Cloud computing coursework Report - Serhiy Pikho

Phase A - Install and deploy software in virtualised environments

Github link: https://github.com/Serhiy1/Cloud-computing-coursework---piazza

1. The development environment uses a VS code development container.
    - The development container is based of the default nodeJs Docker image
    - The image used for the dev container can be re-used for deploying the service to the cloud
    - The dev container is set to install a set of default VS code plugins:
        1. `thunder-client` a postMan Clone that is present inside VS code
        2. `Spell checker`
        3. `Mongo DB` plugin for connecting and debugging the database
        4. `Jest` for running tests from the IDE
2. Setting up typescript compilation and default packages
    - This guide was used for getting node, express and typescript to work with each other https://blog.logrocket.com/how-to-set-up-node-typescript-express/
    - Typescript was selected due to the advantages that type checking and autocomplete provides with reducing bugs and speeding up development with autocomplete
3. setup Linting and formatting configurations
    - Eslint and prettier were selected as they are the industry defaults
4. Selecting packages and plugins for the project
    1. morgan - Logging middle ware for express JS
    2. Mongoose - For interacting with the mongo data base
    3. express-validator - validating user input
    4. bycrypt - hashing passwords so they are not stored as plain text
    5. jsonwebtoken - For creating the JSON web tokens for oauth
    6. Faker - For creating dummy data in the tests
5. Project structure - The folder structure of the project follows industry standards, all code is located in the `src` folder.
    - `.src/app/api/routes` subfolder contains specific handlers for the different API endpoints
    - `.src/app/models` subfolder contains the types for database / API interaction
    - `.src/app/utils` contains utility code and classes
    - `.src/testing` Contains all the test code

    The `./dist` folder contains the compiled javascript that is run by node.

Running the app locally.

1. Open the project in VS code
2. Copy `.env-empty-copy` and rename it to `.env`. Add all the secure details for the deployment
3. Install the remote development extension for VS code
4. Re-open the project inside the development container - This should start the docker image and attach VS code to it.
5. run `npm run dev` to start the dev server

Phase B - user Authentication

file structure

- The file that implements the API routes in `src/app/api/routes/userRouter.ts`
- The file that implements the mongo models is in `src/app/models/user.ts`
- `src/app/utils/auth.ts` contains helper code

API Paths

```
POST ${host}/user/signup -> user posts email, username and password to register
themselves on the app
POST ${host}/user/login -> user posts email and password to authenticate
themselves and receive a JWT token

GET ${host}/user -> User can see their own public details
GET ${host}/user/${userID} -> User can see the public details of other users
```

- A 200 response containing the Public view of the User is returned when signing up
- A 409 response is returned if an email or username is already in use
- A 400 response is returned if any of the fields are missing
- A 400 response is returned if the password complexity is not met: 8 char min, 1 capital, 1 number, 1 symbol

When Signing up the following payload needs to be sent

`POST user/signup`

```
{
  "email": "${email}",
  "password" : "${password}",
  "userName" : "${username}"
}
```

When logging in the following payload needs to be sent

`POST user/login`

```
{
  "email": "${email}",
  "password" : "${password}"
}
```

A 400 response is returns if any of the fields are missing A 403 response is returned when the username or password does not match A 200 response is returned when successfully logging in `{message: "auth succeeded", "token" : "${jwt token}"}`

Viewing a user

A 400 response is returned when the user ID does not exist

When A user is successfully found the returned format is `GET user/${userId}`

```
{
  "user": {
    "userName": "${username}",
    "email": "${email}",
    "likedComments": [
      "${mongo ID}"
    ],
    "diLikedComments": [
      "${mongo ID}"
    ]
  },
  "posts": [
    // ${Post JSON}
  ],
  "comments": [
    // ${Post JSON}
  ]
}
```

Database design for the user

```
const userSchema = new Schema<IUser>({
  _id: { type: Schema.Types.ObjectId, required: true },
  userName: { type: String, required: true },
  email: { type: String, required: true },
  passwordHash: { type: String, required: true },
  likedComments: { type: [Schema.Types.ObjectId], required: false, default: [] },
  diLikedComments: { type: [Schema.Types.ObjectId], required: false, default: []
},
});
```

Some notable design decisions with the documents

- The password is stored as a slated hash as part of security considerations
- Liked/disliked comments are stored on the User Document. This was a performance consideration.
    - Posts will be interacted with by many users, Incrementing / decrementing a counter is an quick operation
    - User documents will not be under such a high interaction rate, so more expensive find and remove operations are done on them instead
- Posts are not stored on the User document, Posts made by a user a found by listing all Post documents with a matching user ID

Phase C - Developing the API for creating and viewing posts

file structure

- The file that implements the API routes in `src/app/api/routes/postRouter.ts`
- The file that implements the mongo models is in `src/app/models/post.ts`

API Paths

```
GET ${host}/posts/topics -> return a list of valid topics
GET ${host}/posts/topics/${topicID} -> return a list of all the live posts that
are not comments matching that topic
GET ${host}/posts/topics/${topicID}/expired -> return a list of all the expired
posts that are not comments matching that topic


GET ${host}/posts -> return a list of all the live posts that are not comments
without any topic filter
GET ${host}/posts -> return a list of all the expired posts that are not comments
without any topic filter
GET ${host}/posts/${postID} -> view a single post and a list of all the comments
on it

POST ${host}/posts -> Create a new post
POST ${host}/posts/${postID} -> Comment on a post

POST ${host}/posts/${postID}/like -> Like a post
POST ${host}/posts/${postID}/dislike -> dislike a post
```

Global Rules

- User cannot like, dislike or comment on post that is marked as expired
- All Posts go expired depending on the `expiredTimeHours` environment variable
    - When attempting to interact with an expired comment as 400 response is returned
- All API endpoints on /post require a jwt token from the /login endpoint
    - When attempting to interact without a token a 403 response is returned
- All posts require atlas one topic entry in `"politics", "Health", "sport", "Tech"`

Liking, disliking and commenting on Posts

- when a person likes a post that they have already disliked it un-does the dislike and vice versa
- When a person likes/dislikes a post a second time it un-does the first action
- The parentId field is only present on comments
- A person cannot like or dislike their own post
- When Listing posts on a topic or globally, the comments are compressed into a count. To view individual comments you need to specifically `GET` a post

selecting the order Posts are listed

users can append `?orderBy=` query to to the `/posts` and `/posts/topics/${topicID}` paths. by default the ordering is by created date

```
?orderBy=Liked    # order by the most liked posts
?orderBy=Disiked    # order by the most Disliked posts
?orderBy=Activity    # order by the most liked and disliked posts
```

post JSON example Response

```
{
  "title" : "${title}"
  "userName": "${username}",
  "content": "${content}",
  "likes": 0,
  "dislikes": 0,
  "created": "2023-12-05T15:06:58.468Z",
  "topics": ["Tech"],
  "link": "${host}/posts/${mongo ID}",
  "user_link": "${host}/users/${mongo ID}",
  "post_type": "Post",  // or Comment
  "comments": 1,
  "status": "Active", // or "inactive",
  "Expires_in" : "1 hour"
}
```

Creating a post

`POST /posts`

```
{
  "title" : "${title}",
  "content" : "${content}",
  "topics" : ["${valid topic}"]
}
```

Commenting on a post

`POST /posts/${postId}`

```
{
  "content" : "${content}",
}
```

Database Design for the post document

This is the final schema for Posts and Comments

```
const PostSchema = new Schema<IPOST>({
  _id: { type: Schema.Types.ObjectId, required: true },
  title: { type: String, required: false, default: null },
  ownerId: { type: Schema.Types.ObjectId, required: true, ref: "User" },
  userName: { type: String, required: true },
  parentId: { type: Schema.Types.ObjectId, required: false, default: null, ref:
"Post" },
  childIds: { type: [Schema.Types.ObjectId], required: false, default: [], ref:
"Post" },
  content: { type: String, required: true },
  likes: { type: Number, required: false, default: 0 },
  dislikes: { type: Number, required: false, default: 0 },
  activity: { type: Number, required: false, default: 0 },
  created: { type: Date, required: true },
  topics: { type: [String], enum: Object.values(ValidTopics), required: true },
});
```

- The Post Document represents both root posts and comments
- With performance in mind there the number of references to other documents is kept to a minimum
    - Since there are no deletions only append operations need to happen on the document.
    - Who liked or disliked a post is not tracked on the post itself

- Only the Created Date is stored on the document
    - If its active or not is calculated on the fly by the application, this allows for configuring the expiry time of a post without the the need for migrations and the ability for dynamic information such a countdown to the expiry
- Only immediate children are considered for the comment count, The design does allow for branching threads (tree like structures) counting comments across the tree would be extremely expensive as multiple database queries would be made.

Phase D - Testing the application

- Adhoc testing during the development is done with the postman clone thunder client.
- The `jest`, `supertest`, `MongoMemoryServer` packages are used for the structured testing. These where selected as recommendations from the following guide.
- To run these tests you need to type into the terminal `npm run test`

Reasons for the package selection

- `Jest` it integrates with the jest vs code plugin and allows breakpoint debugging.
- `supertest` is a test framework specifically express app and integrates directly with it
- `MongoMemoryServer` is used to to create temporary databases, preventing pollution from run to run

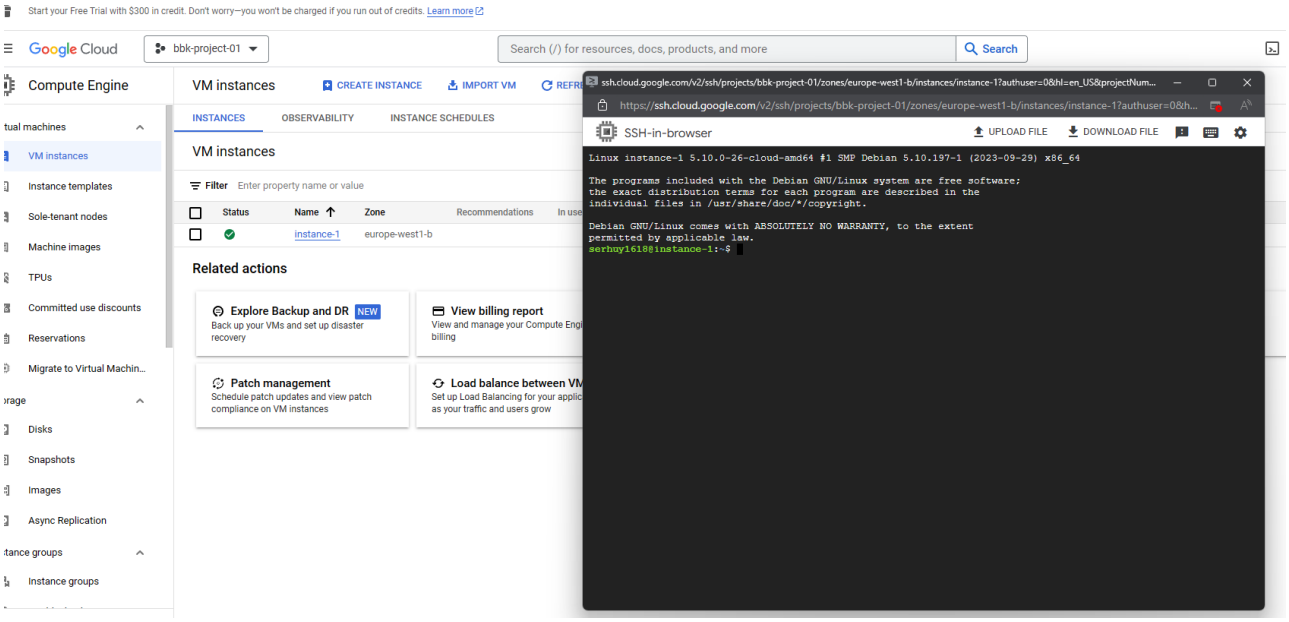
Test Coverage

- `src/test/e2e.test.ts` contains all the tests specified by the worksheet
- `src/test/auth.test.ts` covers the sign up process
- `src/test/expired.test.ts` covers the behaviour of expired posts
- `src/test/post.test.ts` covers the behaviour of posting and commenting on the app
- `src/test/OrderBy.test.ts` contains all the tests specific to the order of listing posts
- `src/test/utils.ts` contains code for common functionality used in the tests

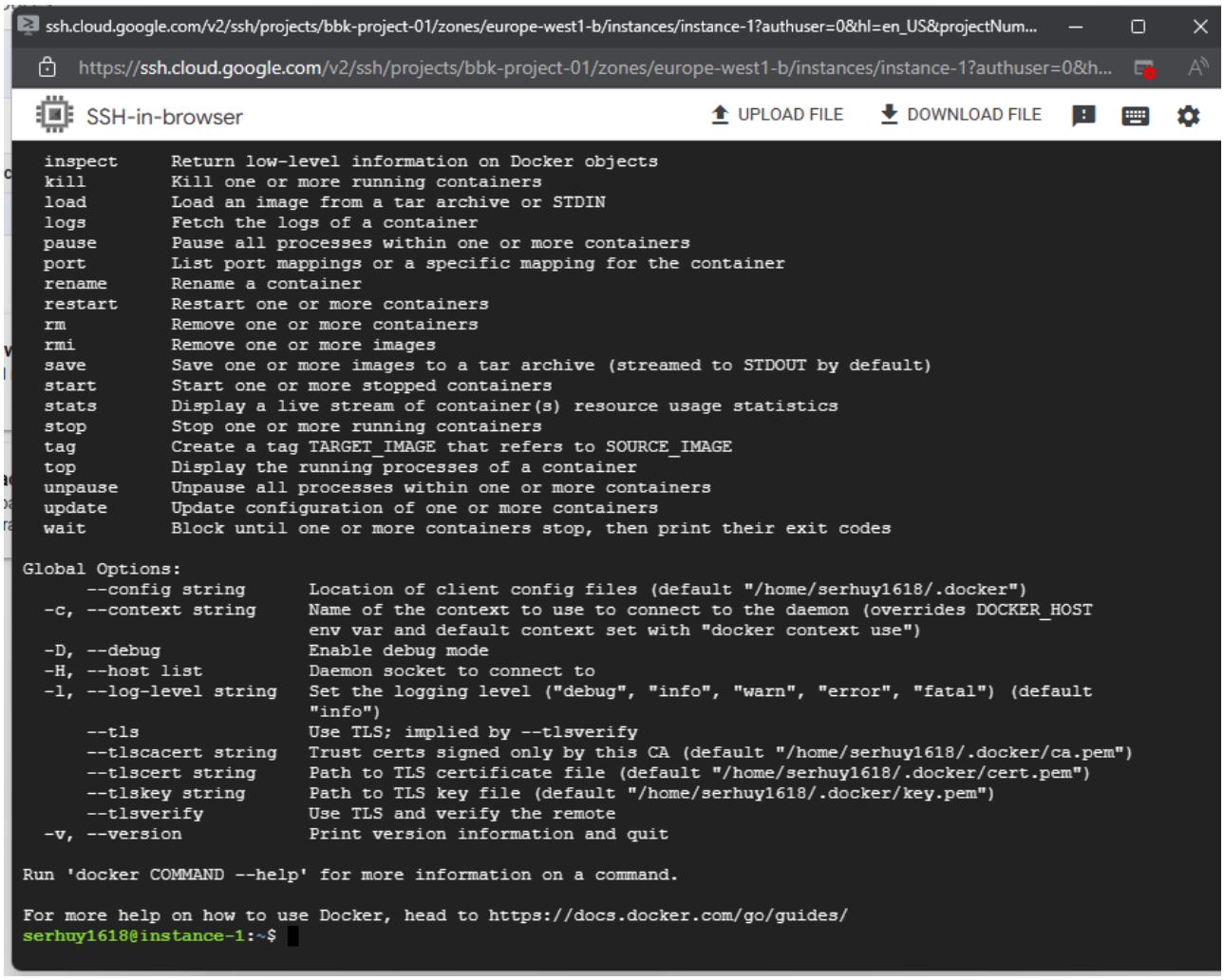
Complete run of all the tests

Phase E - Deploying the application to GCP

1. starting A google compute engine instance



2. install docker



3. clone the repo

```
See 'git help git' for an overview of the system.
serhuy1618@instance-1:~$ ls
serhuy1618@instance-1:~$ ls -a
.  ..  .bash_logout  .bashrc  .profile  .ssh
serhuy1618@instance-1:~$ mkdir git
serhuy1618@instance-1:~$ cd git
serhuy1618@instance-1:~/git$ git clone https://github.com/Serhiy1/Cloud-computing-coursework---piazza.git
Cloning into 'Cloud-computing-coursework---piazza'...
remote: Enumerating objects: 252, done.
remote: Counting objects: 100% (252/252), done.
remote: Compressing objects: 100% (153/153), done.
remote: Total 252 (delta 116), reused 204 (delta 68), pack-reused 0
Receiving objects: 100% (252/252), 386.64 KiB | 9.21 MiB/s, done.
Resolving deltas: 100% (116/116), done.
serhuy1618@instance-1:~/git$ ls
Cloud-computing-coursework---piazza
serhuy1618@instance-1:~/git$ cd Cloud-computing-coursework---piazza/
serhuy1618@instance-1:~/git/Cloud-computing-coursework---piazza$
```

4. Create the .env file with all the settings

```
Cloud-computing-coursework---piazza
serhuy1618@instance-1:~/git$ cd Cloud-computing-coursework---piazza/
serhuy1618@instance-1:~/git/Cloud-computing-coursework---piazza$ touch .env
serhuy1618@instance-1:~/git/Cloud-computing-coursework---piazza$ nano .env
serhuy1618@instance-1:~/git/Cloud-computing-coursework---piazza$
```

5. Build the app with the following command `docker build -t piazza -f .devcontainer/Dockerfile .`

```
serhuy1618@instance-2:~/git/Cloud-computing-coursework---piazza$ sudo docker build -t piazza -f .devcontainer/Dockerfile .
[+] Building 103.1s (13/13) FINISHED                                                                                    docker:default
 => [internal] load .dockerignore                                                                                                0.0s
 => => transferring context: 2B                                                                                                  0.0s
 => [internal] load build definition from Dockerfile                                                                             0.1s
 => => transferring dockerfile: 577B                                                                                             0.0s
 => [internal] load metadata for docker.io/library/node:lts-bullseye                                                             0.7s
 => [1/8] FROM docker.io/library/node:lts-bullseye@sha256:c1b3e61fa0fde701bdb45032796bc4bb020a8c53f06cba5765584b620087fa03      29.7s
 => => resolve docker.io/library/node:lts-bullseye@sha256:c1b3e61fa0fde701bdb45032796bc4bb020a8c53f06cba5765584b620087fa03        0.0s
 => => sha256:c1b3e61fa0fde701bdb45032796bc4bb020a8c53f06cba5765584b620087fa03 1.21kB / 1.21kB                                   0.0s
 => => sha256:bc51e804b01c020aedddc0f3c54622ccf74ac7380f9d9c0a5de2d1a2a851124c 7.52kB / 7.52kB                                   0.0s
 => => sha256:c7b1e60e9d5a0f16eb1f998245666f7a64a44f8b1f2317bd31e8a658150c23d3 54.60MB / 54.60MB                                 1.3s
 => => sha256:87d6fd97b714be5f3c2f8439fa36abd7a3032ef8a494330c46724c7d26c0cc14 2.00kB / 2.00kB                                   0.0s
 => => sha256:d1da99c2f14827498c4a9bb3623ae909b44564bdabad1802f064169069df81fb 55.06MB / 55.06MB                                 1.0s
 => => sha256:577ff23cfe55ac8872bc433ce99971a34011e7a15f7c8afa3d6492c78d6d23e5 15.76MB / 15.76MB                                 0.6s
 => => sha256:beefab36cbfedf8896b5f9f0bc33336fa13c0f01a8cb2333128dd247895a5f3b 196.88MB / 196.88MB                               4.3s
 => => extracting sha256:d1da99c2f14827498c4a9bb3623ae909b44564bdabad1802f064169069df81fb                                        4.9s
 => => sha256:f2726f1819ba625661f62225318c04ab8d4cffeab7daf6d953aec264d37e6dca 47.93MB / 47.93MB                                 2.3s
 => => sha256:ea2a9dfb8aae8dab2c7f0463881c06b387546002025b6bc9279e468c5c1fa93a 4.20kB / 4.20kB                                   1.5s
 => => sha256:0467cf59a39d0458dfc2d0a28085a8d032bb91bc162db2a789d0b2fc2b6a1a53 2.21MB / 2.21MB                                   1.7s
 => => sha256:603301d673bd50cb291cdb2552f835580c5b3e9172fe4ed5cddb6361436794d2 452B / 452B                                       1.8s
 => => extracting sha256:577ff23cfe55ac8872bc433ce99971a34011e7a15f7c8afa3d6492c78d6d23e5                                        0.7s
 => => extracting sha256:c7b1e60e9d5a0f16eb1f998245666f7a64a44f8b1f2317bd31e8a658150c23d3                                        4.2s
 => => extracting sha256:beefab36cbfedf8896b5f9f0bc33336fa13c0f01a8cb2333128dd247895a5f3b                                       11.5s
 => => extracting sha256:ea2a9dfb8aae8dab2c7f0463881c06b387546002025b6bc9279e468c5c1fa93a                                        0.0s
 => => extracting sha256:f2726f1819ba625661f62225318c04ab8d4cffeab7daf6d953aec264d37e6dca                                        4.0s
 => => extracting sha256:0467cf59a39d0458dfc2d0a28085a8d032bb91bc162db2a789d0b2fc2b6a1a53                                        0.1s
 => => extracting sha256:603301d673bd50cb291cdb2552f835580c5b3e9172fe4ed5cddb6361436794d2                                        0.0s
 => [internal] load build context                                                                                                0.0s
 => => transferring context: 408.78kB                                                                                            0.0s
 => [2/8] RUN apt update     && apt install -y --no-install-recommends sudo     && apt autoremove -y     && rm -rf /var/lib/apt/lists/*     && ech  7.7s
 => [3/8] ADD src /src                                                                                                           0.1s
 => [4/8] ADD tsconfig.json /tsconfig.json                                                                                       0.1s
 => [5/8] ADD package-lock.json /package-lock.json                                                                               0.1s
 => [6/8] ADD package.json /package.json                                                                                         0.1s
 => [7/8] RUN npm ci                                                                                                            48.0s
 => [8/8] RUN npm run build                                                                                                     10.6s
 => exporting to image                                                                                                           6.0s
 => => exporting layers                                                                                                          6.0s
 => => writing image sha256:57a0c82d256136524789622 7cbd9c10af44e3bb2efdcc5413ee274eee4b9ab5c                                    0.0s
 => => naming to docker.io/library/piazza                                                                                        0.0s
serhuy1618@instance-2:~/git/Cloud-computing-coursework---piazza$
```

6. Run the app with the following command `docker run -p 80:80 --env-file .env piazza`

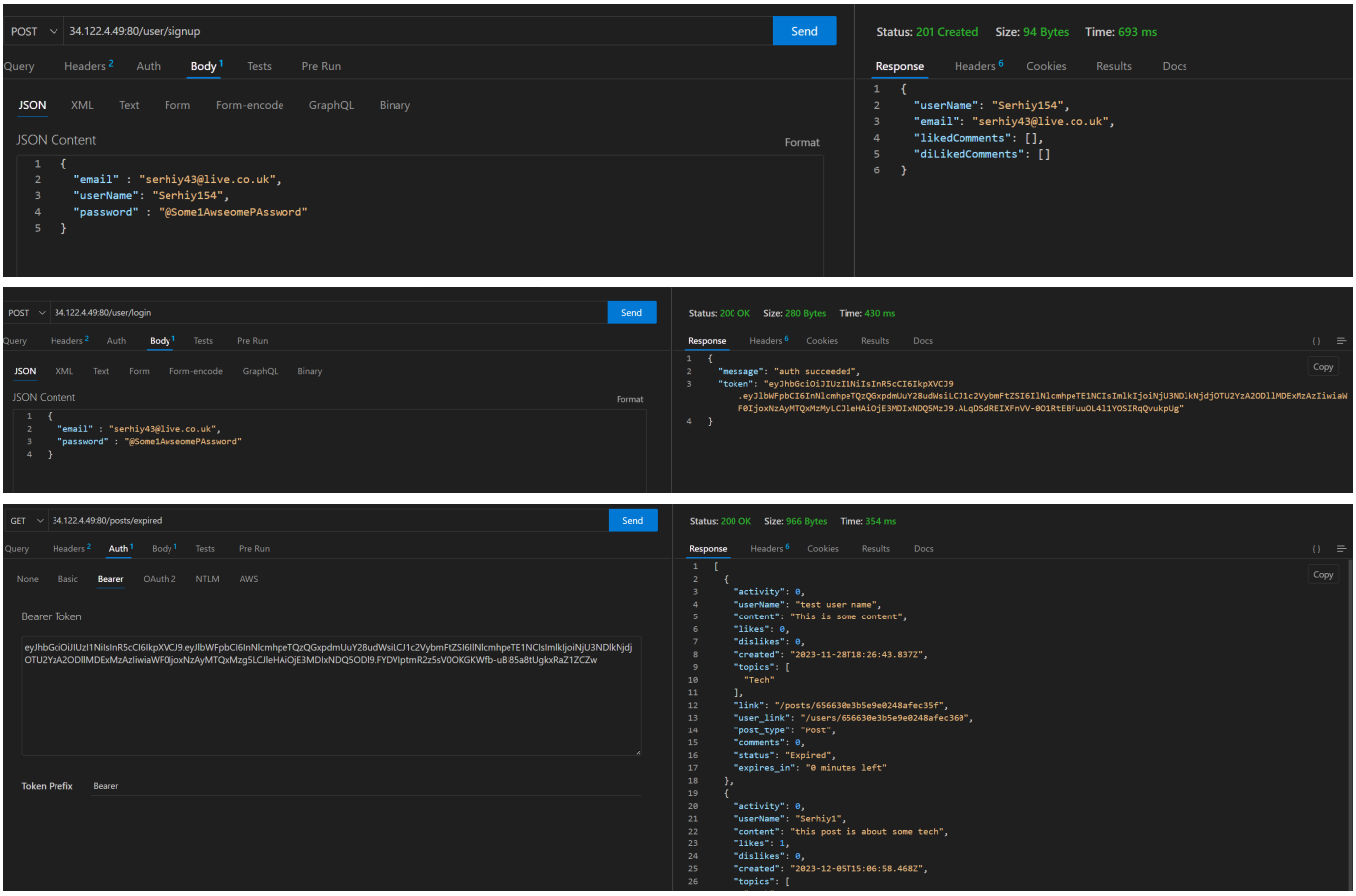
```
serhuy1618@instance-2:~/git/Cloud-computing-coursework---piazza$ sudo docker run -p 80:80 --env-file .env piazza

> piazza-coursework-serhiy@1.0.0 start
> node dist/app/server.js

listening on port 80
```

7. Testing the app works across the internet

Note the Public IP address that is being used



Phase G - Report Additional information

This Report was written as the application was being developed.

Tutorial Material

1. I learnt about express using this youTube tutorial series:
2. I followed this to setup typescript for myp project https://blog.logrocket.com/how-to-set-up-node-typescript-express/
3. The Jest tutorial was used to set up testing in the repository https://jestjs.io/docs/getting-started

Phase H - Submit quality scripts

Please see the Testing section for details on where to locate the tests and how to run them