

## Contents

Introduction .....	3
Resources .....	3
Database .....	4
Post Schema .....	4
User Schema.....	5
Database Structure .....	5
File Structure .....	7
Files and folders directory structure of the application is given: .....	8
File Details .....	8
Design .....	10
Implementation.....	10
Installing packages and libraries.....	10
Environment Setup .....	11
Basic Setup to run the node.js code. ....	11
Libraries.....	13
Authentication and Authorization using JWT.....	13
RESTful API Implementation .....	14
CRUD Functionalities .....	14
Testing the application.....	15
TC1.Olga, Nick, Mary, and Nestor register and are ready to access the Piazza API.....	15
Users in the DB .....	17
TC2. Olga, Nick, Mary, and Nestor use the oAuth v2 authorisation service to register and get their tokens. ....	18
TC3 Olga makes a call to the API without using her token. This call should be unsuccessful as the user is unauthorised.....	20
T4.Olga posts a message in the Tech topic with an expiration time (e.g. 5 minutes) using her token. After the end of the expiration time, the message will not accept any further user interactions (likes, dislikes, or comments). ....	21
Posted a message with 5 min expiration time.....	21
Msg in the DB .....	21
Like a post that is expired.....	22
Dislike a post that is expired.....	22
Add comment to expired post.....	23

T5. Nick posts a message in the Tech topic with an expiration time using his token. ....	23
T6. Mary posts a message in the Tech topic with an expiration time using her token. ....	24
TC7. Nick and Olga browse all the available posts in the Tech topic; three posts should be available with zero likes, zero dislikes and no comments. ....	25
Nick browses Messages in Tech Topic using his login .....	26
Olga browses Messages in Tech Topic .....	27
TC8. Nick and Olga “like” Mary’s post on the Tech topic. ....	28
Nick likes Mary’s post on the tech topic. ....	28
Olga like’s Mary’s post on the tech topic .....	28
TC9. Nestor “likes” Nick’s post and “dislikes” Mary’s on the Tech topic. ....	29
Nestor ‘likes’ Nick’s post .....	29
Nestor dislikes ‘Mary’s’ post .....	30
TC10. Nick browses all the available posts on the Tech topic; at this stage, he can see the number of likes and dislikes for each post (Mary has two likes and one dislike, and Nick has one like). There are no comments made yet. ....	31
TC11. Mary likes her post on the Tech topic. This call should be unsuccessful; in Piazza, a post owner cannot like their messages. ....	32
TC12. Nick and Olga comment on Mary’s post on the Tech topic in a round-robin fashion (one after the other, adding at least two comments each) .....	33
Nick comment on Mary’s post .....	34
Olga comment on Mary’s post .....	35
TC13. Nick browses all the available posts in the Tech topic; at this stage, he can see the number of likes and dislikes of each post and the comments made. ....	35
TC14. Nestor posts a message in the Health topic with an expiration time using her token .....	37
TC15. Mary browses all the available posts on the health topic; at this stage, she can see only Nestor’s post. ....	37
TC16. Mary posts a comment in Nestor’s message on the Health topic. ....	38
TC17. Mary dislikes Nestor’s message on the health topic after the end of post-expiration time. This should fail. ....	39
TC18. Nestor browses all the messages on the Health topic. There should be only one post (his own) with one comment (Mary’s). ....	40
TC19. Nick browses all the expired messages on the Sports topic. These should be empty .....	41
TC20. Nestor queries for an active post with the highest interest (maximum number of likes and dislikes) in the Tech topic. This should be Mary’s post. ....	42
Deploy your Piazza project into a VM using Docker .....	43
6. Create a docker file named dockerfile .....	45
Contents of the Dockerfile .....	45
7. Docker image created using the command .....	45

Deploy Piazza application in Kubernetes .....	46
-----------------------------------------------	----

## Introduction

Computing resources are costly for single organizational usage, hence using offered computing services results in flexible, speedy and relatively fewer operating costs. The services include software, database, network, servers, analysis on the internet aka cloud. Software-as-a-Service is a method that offers access to software to its remote users from the cloud. The purchase and installation of a software is costlier than subscribing to it.

Piazza is a SaaS based cloud application that allows authenticated users to post their thoughts for other users authorised to the software. The users like and comment on each other's posts which builds a community altogether. This report concentrates on implementation, development and testing of a Piazza application.

The report starts with a brief introduction about the domain. Next section focuses on the resources such as tools and platforms used in implementation. Next, database model and file structure are described followed by UML modelling of the application. The later sections cover the installation of libraries, the authentication and validation using JWT (Copes, 2023), development of RESTful API (including CRUD operations), test cases generation and deployment.

### **Functionalities offered by Piazza are as listed below:**

1. OAuth v2.0 protocol authorises the users
2. Authorised users can share their thoughts via text.
3. All posts are available to read to the valid users
4. Users can comment, like, dislike on the posts
5. The Post owner can set expiry time after which the post cannot be liked, disliked or commented.
6. The posts can be retrieved by a particular topic

## Resources

Resources The tools and platforms used in implementation of this application are as listed below:

1. GCP – To create a VM instance for deployment

Deepa Karthick  
Candidate Num: Y106847

2. Node JS – Writing code in JS (Chris, 2021)
3. GitHub – hosting code (Kaelin,2022)
4. MongoDB – Data storage (MongoDB tutorial, 2022)
5. Postman – design and test the generated test cases (Node.js, 2022)

## Database

The database is designed using MongoDB for piazza application. The schema design is as described:

### Post Schema

```
models > JS PPOST > PPOSTSchema > post_owner > type
1  const mongoose = require('mongoose')
2
3  const PPOSTSchema = mongoose.Schema({
4    post_title:{
5      type:String
6    },
7    post_topic:{
8      type:String,
9      enum:['Politics', 'Health', 'Sport', 'Tech']
10   },
11
12   likes_count:{
13     type:Number,
14     default:0
15   },
16   dislikes_count:{
17     type:Number,
18     default:0
19   },
20
21   comments:{
22     type:Array
23   },
24
25   post_date:{
26     type>Date,
27     default:Date.now
28   },
29
30   post_owner:{
31     type:String
32   },
33
34   post_time:{
35     type>Date
36   },
37
38   post_owner_id:{
39     type:String
40   },
41
42   post_status:{
```

## User Schema

```
models > JS User.js > [?] userSchema
1  const mongoose = require('mongoose')
2
3  const userSchema = mongoose.Schema({
4    username:{
5      type:String,
6      require:true,
7      min:3,
8      max:256
9    },
10   email:{
11     type:String,
12     require:true,
13     min:6,
14     max:256
15   },
16   password:{
17     type:String,
18     require:true,
19     min:6,
20     max:1024
21   },
22   date:{
23     type>Date,
24     default>Date.now
25   }
26 })
27 module.exports=mongoose.model('users',userSchema)
```

## Database Structure

Mongo DB is used to manage the data. New database 'piazza' is created, and two collections posts and users are created. Posts holds all the messages and users hold the user details.

+ Create Database

Q Search Namespaces

CW\_PiazzaPost

CW\_PiazzaPost

MiniFilms

Minipost

piazza

posts

users

test

piazza.posts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.28KB TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 72KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

Filter

Type a query: { field: 'value' }

\_id: ObjectId('6572479a6600eb31d071d838')

post\_title: "Tech details"

post\_topic: "Tech"

likes\_count: 0

dislikes\_count: 0

comments: Array (empty)

post\_owner: "olga"

post\_time: 2023-12-07T22:35:00.000+00:00

post\_owner\_id: "65722e58477d8317b02e1ef8"

post\_status: "Live"

post\_date: 2023-12-07T22:30:50.844+00:00

\_\_v: 0

\_id: ObjectId('657365e434b5b817a893d605')

post\_title: "Tech details-Nick"

post\_topic: "Tech"

likes\_count: 1

dislikes\_count: 0

comments: Array (empty)

post\_owner: "Nick"

post\_time: 2023-12-09T22:30:00.000+00:00

+ Create Database

Q Search Namespaces

CW\_PiazzaPost

CW\_PiazzaPost

MiniFilms

Minipost

piazza

posts

users

test

piazza.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 668B TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes

Filter

Type a query: { field: 'value' }

RESET

Apply

C

QUERY RESULTS: 1-4 OF 4

\_id: ObjectId('65722e58477d8317b02e1ef8')

username: "olga"

email: "olga@piazza.com"

password: "\$2a\$05\$GcLMh1LWhLP47Ue6NejLnu6E2aPgzbMfSrZphJ/aKdtIkmz8BdPZ6"

date: 2023-12-07T20:43:04.334+00:00

\_\_v: 0

\_id: ObjectId('65722e8866bbe631b071d832')

username: "Nick"

email: "Nick@piazza.com"

password: "\$2a\$05\$40ZvnJweyVSXypFqm3vFr.qqJ3L.9mDxK5KV7N48MbNVvAveBxZ2"

date: 2023-12-07T20:43:52.436+00:00

\_\_v: 0

\_id: ObjectId('65722eb6477d8317b02e1efb')

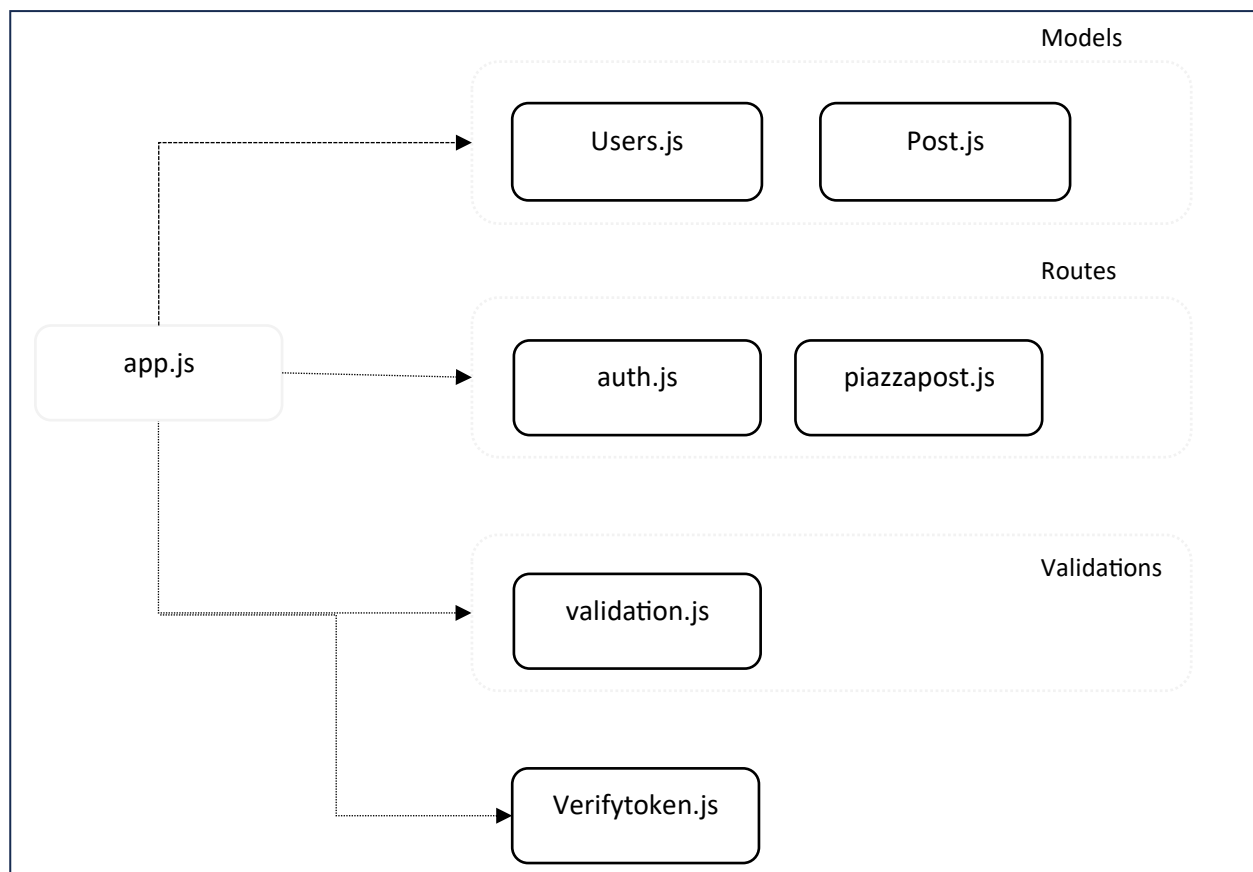
System Status: All Good

©2023 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

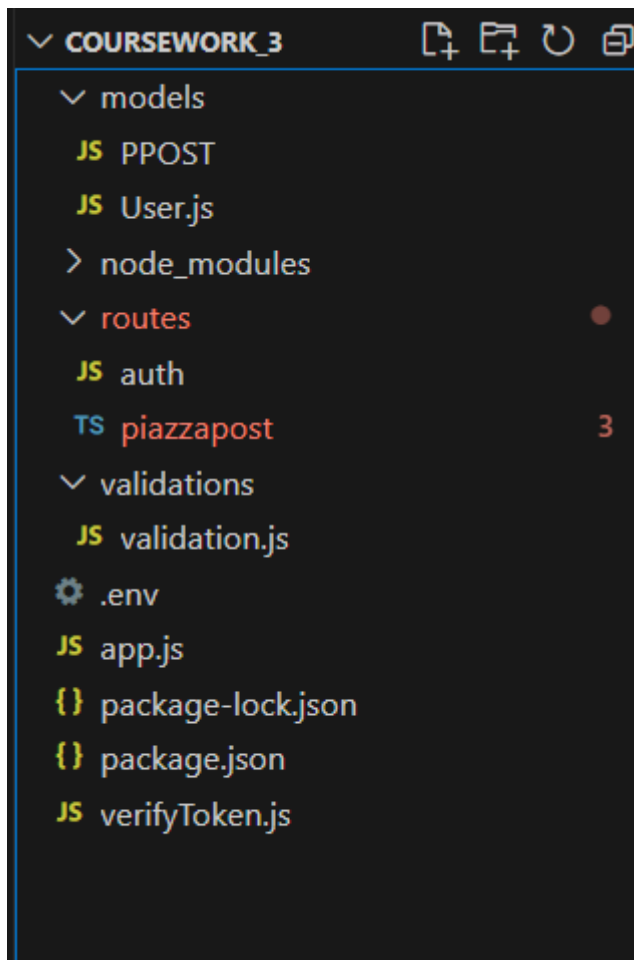
6

## File Structure

Files in the application are organized in MVC fashion; where models are stored in folder Models, View in the folder Routes and Controller in the Validations folder. The figure below describes detailed file structure formed. The app has three pages, home: app.js; auth.js, piazza.js that has all the feeds and validations.js to validate the input from the user at registration and login sessions. verifyToken.js contains code in relation to authentication discussed in later sections.



Files and folders directory structure of the application is given:



### File Details

1. **app.js file:** This is the entry point to the application where middleware is created, API endpoints are mapped, connection to MongoDB is established and finally the server is set on port 3000.

2. **models folder:** Includes the files to create the model to define each database structure

a. Post.js: A schema to define the posts database table and export the model.

- Post\_title – String – given by user
- Post\_topic – Enum and holds four values 'Politics' , 'Sport', 'Health' and 'Tech'
- Likes\_count – Every time a post is liked, the counter is increased.
- Dislikes\_count - Every time a post is disliked, the counter is increased.
- Comments – creates an array of comments when input by the user.
- Post\_date – created by the system for every post
- Post\_time – expiry date time set for each post.
- Post\_owner\_id – created from the login authentication token to identify post owner.
- Post\_status – Enum can hold 'live' and 'expired' calculated by the system. Defaults live and sets it to expired if current date > post\_time.



- a. User.js: A schema to define the user's database table and export the model.
  - Username – String and needs to be minimum 3 char and max 256, mandatory
  - Email – email address and min 6 characters, mandatory
  - password – string, min 6 characters, mandatory.

3. **routes folder:** Files to create the routes to the api endpoints

a. posts.js: Imports the Post schema model and do the following operations with user authentication.

- i. post- for posting a new post,
- ii. get - to get **all the posts** and get **posts by id**, to get **posts by topic**, to get posts by **topic and status**.
- iii. post-to like a post by incrementing the likes\_count
- iv. Post – to dislike a post by incrementing the dislikes\_count
- v. post-to comment a particular post

**b. user-auth.js:** Imports the User schema model and do the validations from validation.js file and after successful login give the auth-token to the user.

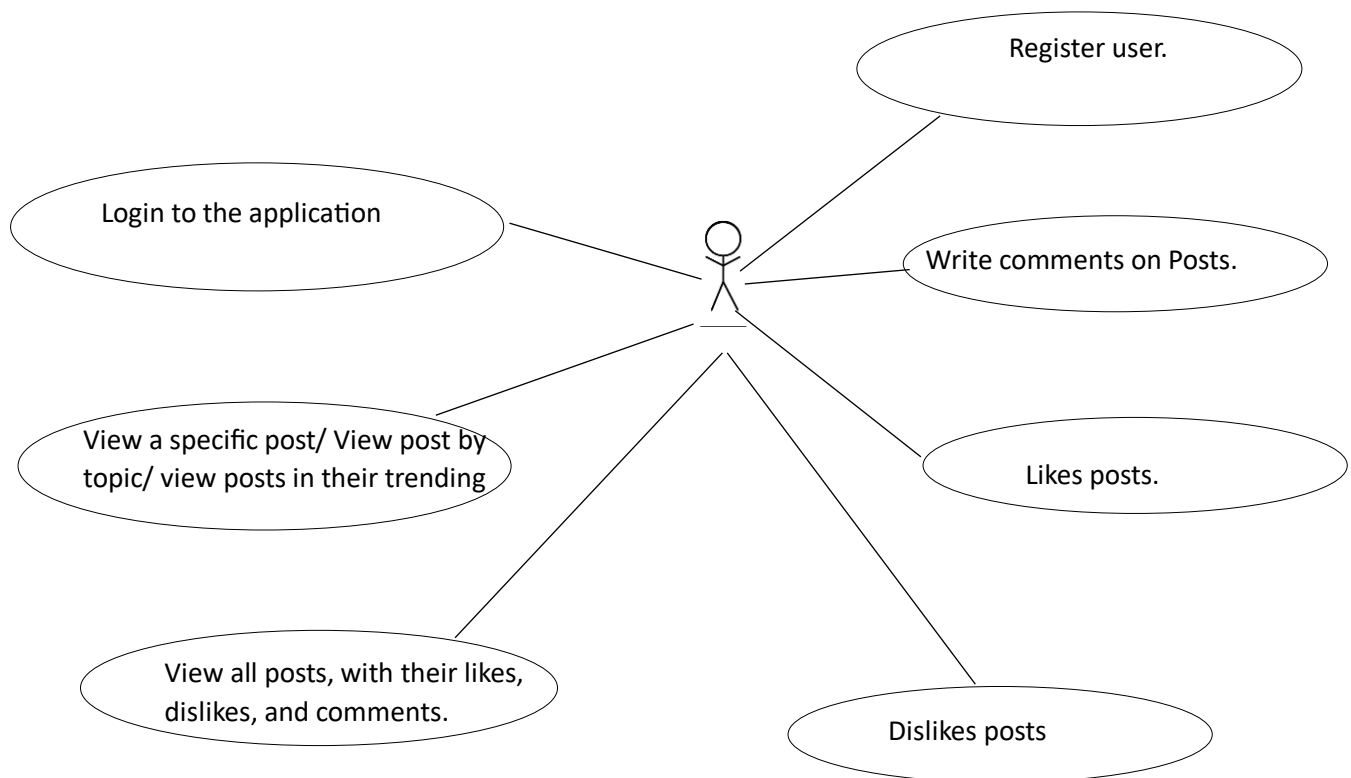
4. **validations folder:** a. validation.js: This file contains all the validations for user registration and login.

5. **verifyToken.js file:** This file contains the code for Token verification once it is provided by the user for CRUD operations after login

6. **env file:** This file contains the path DB\_CONNECTOR to connect the application to the MongoDB and TOKEN\_SECRET to generate token for authentication.

## Design

### Use case Diagram



## Implementation

### Installing packages and libraries

1. npm init: This command generates the package. Json file in the root folder “Coursework\_3”, which contains all the project details, scripts and dependencies used for the project.

2. npm install: The following are the dependencies installed:

- a. express: framework is required to develop web RESTful APIs
- b. nodemon: very helpful in restarting the application as soon as the code files are changed and saved.
- c. mongoose: this package is useful when MongoDB holds application data. It models, validates and manipulates data.
- d. body-parser :helps in parsing the body that includes information when operations like POST, PATCH or PUT are involved.
- e. dotenv: – newly added feature that loads environment variables added into the .env file created by the developer.
- f. joi: A package for object schema descriptions and validations.
- g. jsonwebtoken: provides strong authentication by generating authentication tokens

h. bcryptjs: To encrypt and decrypt the passwords.

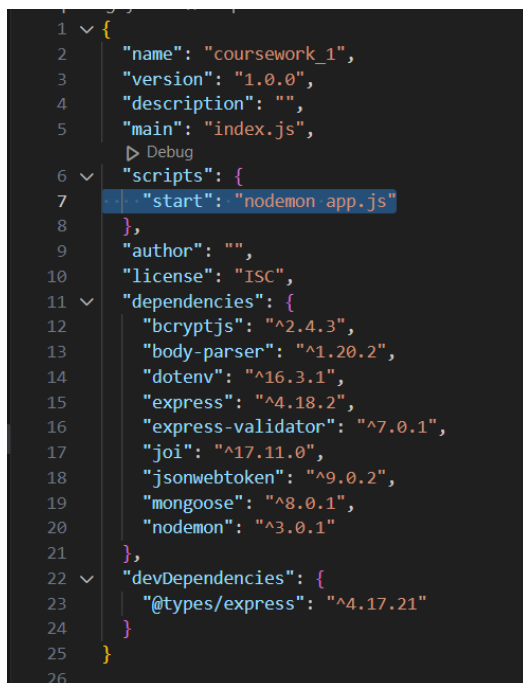
3. npm start: To start the server listening at port 3000, written in file app.js

## Environment Setup

1. The packages are first installed using the terminal of Node JS. The package.json file contains the starting point of the application (figure 6). In Scripts, test:"echo..." is changed with start:"nodemon app.js". This confirms that the starting point of this application is app.js. Every time start command is invoked; it asks nodemon to run app.js app.

## Basic Setup to run the node.js code.

These are the basic commands used:



```
1 {
2   "name": "coursework_1",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "start": "nodemon app.js"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "bcryptjs": "^2.4.3",
13    "body-parser": "^1.20.2",
14    "dotenv": "^16.3.1",
15    "express": "^4.18.2",
16    "express-validator": "^7.0.1",
17    "joi": "^17.11.0",
18    "jsonwebtoken": "^9.0.2",
19    "mongoose": "^8.0.1",
20    "nodemon": "^3.0.1"
21  },
22  "devDependencies": {
23    "@types/express": "^4.17.21"
24  }
25 }
```

specification of app.js as starting point

2. npm start – command starts the server and the web application is ready for testing as shown in figure.

The screenshot shows a VS Code editor with a file explorer on the left containing files like models, PPOST, User.js, node\_modules, routes, auth, piazzapost, validations, validation.js, .env, app.js, package-lock.json, package.json, and verifyToken.js. The main editor displays the app.js file with the following code:

```

1  const express = require('express')
2  const app = express()
3  require('dotenv').config()
4
5  const bodyParser = require('body-parser')
6  const mongoose = require('mongoose')
7
8  const piazzapostRoute = require('./routes/piazzapost')
9
10 const authRoute = require('./routes/auth')
11
12 app.use(bodyParser.json())
13
14
15 app.use('/piazzapost', piazzapostRoute)
16 app.use('/user', authRoute)
17
18
19 app.get('/', (req, res) => {
20   res.send('Piazza test')
21 })
22
23
24 const MURL = 'mongodb+srv://deepakarthick2022:cloud1234@cluster0.vqy4urh.mongodb.net/piazza?retryWrites=true&w=majority'
25 mongoose.connect(MURL)
26
27 app.listen(3000, () => {
28   console.log('Your server is up and running...')
29 })

```

The terminal at the bottom shows the following output:

```

SUCCESS: The process with PID 29416 has been terminated.
PS C:\Users\karth\OneDrive\Deepa\Birbeck\cloud computing\coursework_2nd\Coursework_3> npm start
> coursework_1@1.0.0 start
> nodemon app.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node app.js'
Your server is up and running...

```

npm start

- The web application is deployed in virtual machine and has five replicas of the service implemented in Kubernetes using the deployment.yaml and service.yaml so the application can be accessed with url: 34.171.134.125 given by the 'get services' in Kubernetes, that invokes app.js file as shown in figure.

The screenshot shows a Cloud Shell terminal with the following output:

```

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cloudclass1-401818.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
deepakarthick2022@cloudshell:~ (cloudclass1-401818) $ kubectl get services
NAME                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes           ClusterIP           10.32.0.1       <none>            443/TCP          2d
piazza-post-service  LoadBalancer       10.32.6.130     34.171.134.125   80:30044/TCP     29h
deepakarthick2022@cloudshell:~ (cloudclass1-401818) $

```

To get the External IP to run the application

- Endpoints

API endpoints	Description
POST - 34.171.134.125/user/register	To register a new user
POST- 34.171.134.125/user/login	To login as a registered user
POST - 34.171.134.125/piazzapost	To create a new post
POST - 34.171.134.125/piazzapost/postId/likes	To like a particular post
POST - 34.171.134.125/piazzapost/postId/dislikes	To dislike a particular post
POST - 34.171.134.125/piazzapost/postId/comments	To comment on a post
GET - 34.171.134.125/piazzapost	To retrieve all the posts

GET - 34.171.134.125/piazzapost/postId	To retrieve post by specific ID
GET - 34.171.134.125/piazzapost?topic=Sport	To get the posts corresponding to Sport Topic
GET- 34.171.134.125/ piazzapost?topic=Sport&status=expired	To get posts for Sport topic and expired status
GET- 34.171.134.125/piazzapost?topic=Sport&status=live	To get posts for Sport topic and live status
GET- 34.171.134.125/piazzapost/trending	To get the post that has most number of likes and dislikes

## Libraries

The libraries express and nodemon are imported at the beginning before anything else in the application.

## Authentication and Authorization using JWT

### Authentication

Using email and password, user is allowed to login to the system. If the username does not exist in the database, it prompts 'User doesn't exist'. Similarly, for incorrect password, it displays the message to the user. Once the user is allowed to login with correct credentials, JWT creates an authentication token for the logged in user. This token is important to pass as parameter in the header to access the Piazza application further. The users in the application are authenticated using OAuth 2.0 protocol as shown in figure 9.

### Authorization

The users are expected to register by providing their username, email and password. The application checks for the existing email and username, if either exists, it prompts that user that email or username exists and hence duplicate registration is not accepted. The user is not allowed to comment or like own posts. If the attempt is made the application sends a message that the operation is not allowed. No other user can edit or delete posts of the other users.

Figure 9 Authentication with OAuth 2.0

## RESTful API Implementation

The implementation of RESTful API involves following basic and necessary steps:

1. Directory – the directories are created as per the requirements of the application.
2. App Express – the express and nodemon libraries are imported.
3. Schema – The models are created as per requirements.
4. Authentication – JWT authentication is important for secured application.

## CRUD Functionalities

The application effectively implemented CRUD functionalities.

Create – creates users, posts and comments

Read – reads data

Update – updates posts

Delete – deletes posts

Deepa Karthick  
Candidate Num: Y106847

Testing the application.

TC1.Olga, Nick, Mary, and Nestor register and are ready to access the Piazza API.

The screenshot displays a REST client interface for a POST request to the endpoint `34.171.134.125/user/register`. The request body is a JSON object with the following fields: `username` (olga), `email` (olga@piazza.com), and `password` (olga12). The response status is 200 OK, and the response body is a JSON object containing: `_id` (65722e58477d8317b02e1ef8), `date` (2023-12-07T20:43:04.334Z), and `__v` (0).

```
POST 34.171.134.125/user/register

Params Authorization Headers (9) Body Pre-request Script Tests Settings
● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON

1
2 "username": "olga",
3 "email": "olga@piazza.com",
4 "password": "olga12"
5

Body Cookies Headers (7) Test Results Status: 200 OK Time: 458 ms
Pretty Raw Preview Visualize JSON

1
2 "username": "olga",
3 "email": "olga@piazza.com",
4 "password": "$2a$05$GcLMh1LWh1P47Ue6NejLnu6E2aPgzbMfsrZphJ/aKdtIkmz8BdPZ6",
5 "_id": "65722e58477d8317b02e1ef8",
6 "date": "2023-12-07T20:43:04.334Z",
7 "__v": 0
8
```

POST 34.171.134.125/user/register

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "username": "Nick",
3   "email": "Nick@piazza.com",
4   "password": "Nick12"
5 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 391 ms Size: 430 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "username": "Nick",
3   "email": "Nick@piazza.com",
4   "password": "$2a$05$40ZvnJweyVSXypFqm3vFi.qqJ3L.9mDxK5KV7N48MbNVvAveBxZ2",
5   "_id": "65722e8866bbe631b071d832",
6   "date": "2023-12-07T20:43:52.436Z",
7   "__v": 0
8 }
```

POST 34.171.134.125/user/register

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "username": "Mary",
3   "email": "Mary@piazza.com",
4   "password": "Mary12"
5 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 343 ms Size: 430 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "username": "Mary",
3   "email": "Mary@piazza.com",
4   "password": "$2a$05$KVmn7PEVmapq4gCcx7jhduaGEbCH.Xrqx92dZvMjG55Z20tSjUJD0",
5   "_id": "65722eb6477d8317b02e1efb",
6   "date": "2023-12-07T20:44:38.717Z",
7   "__v": 0
8 }
```



Deepa Karthick  
Candidate Num: Y106847

POST

34.171.134.125/user/register

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

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

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

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506</

## Users in the DB

QUERY RESULTS: 1-4 OF 4

```

_id: ObjectId('65722e58477d8317b02e1ef8')
username: "olga"
email: "olga@piazza.com"
password: "$2a$05$GcLMh1LWhlP47Ue6NejLnu6E2aPgzbMfsrZphJ/aKdtIkmz8BdPZ6"
date: 2023-12-07T20:43:04.334+00:00
_v: 0

```

```

_id: ObjectId('65722e8866bbe631b071d832')
username: "Nick"
email: "Nick@piazza.com"
password: "$2a$05$40ZvnJweyVSXypFqm3vFr.qqJ3L.9mDxK5KV7N48MbNvvvAveBxZ2"
date: 2023-12-07T20:43:52.436+00:00
__v: 0

```

```
_id: ObjectId('65722eb6477d8317b02e1efb')
username: "Mary"
email: "Mary@piazza.com"
password: "$2a$05$KVMn7PEVmapq4gCcx7jhdua6EbcH.Xrqx92dZvMjG55Z20tSjUJD0"
date: 2023-12-07T20:44:38.717+00:00
__v: 0
```

```
_id: ObjectId('65722ed466bbe631b071d835')
username: "Nestor"
email: "Nestor@piazza.com"
password: "$2a$05$gdjTVoPLLSQd/gdLrW5AU0m4du5j0vA56Ix3fN5Mi49yPVgvVILLm"
date: 2023-12-07T20:45:08.964+00:00
__v: 0
```

TC2. Olga, Nick, Mary, and Nestor use the oAuth v2 authorisation service to register and get their tokens.

The screenshot displays a REST client interface. At the top, a POST request is configured to the URL `34.171.134.125/user/login`. The request body is in JSON format, containing the following fields:

```
{
  "email": "olga@piazza.com",
  "password": "olga12"
}
```

The response status is 200 OK, with a time of 351 ms and a size of 565 B. The response body is also in JSON format, containing an authentication token:

```
{
  "auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfcmFudD8iZm9zaW9yZj48E3YjAyZTF1Zjg1LCJpYXQiOiE3MDE5ODIwMzd9.PvBLeoAiRwZ8wcuQCostrUfu0KCYvZj48EbJCbPu0WQ"
}
```

Deepa Karthick  
Candidate Num: Y106847

POST

34.171.134.125/user/login

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

"email": "Nick@piazza.com",

"password": "Nick12"

BodyCookiesHeaders (8)Test Results

Status: 200 OKTime: 317 msSize: 565 BSav

PrettyRawPreviewVisualizeJSON

1

2

3

"auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTcyMmU4ODY2YmJlNjMxYjA3MMwQ4MzIiLCJpYXQiOiE3MDE5ODIwODZ9. JcwK-VCwCuo5I12\_9Kxht8CJAj0fEIr7F-wxBogjhRg"

POST

34.171.134.125/user/login

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

"email": "Mary@piazza.com",

"password": "Mary12"

BodyCookiesHeaders (8)Test Results

Status: 200 OKTime: 411 msSize: 565 BSav

PrettyRawPreviewVisualizeJSON

1

2

3

"auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTcyMmViNjQ3N2Q4MzE3YjAyZTF1ZmIiLCJpYXQiOiE3MDE5ODIxMjR9. y-f\_j\_ky3GE2RDPYbB6kJuvENi2MaZ5waof1fmqAOk"

Deepa Karthick  
Candidate Num: Y106847

POST 34.171.134.125/user/login

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▾

```
1 {
2   "email": "Nestor@piazza.com",
3   "password": "Nestor"
4 }
```

Body Cookies Headers (8) Test Results Status: 200 OK Time: 365 ms Size: 565 B Save

Pretty Raw Preview Visualize JSON ▾

```
1 {
2   "auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfawQ10iI2NTcyMmVkNDY2YmJlNjMxYjA3MmQ4MzU1LCJpYXQ10jE3MDE5ODIxNjB9.2yu4TH2c1Po71j0wJHYo3PQ_1FQFFsgYxHFX00PBVQM"
3 }
```

TC3 Olga makes a call to the API without using her token. This call should be unsuccessful as the user is unauthorised.

GET 34.171.134.125/piazzapost

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> Content-Length	<calculated when request is sent>	
<input checked="" type="checkbox"/> Host	<calculated when request is sent>	
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.35.0	
<input checked="" type="checkbox"/> Accept	*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input checked="" type="checkbox"/> auth-token		

body Cookies Headers (7) Test Results Status: 401 Unauthorized Time: 256 ms Size: 272 B Save

Pretty Raw Preview Visualize JSON ▾

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

Deepa Karthick  
Candidate Num: Y106847

T4.Olga posts a message in the Tech topic with an expiration time (e.g. 5 minutes) using her token. After the end of the expiration time, the message will not accept any further user interactions (likes, dislikes, or comments).

Posted a message with 5 min expiration time

The screenshot shows a REST client interface with a POST request to `34.171.134.125/piazzapost`. The request body is a JSON object with the following fields:

```
1 {
2   "post_title": "Tech details",
3   "post_topic": "Tech",
4   "post_owner": "olga",
5   "post_time": "2023-12-07T22:35:00.000Z"
6 }
```

The response status is 200 OK, with a time of 358 ms and a size of 538 B. The response body is a JSON object with the following fields:

```
1 {
2   "post_title": "Tech details",
3   "post_topic": "Tech",
4   "likes_count": 0,
5   "dislikes_count": 0,
6   "comments": [],
7   "post_owner": "olga",
8   "post_time": "2023-12-07T22:35:00.000Z",
9   "post_owner_id": "65722e58477d8317b02e1ef8",
10  "post_status": "live",
11  "_id": "6572479a66bbe631b071d838",
12  "post_date": "2023-12-07T22:30:50.844Z",
13  "__v": 0
14 }
```

The interface also shows a sidebar with a tree view containing `piazza`, `posts`, `users`, and `test`. The bottom status bar shows the system time as 22:30 on 07/12/2023.

Msg in the DB

▼ piazza  
| posts  
users  
► test

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('6572479a66bbe631b071d838')
post_title: "Tech details"
post_topic: "Tech"
likes_count: 0
dislikes_count: 0
comments: Array (empty)
post_owner: "olga"
post_time: 2023-12-07T22:35:00.000+00:00
post_owner_id: "65722e58477d8317b02e1ef8"
post_status: "live"
post_date: 2023-12-07T22:30:50.844+00:00
__v: 0
```

Like a post that is expired

POST

34.171.134.125/piazzapost/6572479a66bbe631b071d838/like

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.35.0	
<input checked="" type="checkbox"/>	Accept	*/*	
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/>	Connection	keep-alive	
<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 403 Forbidden Time: 394 ms Size: 298 B Save

Pretty

Raw

Preview

Visualize

JSON

```
1  {"message": "This post has expired and cannot be liked."}
2
3
```

Dislike a post that is expired.

POST

34.171.134.125/piazzapost/6572479a66bbe631b071d838/dislikes

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 403 Forbidden Time: 325 ms Size: 301 B Save

Pretty

Raw

Preview

Visualize

JSON

```
1  {"message": "This post has expired and cannot be disliked."}
2
3
```

Deepa Karthick  
Candidate Num: Y106847

## Add comment to expired post

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

- Method:** POST
- URL:** 34.171.134.125/piazzapost/6572479a66bbe631b071d838/comment
- Body:**

```
{
  "comments": "This is a tech story test for expired post"
}
```
- Response:** Status: 403 Forbidden, Time: 310 ms, Size: 307 B. The response body is: 

```
{
  "message": "This post has expired and comments cannot be added."
}
```

T5.Nick posts a message in the Tech topic with an expiration time using his token.

Deepa Karthick  
Candidate Num: Y106847

POST

34.171.134.125/piazzapost

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

"post\_title": "Tech details-Nick",

"post\_topic": "Tech",

"post\_owner": "Nick",

"post\_time": "2023-12-09T22:30:00.000Z"

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 326 ms

Size: 543 B

Save

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

"post\_title": "Tech details-Nick",

"post\_topic": "Tech",

"likes\_count": 0,

"dislikes\_count": 0,

"comments": [],

"post\_owner": "Nick",

"post\_time": "2023-12-09T22:30:00.000Z",

"post\_owner\_id": "65722e8866bbe631b071d832",

"post\_status": "live",

"\_id": "657365e434b5b817a893d605",

"post\_date": "2023-12-08T18:52:20.114Z",

"\_\_v": 0

T6.Mary posts a message in the Tech topic with an expiration time using her token.



The screenshot displays a REST client interface with a POST request to the URL `34.171.134.125/piazzapost`. The request body is a JSON object with the following fields:

```
1 {
2   "post_title": "Tech details-Nick",
3   "post_topic": "Tech",
4   "post_owner": "Mary",
5   "post_time": "2023-12-09T22:30:00.000Z"
6 }
```

The response status is 200 OK, with a time of 306 ms and a size of 543 B. The response body is a JSON object with the following fields:

```
1 {
2   "post_title": "Tech details-Nick",
3   "post_topic": "Tech",
4   "likes_count": 0,
5   "dislikes_count": 0,
6   "comments": [],
7   "post_owner": "Mary",
8   "post_time": "2023-12-09T22:30:00.000Z",
9   "post_owner_id": "65722eb6477d8317b02e1efb",
10  "post_status": "live",
11  "_id": "6573668734b5b817a893d607",
12  "post_date": "2023-12-08T18:55:03.676Z",
13  "__v": 0
14 }
```

TC7. Nick and Olga browse all the available posts in the Tech topic; three posts should be available with zero likes, zero dislikes and no comments.

Deepa Karthick  
Candidate Num: Y106847

## Nick browses Messages in Tech Topic using his login

GET

http://localhost:3000/piazzapost?topic=Tech

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.35.0	
<input checked="" type="checkbox"/>	Accept	/*/*	
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/>	Connection	keep-alive	
<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

body

Cookies

Headers (7)

Test Results

Status: 200 OK Time: 770 ms Size: 1.13 KB Save

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "_id": "6572479a66bbe631b071d838",
3    "post_title": "Tech details",
4    "post_topic": "Tech",
5    "likes_count": 0,
6    "dislikes_count": 0,
7    "comments": [],
8    "post_owner": "olga",
9    "post_time": "2023-12-07T22:35:00.000Z",
10   "post_owner_id": "65722e58477d8317b02e1ef8",
11   "post_status": "expired",
12   "post_date": "2023-12-07T22:30:50.844Z",
13   "__v": 0
14 }
15
```

```
    "_id": "657365e434b5b817a893d605",
    "post_title": "Tech details-Nick",
    "post_topic": "Tech",
    "likes_count": 0,
    "dislikes_count": 0,
    "comments": [],
    "post_owner": "Nick",
    "post_time": "2023-12-09T22:30:00.000Z",
    "post_owner_id": "65722e8866bbe631b071d832",
    "post_status": "live",
    "post_date": "2023-12-08T18:52:20.114Z",
    "__v": 0
  },
  {
    "_id": "6573668734b5b817a893d607",
    "post_title": "Tech details-Nick",
    "post_topic": "Tech",
    "likes_count": 0,
    "dislikes_count": 0,
    "comments": [],
    "post_owner": "Mary",
    "post_time": "2023-12-09T22:30:00.000Z",
    "post_owner_id": "65722eb6477d8317b02e1efb",
    "post_status": "live",
    "post_date": "2023-12-08T18:55:03.676Z",
    "__v": 0
  }
}
```

Deepa Karthick  
Candidate Num: Y106847

## Olga browses Messages in Tech Topic

GET

http://localhost:3000/piazzapost?topic=Tech

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Connection

Keep-alive

auth-token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...

Key

Value

Description

body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 117 ms

Size: 1.13 KB

Save

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "_id": "6572479a66bbe631b071d838",
3    "post_title": "Tech details",
4    "post_topic": "Tech",
5    "likes_count": 0,
6    "dislikes_count": 0,
7    "comments": [],
8    "post_owner": "olga",
9    "post_time": "2023-12-07T22:35:00.000Z",
10   "post_owner_id": "65722e58477d8317b02e1ef8",
11   "post_status": "expired",
12   "post_date": "2023-12-07T22:30:50.844Z",
13   "__v": 0
14 }
15
```

```
{
  "_id": "657365e434b5b817a893d605",
  "post_title": "Tech details-Nick",
  "post_topic": "Tech",
  "likes_count": 0,
  "dislikes_count": 0,
  "comments": [],
  "post_owner": "Nick",
  "post_time": "2023-12-09T22:30:00.000Z",
  "post_owner_id": "65722e886bbe631b071d832",
  "post_status": "live",
  "post_date": "2023-12-08T18:52:20.114Z",
  "__v": 0
},
{
  "_id": "6573668734b5b817a893d607",
  "post_title": "Tech details-Nick",
  "post_topic": "Tech",
  "likes_count": 2,
  "dislikes_count": 0,
  "comments": [],
  "post_owner": "Mary",
  "post_time": "2023-12-09T22:30:00.000Z",
  "post_owner_id": "65722eb6477d8317b02e1efb",
  "post_status": "live",
  "post_date": "2023-12-08T18:55:03.676Z",
  "__v": 0
}
```

Deepa Karthick  
Candidate Num: Y106847

## TC8. Nick and Olga “like” Mary’s post on the Tech topic.

Nick likes Mary’s post on the tech topic.

The image shows a Postman interface for a REST client. The top bar indicates a POST request to the URL `34.171.134.125/piazzapost/6573668734b5b817a893d607/like`. The 'Headers' tab is selected, showing the following headers:

Key	Value	Description
User-Agent	PostmanRuntime/7.35.0	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	

The 'Body' tab is selected, showing the response in JSON format. The status is 200 OK, with a time of 438 ms and a size of 543 B. The response body is:

```
1 {
2   "_id": "6573668734b5b817a893d607",
3   "post_title": "Tech details-Nick",
4   "post_topic": "Tech",
5   "likes_count": 1,
6   "dislikes_count": 0,
7   "comments": [],
8   "post_owner": "Mary",
9   "post_time": "2023-12-09T22:30:00.000Z",
10  "post_owner_id": "65722eb6477d8317b02e1efb",
11  "post_status": "live",
12  "post_date": "2023-12-08T18:55:03.676Z",
13  "__v": 0
14 }
```

Olga like’s Mary’s post on the tech topic

Deepa Karthick  
Candidate Num: Y106847

POST 34.171.134.125/piazzapost/6573668734b5b817a893d607/like

Params Authorization Headers (8) Body Pre-request Script Tests Settings

<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.35.0	
<input checked="" type="checkbox"/>	Accept	*/*	
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/>	Connection	keep-alive	
<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 352 ms Size: 543 B Save

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "6573668734b5b817a893d607",
3   "post_title": "Tech details-Nick",
4   "post_topic": "Tech",
5   "likes_count": 2,
6   "dislikes_count": 0,
7   "comments": [],
8   "post_owner": "Mary",
9   "post_time": "2023-12-09T22:30:00.000Z",
10  "post_owner_id": "65722eb6477d8317b02e1efb",
11  "post_status": "live",
12  "post_date": "2023-12-08T18:55:03.676Z",
13  "__v": 0
14 }
```

TC9.Nestor “likes” Nick’s post and “dislikes” Mary’s on the Tech topic

Nestor ‘likes’ Nick’s post

POST

34.171.134.125/piazzapost/657365e434b5b817a893d605/like

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

<input checked="" type="checkbox"/>	Host	<calculated when request is sent>	
<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.35.0	
<input checked="" type="checkbox"/>	Accept	*/*	
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/>	Connection	keep-alive	
<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 200 OK Time: 340 ms Size: 543 B Sav

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "_id": "657365e434b5b817a893d605",
3    "post_title": "Tech details-Nick",
4    "post_topic": "Tech",
5    "likes_count": 1,
6    "dislikes_count": 0,
7    "comments": [],
8    "post_owner": "Nick",
9    "post_time": "2023-12-09T22:30:00.000Z",
10   "post_owner_id": "65722e8866bbe631b071d832",
11   "post_status": "live",
12   "post_date": "2023-12-08T18:52:20.114Z",
13   "__v": 0
14 }
```

Nestor dislikes ‘Mary’s’ post

POST

34.171.134.125/piazzapost/6573668734b5b817a893d607/dislikes

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 200 OK Time: 418 ms Size: 543 B Sav

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "_id": "6573668734b5b817a893d607",
3    "post_title": "Tech details-Nick",
4    "post_topic": "Tech",
5    "likes_count": 2,
6    "dislikes_count": 1,
7    "comments": [],
8    "post_owner": "Mary",
9    "post_time": "2023-12-09T22:30:00.000Z",
10   "post_owner_id": "65722eb6477d8317b02e1efb",
11   "post_status": "live",
12   "post_date": "2023-12-08T18:55:03.676Z",
13   "__v": 0
14 }
```

TC10. Nick browses all the available posts on the Tech topic; at this stage, he can see the number of likes and dislikes for each post (Mary has two likes and one dislike, and Nick has one like). There are no comments made yet.

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

- Method:** GET
- URL:** http://localhost:3000/piazzapost?topic=Tech
- Headers:** 7 headers are listed, including 'auth-token' with a long alphanumeric value.
- Status:** 200 OK
- Time:** 120 ms
- Size:** 1.13 KB

The response body is a JSON object:

```
{
  "_id": "6572479a66bbe631b071d838",
  "post_title": "Tech details",
  "post_topic": "Tech",
  "likes_count": 0,
  "dislikes_count": 0,
  "comments": [],
  "post_owner": "olga",
  "post_time": "2023-12-07T22:35:00.000Z",
  "post_owner_id": "65722e58477d8317b02e1ef8",
  "post_status": "expired",
  "post_date": "2023-12-07T22:30:50.844Z",
  "__v": 0
},
```

Deepa Karthick  
Candidate Num: Y106847

GET http://localhost:3000/piazzapost?topic=Tech

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Key	Value	Description
auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmYWQiOiI2N...	

body Cookies Headers (7) Test Results Status: 200 OK Time: 120 ms Size: 1.13 KB Sav

Pretty Raw Preview Visualize JSON

```
10  {
11    {
12      "_id": "657365e434b5b817a893d605",
13      "post_title": "Tech details-Nick",
14      "post_topic": "Tech",
15      "likes_count": 1,
16      "dislikes_count": 0,
17      "comments": [],
18      "post_owner": "Nick",
19      "post_time": "2023-12-09T22:30:00.000Z",
20      "post_owner_id": "65722e8866bbe631b071d832",
21      "post_status": "live",
22      "post_date": "2023-12-08T18:52:20.114Z",
23      "__v": 0
24    },
25    {
26      "_id": "6573668734b5b817a893d607",
27      "post_title": "Tech details-Nick",
28      "post_topic": "Tech",
29      "likes_count": 2,
30      "dislikes_count": 1,
31      "comments": [],
32      "post_owner": "Mary",
33      "post_time": "2023-12-09T22:30:00.000Z",
34      "post_owner_id": "65722eb6477d8317b02e1efb",
35      "post_status": "live",
36      "post_date": "2023-12-08T18:55:03.676Z",
37      "__v": 0
38    }
39  ]
40 }
```

TC11. Mary likes her post on the Tech topic. This call should be unsuccessful; in Piazza, a post owner cannot like their messages.



The screenshot shows the Postman interface for a POST request. The URL bar contains the method 'POST' and the URL '34.171.134.125/piazzapost/6573668734b5b817a893d607/like'. The 'Headers' tab is selected, showing a list of headers with checkboxes for each. The 'Body' tab is also visible, showing a JSON response.

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Settings
<input checked="" type="checkbox"/>		Content-Length		(i) 0		
<input checked="" type="checkbox"/>		Host		(i) <calculated when request is sent>		
<input checked="" type="checkbox"/>		User-Agent		(i) PostmanRuntime/7.35.0		
<input checked="" type="checkbox"/>		Accept		(i) */*		
<input checked="" type="checkbox"/>		Accept-Encoding		(i) gzip, deflate, br		
<input checked="" type="checkbox"/>		Connection		(i) keep-alive		
<input checked="" type="checkbox"/>		auth-token		eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...		
		Key		Value		Description

Body Cookies Headers (7) Test Results

Status: 400 Bad Request Time: 439 ms Size: 288 B Save

Pretty Raw Preview Visualize JSON

```
1  
2  "message": "You cannot like your own post."  
3
```

TC12. Nick and Olga comment on Mary's post on the Tech topic in a round-robin fashion (one after the other, adding at least two comments each)

## Nick comment on Mary's post

POST

34.171.134.125/piazzapost/6573668734b5b817a893d607/comment

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

"comments": "This is Nick's second's comment on Mary post"

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 350 ms

Size: 631 B

Sav

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

"\_id": "6573668734b5b817a893d607",

"post\_title": "Tech details-Nick",

"post\_topic": "Tech",

"likes\_count": 2,

"dislikes\_count": 1,

"comments": [

"This is Nick first comment on Mary post",

"This is Nick's second's comment on Mary post"

],

"post\_owner": "Mary",

"post\_time": "2023-12-09T22:30:00.000Z",

"post\_owner\_id": "65722eb6477d8317b02e1efb",

"post\_status": "live",

"post\_date": "2023-12-08T18:55:03.676Z",

"\_v": 2

Deepa Karthick  
Candidate Num: Y106847

## Olga comment on Mary's post

The screenshot displays a REST client interface with a POST request to the URL `34.171.134.125/piazzapost/6573668734b5b817a893d607/comment`. The request body is a JSON object: `{ "comments": "This is Olga's first comment on Mary post" }`. The response status is 200 OK, with a time of 320 ms and a size of 722 B. The response body is a JSON object containing post details and a list of comments.

```
POST 34.171.134.125/piazzapost/6573668734b5b817a893d607/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

1
2 "comments": "This is Olga's first comment on Mary post"
3

Body Cookies Headers (7) Test Results
Status: 200 OK Time: 320 ms Size: 722 B Save

Pretty Raw Preview Visualize JSON
1
2 {
3   "_id": "6573668734b5b817a893d607",
4   "post_title": "Tech details-Nick",
5   "post_topic": "Tech",
6   "likes_count": 2,
7   "dislikes_count": 1,
8   "comments": [
9     "This is Nick first comment on Mary post",
10    "This is Nick's second's comment on Mary post",
11    "This is Olga's second comment on Mary's post",
12    "This is Olga's first comment on Mary post"
13  ],
14   "post_owner": "Mary",
15   "post_time": "2023-12-09T22:30:00.000Z",
16   "post_owner_id": "65722eb6477d8317b02e1efb",
17   "post_status": "live",
18   "post_date": "2023-12-08T18:55:03.676Z",
19   "__v": 4
}
```

TC13.Nick browses all the available posts in the Tech topic; at this stage, he can see the number of likes and dislikes of each post and the comments made.

GET

http://localhost:3000/piazzapost?topic=Tech

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 200 OKTime: 122 msSize: 1.31 KBSave

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "id": "6572479a66bbe631b071d838",
3    "post_title": "Tech details",
4    "post_topic": "Tech",
5    "likes_count": 0,
6    "dislikes_count": 0,
7    "comments": [],
8    "post_owner": "olga",
9    "post_time": "2023-12-07T22:35:00.000Z",
10   "post_owner_id": "65722e58477d8317b02e1ef8",
11   "post_status": "expired",
12   "post_date": "2023-12-07T22:30:50.844Z",
13   "__v": 0
14 },
15 {
16   }
```

GET

http://localhost:3000/piazzapost?topic=Tech

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 200 OKTime: 122 msSize: 1.31 KBSave

Pretty

Raw

Preview

Visualize

JSON

```
16  {
17    "id": "657365e434b5b817a893d605",
18    "post_title": "Tech details-Nick",
19    "post_topic": "Tech",
20    "likes_count": 1,
21    "dislikes_count": 0,
22    "comments": [],
23    "post_owner": "Nick",
24    "post_time": "2023-12-09T22:30:00.000Z",
25    "post_owner_id": "65722e8866bbe631b071d832",
26    "post_status": "live",
27    "post_date": "2023-12-08T18:52:20.114Z",
28    "__v": 0
29  },
30  {
31    "id": "6573668734b5b817a893d607",
32    "post_title": "Tech details-Nick",
33    "post_topic": "Tech",
34    "likes_count": 2,
35    "dislikes_count": 1,
36    "comments": [
37      "This is Nick first comment on Mary post",
38      "This is Nick's second's comment on Mary post",
39      "This is Olga's second comment on Mary's post",
40      "This is Olga's first comment on Mary post"
41    ],
42    "post_owner": "Mary",
43    "post_time": "2023-12-08T20:00:00.000Z",
44    "post_owner_id": "65722e58477d8317b02e1ef8",
45    "post_status": "live",
46    "post_date": "2023-12-08T18:52:20.114Z",
47    "__v": 0
48  }
49 ]
```

Deepa Karthick  
Candidate Num: Y106847

TC14. Nestor posts a message in the Health topic with an expiration time using her token

POST 34.171.134.125/piazzapost

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "post_title": "Health details-Nestor",
3   "post_topic": "Health",
4   "post_owner": "Nestor",
5   "post_time": "2023-12-08T20:10:00.000Z"
6 }
```

body Cookies Headers (7) Test Results Status: 200 OK Time: 363 ms Size: 551 B Save

Pretty Raw Preview Visualize JSON

```
1 {
2   "post_title": "Health details-Nestor",
3   "post_topic": "Health",
4   "likes_count": 0,
5   "dislikes_count": 0,
6   "comments": [],
7   "post_owner": "Nestor",
8   "post_time": "2023-12-08T20:10:00.000Z",
9   "post_owner_id": "65722ed466bbe631b071d835",
10  "post_status": "live",
11  "_id": "6573765ffa095d6cfca60151",
12  "post_date": "2023-12-08T20:02:39.120Z",
13  "__v": 0
14 }
```

TC15. Mary browses all the available posts on the health topic; at this stage, she can see only Nestor's post.

Deepa Karthick  
Candidate Num: Y106847

GET http://localhost:3000/piazzapost?topic=Health

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Key	Value	Description
auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	

Body Cookies Headers (7) Test Results Status: 200 OK Time: 117 ms Size: 553 B Sav

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "6573765ffa095d6cfca60151",
3   "post_title": "Health details-Nestor",
4   "post_topic": "Health",
5   "likes_count": 0,
6   "dislikes_count": 0,
7   "comments": [],
8   "post_owner": "Nestor",
9   "post_time": "2023-12-08T20:10:00.000Z",
10  "post_owner_id": "65722ed466bbe631b071d835",
11  "post_status": "live",
12  "post_date": "2023-12-08T20:02:39.120Z",
13  "__v": 0
14 }
15
16 }
```

TC16. Mary posts a comment in Nestor's message on the Health topic.

POST 34.171.134.125/piazzapost/6573765ffa095d6cfca60151/comment

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "comments": "This is Mary's first comment on Nestor health post"
3 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 321 ms Size: 603 B Save

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "6573765ffa095d6cfca60151",
3   "post_title": "Health details-Nestor",
4   "post_topic": "Health",
5   "likes_count": 0,
6   "dislikes_count": 0,
7   "comments": [
8     "This is Mary's first comment on Nestor health post"
9   ],
10  "post_owner": "Nestor",
11  "post_time": "2023-12-08T20:10:00.000Z",
12  "post_owner_id": "65722ed466bbe631b071d835",
13  "post_status": "live",
14  "post_date": "2023-12-08T20:02:39.120Z",
15  "__v": 1
16 }
```

TC17. Mary dislikes Nestor’s message on the health topic after the end of post-expiration time. This should fail.

POST

34.171.134.125/piazzapost/6573765ffa095d6cfca60151/dislikes

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.35.0	
<input checked="" type="checkbox"/>	Accept	*/*	
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/>	Connection	keep-alive	
<input checked="" type="checkbox"/>	auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results

Status: 403 ForbiddenTime: 332 msSize: 301 B

Pretty

Raw

Preview

Visualize

JSON

1

2

3

"message": "This post has expired and cannot be disliked."

Deepa Karthick  
Candidate Num: Y106847

TC18. Nestor browses all the messages on the Health topic. There should be only one post (his own) with one comment (Mary's).

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

- Method:** GET
- URL:** http://localhost:3000/piazzapost?topic=Health
- Headers (7):** Includes an 'auth-token' header with a long alphanumeric value.
- Status:** 200 OK
- Time:** 112 ms
- Size:** 608 B

The response body is a JSON object in 'Pretty' format:

```
1 {
2   "_id": "6573765ffa095d6cfca60151",
3   "post_title": "Health details-Nestor",
4   "post_topic": "Health",
5   "likes_count": 0,
6   "dislikes_count": 0,
7   "comments": [
8     "This is Mary's first comment on Nestor health post"
9   ],
10  "post_owner": "Nestor",
11  "post_time": "2023-12-08T20:10:00.000Z",
12  "post_owner_id": "65722ed466bbe631b071d835",
13  "post_status": "expired",
14  "post_date": "2023-12-08T20:02:39.120Z",
15  "__v": 1
16 }
17
18 }
```



Deepa Karthick  
Candidate Num: Y106847

TC19. Nick browses all the expired messages on the Sports topic.  
These should be empty

GET http://localhost:3000/piazzapost?topic=Sport&status=expired

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Key	Value	Description
auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...	

body Cookies Headers (7) Test Results Status: 404 Not Found Time: 110 ms Size: 271 B Sav

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "No posts found."
3 }
```

Deepa Karthick  
Candidate Num: Y106847

TC20. Nestor queries for an active post with the highest interest (maximum number of likes and dislikes) in the Tech topic. This should be Mary's post.

GET

34.171.134.125/piazzapost/trending

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

☒

auth-token

Key

Value

Description

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 372 ms

Size: 1.75 KB

Sav

Pretty

Raw

Preview

Visualize

JSON

```
1  [
2    {
3      "_id": "6573668734b5b817a893d607",
4      "post_title": "Tech details-Nick",
5      "post_topic": "Tech",
6      "likes_count": 2,
7      "dislikes_count": 1,
8      "comments": [
9        "This is Nick first comment on Mary post",
10       "This is Nick's second's comment on Mary post",
11       "This is Olga's second comment on Mary's post",
12       "This is Olga's first comment on Mary post"
13     ],
14     "post_owner": "Mary",
15     "post_time": "2023-12-09T22:30:00.000Z",
16     "post_owner_id": "65722eb6477d8317b02e1efb",
17     "post_status": "live",
18     "post_date": "2023-12-08T18:55:03.676Z",
19     "_v": 4,
20     "likesAndDislikesSum": 3
21   },
```

Deepa Karthick  
Candidate Num: Y106847

GET 34.171.134.125/piazzapost/trending

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

auth-token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2N...

Key Value Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 372 ms Size: 1.75 KB Sav

Pretty Raw Preview Visualize JSON ↗

```
23  {
24    "_id": "657365e434b5b817a893d605",
25    "post_title": "Tech details-Nick",
26    "post_topic": "Tech",
27    "likes_count": 1,
28    "dislikes_count": 0,
29    "comments": [],
30    "post_owner": "Nick",
31    "post_time": "2023-12-09T22:30:00.000Z",
32    "post_owner_id": "65722e8866bbe631b071d832",
33    "post_status": "live",
34    "post_date": "2023-12-08T18:52:20.114Z",
35    "__v": 0,
36    "likesAndDislikesSum": 1
37  },
38  {
39    "_id": "6572479a66bbe631b071d838",
40    "post_title": "Tech details",
41    "post_topic": "Tech",
42    "likes_count": 0,
43    "dislikes_count": 0,
44    "comments": [],
45    "post_owner": "olga",
46    "post_time": "2023-12-07T22:35:00.000Z",
47    "post_owner_id": "65722e58477d8317b02e1ef8",
48    "post_status": "live",
49    "post_date": "2023-12-07T22:30:50.844Z",
50    "__v": 0,
51    "likesAndDislikesSum": 0
52  }
53 }
```

## Deploy your Piazza project into a VM using Docker

### 1.Code moved to Github

1. Git Init
2. git remote add origin <https://github.com/deepakarthick82/Cloudcomputing.git>
3. git add .
4. git commit -m "Pushing my post"
5. git push -f origin master
6. git push
7. Git hub repository [link](#)

### 2.Github repo

The following screenshot shows the git commands used to upload the project into github repo.

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS

[master 4aa1e35] Pushing my post5
1 file changed, 3 insertions(+), 2 deletions(-)

karth@Diya MINGW64 ~/OneDrive/Deepa/Birbeck/cloud computing/coursework_2nd/coursework_3 (master)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 413 bytes | 413.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/deepakarthick82/Cloudcomputing.git
```

The screenshot shows a GitHub repository named 'Cloudcomputing' by user 'deepakarthick82'. The repository is public and has 2 branches and 0 tags. The current branch is 'master'. A message indicates the branch is 6 commits ahead and 1 commit behind main. A table lists the files in the repository, including folders like 'models', 'node\_modules', 'routes', and 'validations', and files like '.env', 'app.js', 'package-lock.json', 'package.json', and 'verifyToken.js'. A button 'Add a README' is visible at the bottom.

File/Folder	Commit Message	Time
models	Pushing my post3	2 days ago
node_modules	Pushing my post	2 days ago
routes	Pushing my post5	yesterday
validations	Pushing my post	2 days ago
.env	Pushing my post	2 days ago
app.js	Pushing my post3	2 days ago
package-lock.json	Pushing my post	2 days ago
package.json	Pushing my post	2 days ago
verifyToken.js	Pushing my post	2 days ago

Code pushed to github

3.Created a VM in GCP virtual machine called docker-vm is created on google cloud.

The screenshot shows the 'VM instances' page in the Google Cloud Platform. It lists four VM instances: 'cloud-cls1-vm', 'docker-demo', 'gke-plazza-newpost1-clus-default-pool-b9db7730-d8yj', and 'gke-plazza-newpost1-clus-default-pool-'. The 'docker-demo' instance is highlighted.

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	cloud-cls1-vm	us-central1-a			10.128.0.2 (nic0)		SSH
<input checked="" type="checkbox"/>	docker-demo	us-central1-a			10.128.0.3 (nic0)	34.171.128.189 (nic0)	SSH
<input checked="" type="checkbox"/>	gke-plazza-newpost1-clus-default-pool-b9db7730-d8yj	us-central1-c		gke-plazza-newpost1-clus-default-	10.128.0.19 (nic0)	34.69.14.163 (nic0)	SSH
<input checked="" type="checkbox"/>	gke-plazza-newpost1-clus-default-pool-	us-central1-c		gke-plazza-newpost1-clus-default-	10.128.0.20 (nic0)	34.173.174.98 (nic0)	SSH

Created VM named docker-demo in GCP

4.Command to push github code to VM:

```
git clone git@github.com:deepakarthick82/Cloudcomputing.git
```

5.Code is pushed from Git to the VM and the code is checked out.

```
docker-user@docker-demo:~/Cloudcomputing$ ls
'Cloud Computing Coursework 2023 Piazza.pdf'  app.js  node_modules  package.json  validations
Dockerfile                                    models  package-lock.json  routes        verifyToken.js
docker-user@docker-demo:~/Cloudcomputing$
```

6.Create a docker file named dockerfile using *pico* command

```
docker-user@docker-demo:~$ cat Dockerfile
FROM alpine
RUN apk add --update nodejs npm
COPY . /src
WORKDIR /src
EXPOSE 3000
ENTRYPOINT ["node", "./app.js"]
docker-user@docker-demo:~$ ^C
```

Contents of the Dockerfile

7.Docker image created using the command docker image build -t cloudcomputing-image .


8.Create a container and run it docker container run -d --name deepa-web --publish 80:3000 cloudcomputing-image

9.View created Docker images

```
docker-user@docker-demo:~$ docker images
REPOSITORY              TAG          IMAGE ID          CREATED          SIZE
deepakarthick82/ccpiazzapost  1           b94a7d9403f4     29 hours ago    94.3MB
mini-python3-image       latest      e6ccdb57cb85     5 days ago      253MB
ubuntu                  latest      b6548eacb063     7 days ago      77.8MB
alpine                  latest      b541f2080109     7 days ago      7.34MB
python                  latest      afb69f3af77f     6 weeks ago     1.02GB
```

10.Pushed the code to docker hub: docker push deepakarthick82/ccpiazzapost:1

Explore / deepakarthick82/ccpiazzapost



## deepakarthick82/ccpiazzapost ☆

By [deepakarthick82](#) • Updated a day ago

Manage Repository

Pulls 4

Overview **Tags**

Sort by Newest Filter Tags

TAG	DIGEST	OS/ARCH	LAST PULL	COMPRESSED SIZE
1	5395b030a6c8	linux/amd64	13 hours ago	34.06 MB

docker pull deepakarthick82/ccpi...

## Deploy Piazza application in Kubernetes

### 1.Create Kubernetes Cluster

OVERVIEW    OBSERVABILITY    COST OPTIMISATION							
Filter Enter property name or value							
<input type="checkbox"/> Status	Name ↑	Location	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
<input checked="" type="checkbox"/>	<a href="#">piazza-newpost1-cluster</a>	us-central1-c	3	6	12 GB	—	⋮

Piazza-newpost1-cluster is created

### 2.Create deployment.yaml using *pico* command in the cloud shell

```
app: piazza-post
template:
  metadata:
    labels:
      app: piazza-post
  spec:
    containers:
      - name: piazza-post
        image: deepakarthick82/ccpiazzapost:1
        imagePullPolicy: Always
        ports:
          - containerPort: 3000
deepakarthick2022@cloudshell:~ (cloudclass1-401818) $
```

### 3.Apply the deployment file using command *kubectl apply -f deployment.yaml*

PODS details:

```
deepakarthick2022@cloudshell:~ (cloudclass1-401818) $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
piazza-post-deployment-5d4549756d-468x4  1/1     Running   0           23h
piazza-post-deployment-5d4549756d-78t2l  1/1     Running   0           23h
piazza-post-deployment-5d4549756d-7w57r  1/1     Running   0           23h
piazza-post-deployment-5d4549756d-djmzh  1/1     Running   0           23h
piazza-post-deployment-5d4549756d-ljgj5  1/1     Running   0           23h
deepakarthick2022@cloudshell:~ (cloudclass1-401818) $
```

### 4.Create service.yaml using *pico* command and apply it using *kubectl apply -f service.yaml*

```
labels:
  app: piazza-post-service
spec:
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app: piazza-post
  sessionAffinity: None
deepakarthick2022@cloudshell:~ (cloudclass1-401818)$
```

## 5. Get the API end points under virtual machine IP address

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cloudclass1-401818.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
deepakarthick2022@cloudshell:~ (cloudclass1-401818)$ kubectl get services
NAME                                TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes                         ClusterIP      10.32.0.1     <none>         443/TCP          2d
piazza-post-service               LoadBalancer  10.32.6.130   34.171.134.125 80:30044/TCP     29h
deepakarthick2022@cloudshell:~ (cloudclass1-401818)$
```

Test cases are triggered in Post man using the External IP

The screenshot shows the Postman interface for a POST request to the endpoint `34.171.134.125/user/register`. The request body is a JSON object with the following fields:

```
{
  "username": "Nestor",
  "email": "Nestor@piazza.com",
  "password": "Nestor"
}
```

The response is a JSON object with the following fields:

```
{
  "username": "Nestor",
  "email": "Nestor@piazza.com",
  "password": "$2a$05$gdjTVoPLLSQd/gdLrW5AU0m4du5j0vA56Ix3fN5Mi49yPVgvVILLm",
  "_id": "65722ed466bbe631b071d835",
  "date": "2023-12-07T20:45:08.964Z",
  "__v": 0
}
```

The status of the request is 200 OK.

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL '34.171.134.125/piazzapost' is entered. Below this, tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible. The 'Body' tab is active, showing a raw JSON request body with the following content:

```
1 {
2   "post_title": "Health details-Nestor",
3   "post_topic": "Health",
4   "post_owner": "Nestor",
5   "post_time": "2023-12-08T20:10:00.000Z"
6 }
```

Below the request body, tabs for 'Body', 'Cookies', 'Headers (7)', and 'Test Results' are shown. The 'Body' tab is active, displaying the response in 'Pretty' format. The status bar indicates 'Status: 200 OK', 'Time: 363 ms', and 'Size: 551 B'. The response body is a JSON object with the following content:

```
1 {
2   "post_title": "Health details-Nestor",
3   "post_topic": "Health",
4   "likes_count": 0,
5   "dislikes_count": 0,
6   "comments": [],
7   "post_owner": "Nestor",
8   "post_time": "2023-12-08T20:10:00.000Z",
9   "post_owner_id": "65722ed466bbe631b071d835",
10  "post_status": "live",
11  "_id": "6573765ffa095d6cfca60151",
12  "post_date": "2023-12-08T20:02:39.120Z",
13  "__v": 0
14 }
```

## References

Chris, K. (2021) Rest API best practices – rest endpoint design examples, freeCodeCamp.org. freeCodeCamp.org. Available at: <https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/> (Accessed: December 11, 2022).

Copes, F. (2022) JWT authentication: Best practices and when to use it, LogRocket Blog. Available at: <https://blog.logrocket.com/jwt-authentication-best-practices/> (Accessed: December 11, 2022).

Kaelin, M.W. et al. (2022) How to create a virtual machine in Google Cloud Platform, TechRepublic. Available at: <https://www.techrepublic.com/article/how-to-create-a-virtual-machine-in-google-cloud-platform/> (Accessed: December 11, 2022).

MongoDB tutorial: What it is and features - javatpoint (no date) [www.javatpoint.com](http://www.javatpoint.com). Available at: <https://www.javatpoint.com/mongodb-tutorial> (Accessed: December 11, 2022).

Node.js (no date) Node.js. Available at: <https://nodejs.org/en/> (Accessed: December 11, 2022).