27     "@types/express": "^4.17.21"  
28   }  
29 }  
30
```

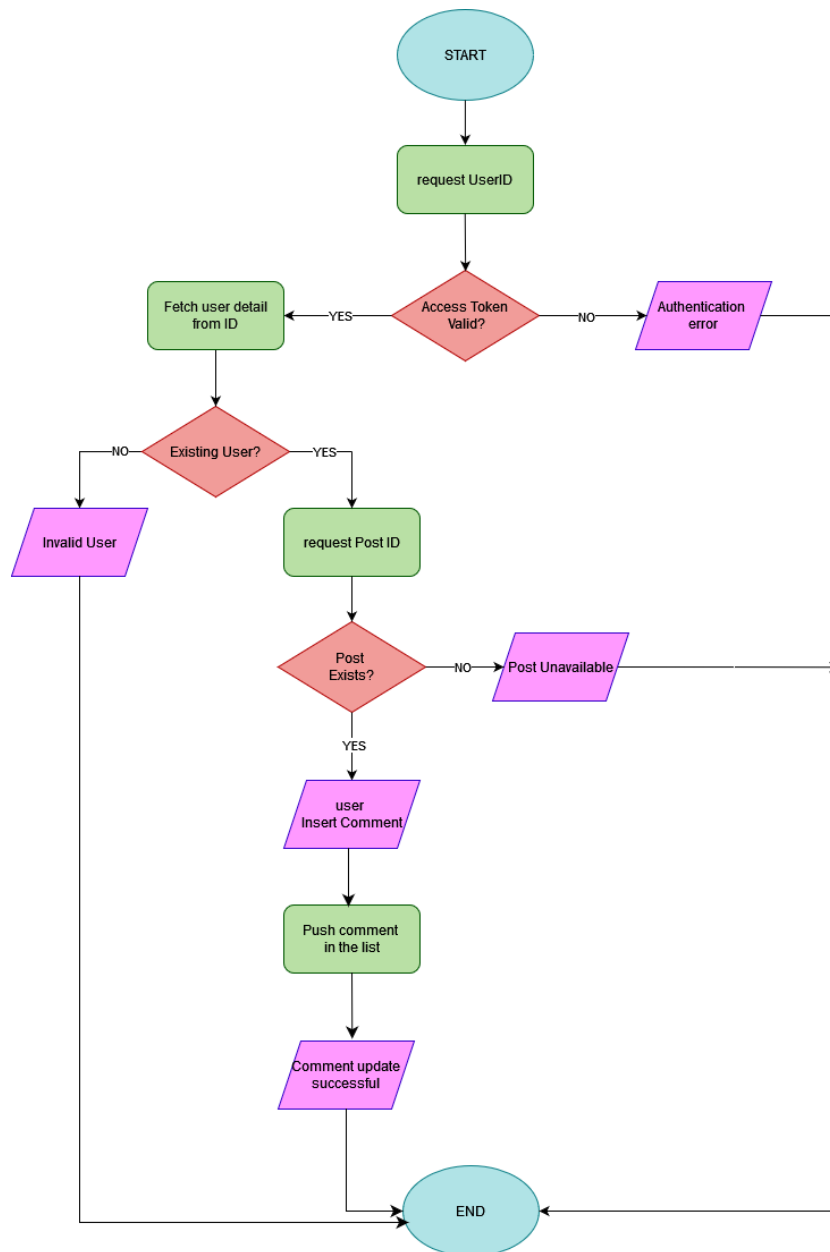
Appendix D

Workflow of the Like route. This image is generated by the student via draw.io.



Appendix E

Workflow of the comment route. This image is generated by the student via draw.io.



Appendix F



oanower/app-piazza:1

DIGEST: sha256:bf61d41945b30f6d156da313c442764ae8f0ac7b29c59ad4e0b045ba81a416bc

OS/ARCH	COMPRESSED SIZE ⓘ	LAST PUSHED	TYPE
linux/amd64	35.15 MB	6 minutes ago by oanower	Image

Image Layers Vulnerabilities

IMAGE LAYERS ⓘ

1	ADD file ... in /	3.25 MB
2	CMD ["/bin/sh"]	0 B
3	/bin/sh -c apk add --update	23.46 MB
4	COPY dir:4874ee0e70aba7060c45caf080d3ad94686cb0...	8.44 MB
5	WORKDIR /src	0 B
6	EXPOSE 3000	0 B
7	ENTRYPOINT ["node" "./app.js"]	0 B

Command

ADD file:1f4eb46669b5b