DECEMBER 11, 2022

DEVELOPMENT OF A CLOUD SOFTWARE AS A SERVICE

MATTHEW BALMFORTH
BIRKBECK, UNIVERSITY OF LONDON
Student ID: 13806785

Post-It

Introduction

In this report I document the creation of Post-It, a Cloud Software as a Service API that enables users to post notes called *post-its* into a space called *the pile*. Users can interact with these post-its via comments or likes, with the most liked post-its surfacing to the top of the pile.

Post-It is written in JavaScript, with the code implemented in Node.js. The data is captured in MongoDB, an easy-to-use NoSQL database program. Post-It currently only exists as a backend, with the API development validated in Postman.

Phase A: Setting up Post-It

Installation of necessary libraries

In the table below I have outlined the different libraries required for Post-It to function. *Express* and *nodemon* are libraries that facilitate rapid app development, while *mongoose* permits facile manipulation of data pulled from MongoDB. Since user registration and login is a prerequisite to access various endpoints including the post-it pile, various libraries are useful to have to allow a user to have autonomy over their password, and to have the app recognise the user as authentic. The two libraries *bcryptjs* and *jsonwebtoken* address these two points. The library *bcryptjs* enables password hashing to prevent a user's password from being stored directly on the MongoDB server. The module *jsonwebtoken* issues a token to a logged in user that is then used to verify the user every time the user interacts with the server, for instance posting, liking or commenting on the post-it pile. The joi package enforces validations on data inputted by a user, thus preventing data from being inserted that doesn't meet certain requirements e.g. an email not containing an '@' symbol, or a post that exceeds a specified length. It additionally enforces required data, thus preventing data from being inserted lacking critical fields.

Table 1. Packages used in this project in Node.js together with a short description of their use

Package	**Use in Node.js**
express	A web application framework to enable app development[1]
nodemon	Enables restart of the app immediately after file changes are made to allow automatic update while developing[2]
mongoose	An object relational mapping tool to enable data manipulation from the MongoDB database[3]
dotenv	Loads environment variables into process.env from a .env file to allow use of passwords and tokens in the app in a safe manner[4]
body-parser	Middleware to parse user request bodies e.g. into JSON format[5]
joi	A schema description language to enforce schema validations onto data[6]
bcryptjs	A library to enable password hashing and decrypting[7]

jsonwebtoken	Enables use of JSON Web Tokens to authenticate app users for access-only API endpoints[8]

In order to install these packages, I initialised the Node Package Manager (npm) and then used *npm install* to install the packages. The commands for these are outlined in the *commands.md* file. Following their installation, I modified the *"scripts"* object in the subsequently compiled *package.json* file to run the file *app.js* using the *nodemon* package when entering the command *npm start*.

I created a *.env* file that contained a DB_CONNECTOR that links the mongoDB under my username and project folder. I additionally added a TOKEN_SECRET that is required for the token verification step. The *.env* file is not included in my scripts, as this contains my mongoDB username and password!

Once the packages were installed and the scripts written, I validated the program using Postman to access a series of API endpoints outlined in the scripts.

Provided below are the API endpoints.

```
21  app.listen(3001, ()=>{
22      console.log('server is running')
23  })
```

```
14  app.use('/api/pile', pileRoute)
15  app.use('/api/user', authRoute)
```

Figure 1. API endpoints for the user and pile routes. User endpoints will be available under "localhost:3001/api/user" and pile endpoint for accessing all post-its will be available under "localhost:3001/api/pile"

```
10  router.post('/register', async(req,res)=>{ ···
40  })
41
42  router.post('/login', async(req,res)=>{ ···
65  })
```

Figure 2. API endpoints for registering and login. As these are properties of user, they are available under "localhost:3001/api/user/register" and "localhost:3001/api/user/login" respectively

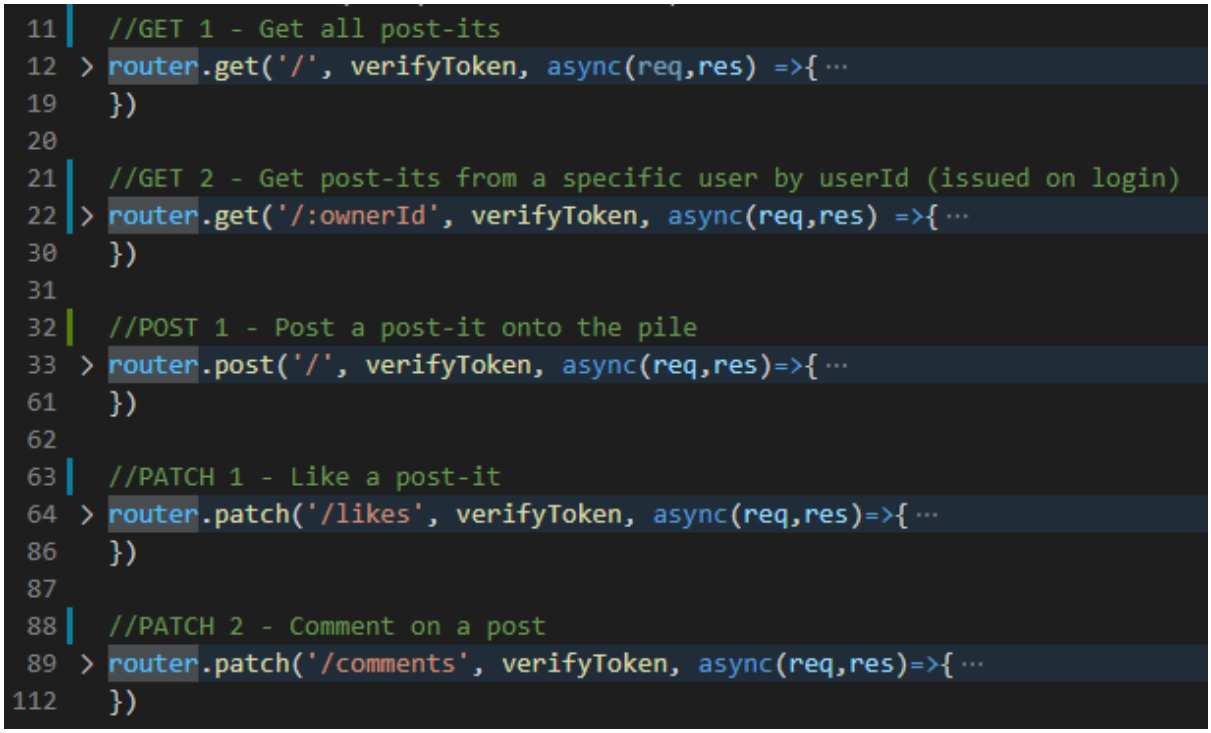

Figure 3. API endpoints for CRUD operations on the pile. Accessing "localhost:3001/api/pile" will return all post-its. Post-its by owner ID can be searched for under "localhost:3001/api/pile/:ownerId". All additional operations can be carried out on pile as outlined in the screenshot above.

Accessing these endpoints via Postman enabled me to interact with the application, using the CRUD operations GET, POST and PATCH.

The scripts are contained with the script folder, which in turn contains three subfolders, */models*, */routes* and */validations*. An additional four files are contained in the top-level directory of the scripts folder. The files and structure of the folders are detailed in the below table and figure.

Table 2. Description for all the different files necessary to run Post-It. Table includes file name and a short description of the contents of the file.

File name	Description
app.js	The script that is initialised to run the program. Connects to mongoDB via DB_CONNECTOR from .env file. Enables program output to be accessed via localhost. Creates API endpoints for accessing User and Post-It pile.
verifyToken.js	Contains the function *auth* which takes the TOKEN_SECRET from .env and uses the library jsonwebtoken to verify tokens provided by the user to access and interact with the Post-It pile.
messages.js	Contains a list of messages that are provided to the user in case of error. Created to make code tidier and less verbose.
commands.md	Contains a short list of commands used to initialise node package manager and install dependencies.
.env	Enabled by the dotenv package, .env contains the connector to access mongoDB as well as the token secret required to validate user-provided tokens to reach API endpoints. This file is not included in the submission.
Postit.js	The model schema for post-its. Describes first the comment schema, which is then nested within the post-it schema.

User.js	The model schema for users.
postits.js	Defines the CRUD operations for post-its. Operations require certain validations to be met to prevent e.g. users liking their own posts.
user-auth.js	Defines the operations for users, including registering and logging into the service. Validations must be met to prevent illegal operations.
validation.js	Details the validations for the register, login, post-it, and comment requirements. The joi package is used extensively here to facilitate basic requirements e.g. for emails.

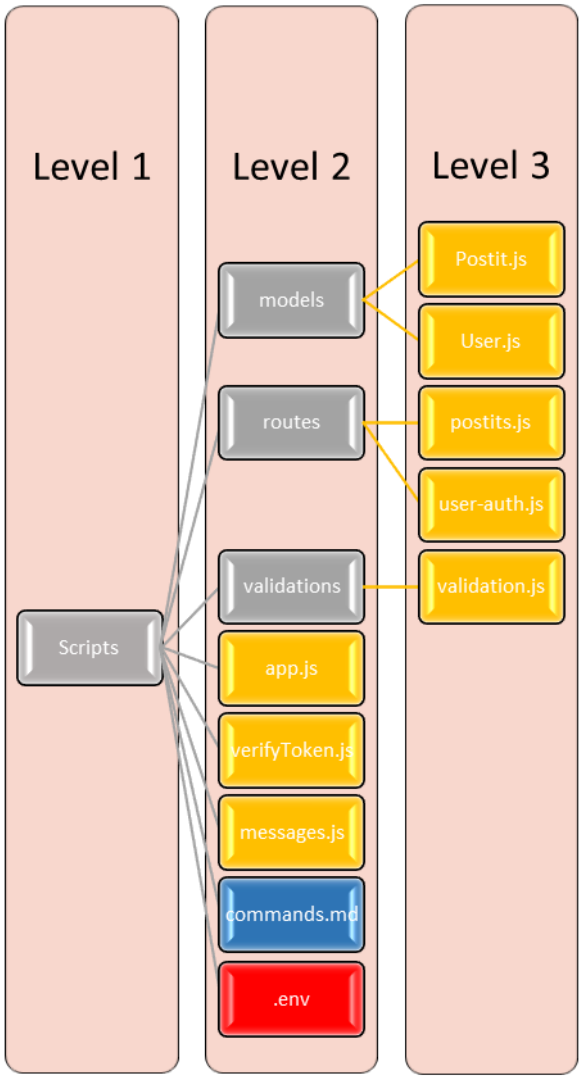

Figure 4. Structure of the files and folders for implementation of Post-It. The scripts are contained within three directories. Level 1 is the home directory. All folders are shown in grey, with JavaScript files shown in yellow, the md file shown in blue, and the .env file shown in red. Note that the .env file is not included in the submitted scripts.

Phase B: Enforcing authentication/verification functionalities

User management and Json Web Token (JWT) functionality have been included in Post-It to enable authorised users to store data in mongoDB via a REST API. Additionally, all CRUD operations include a token verification step to authenticate users each time they carry out

any operation on the post-it pile, such as adding, commenting, or liking a post-it. In order to view the pile, users have to be verified via a token that is provided at the user login stage. The JWT functionality and subsequent token verification step are defined in verifyToken.js.

In addition to the token verification, user input is subjected to validations to ensure that users do not comment or like their own posts, the post-it title doesn't already exist, and that the post-it ID used for liking post-its is actually present in the database.

Phase C: Development of the Post-It RESTful API

Please see the scripts folder for technical details. All specifications outlined in Phase C have been met.

Table 3. API endpoints used in Post-It, together with possible CRUD operations for each endpoint.

API endpoints	**CRUD operations**
localhost:3001/api/user/register	POST
localhost:3001/api/user/login	POST
localhost:3001/api/pile	GET, POST
localhost:3001/api/pile/:*ownerId*	GET
localhost:3001/api/pile/comments	PATCH
localhost:3001/api/pile/likes	PATCH

How to use Post-It

Register and login

1. Run the commands in *commands.md*
2. Open the *package.json* file and modify the value of the *scripts* key as follows:

```
"scripts": {
  "start": "nodemon app.js"
}
```

3. Save the file.
4. Run the command *npm start*
5. Open Postman and register a user using a POST operation on localhost:3001/api/user/register. Registering a user requires three pieces of information:
    a. username
    b. email

c. password

Registering should trigger a host message to state that you have successfully registered. The password is stored in mongoDB in a hashed representation using the *bcryptjs* package.

6. Then login using a POST operation on localhost:3001/api/user/login. Logging in requires only the email and password to be entered. Once logged in, a host message is displayed, along with an authorisation token (auth-token). The user should copy this auth-token for user authentication.

Accessing the pile and adding post-its

7. To view the pile of post-its, enter the auth-token into the key-value field of Headers on Postman, and change the API endpoint to GET localhost:3001/api/pile. If the auth-token is valid, then the post-its will display.
8. To view all the comments of a particular user, To view all the comments of a particular user, change the API endpoint to localhost:3001/api/pile/:*ownerId*, where *ownerId* is the "_id" of the user when they registered.
9. To add post-its to the pile, change the API endpoint to POST localhost:3001/api/pile. A post-it requires three pieces of information:
    a. owner (username registered)
    b. title
    c. content

    The auth-token also needs to be provided in the key-value of the Headers section. Once a post-it has been successfully added to the pile, a host-message will display, along with the added post-it.

Commenting and liking post-its

10. Liking and commenting are PATCH operations. They require changing the API endpoint to localhost:3001/api/pile/comments for comments or localhost:3001/api/pile/likes for likes.
11. Commenting requires three pieces of information:
    a. postitId ("_id" of the post-it to comment on)
    b. owner (username of commenter)
    c. content (content of the comment)

    Once posted, comments appear under the posts along with a host message confirming the successful operation.
12. Liking a post-it requires a similar operation to commenting, except only the postitId and owner are required fields for liking. As with the comment, a host message is returned for proof that a post-it has been liked.

13. Users can like or comment as many times as they wish, although a post-it owner cannot like or comment on their own post-its.

Phase D: Development of the Post-It testing cases

Olga, Nick and Mary register on Post-It.

A screenshot is shown in the figure below showing the registering operation as Olga registers her information on Post-It. Olga receives a host message and the details of her registration.

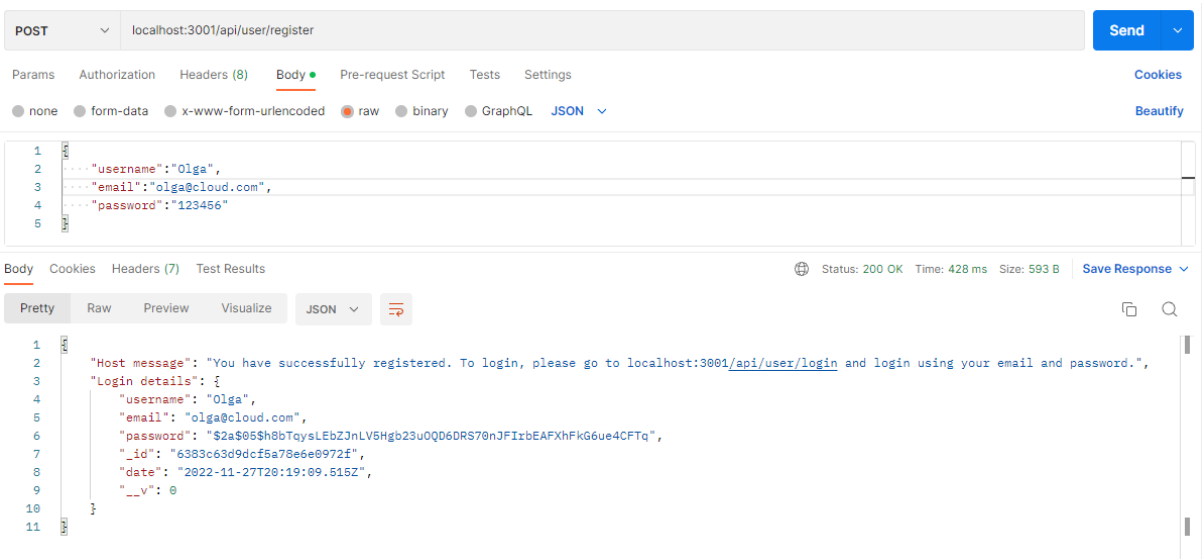

Figure 5. Example showing Olga registering on Post-It via Postman. Olga receives a register confirmation message from the host with instructions on how to login.

Nick and Mary also register, and then Nick tries to login to Post-It. He receives a welcome message as he successfully logs in. Mary and Olga then also login. Olga tries to view the pile of post-its, but is denied, as she does not use a token and therefore is not authorised. She receives a message detailing where to retrieve the token and how to use it.

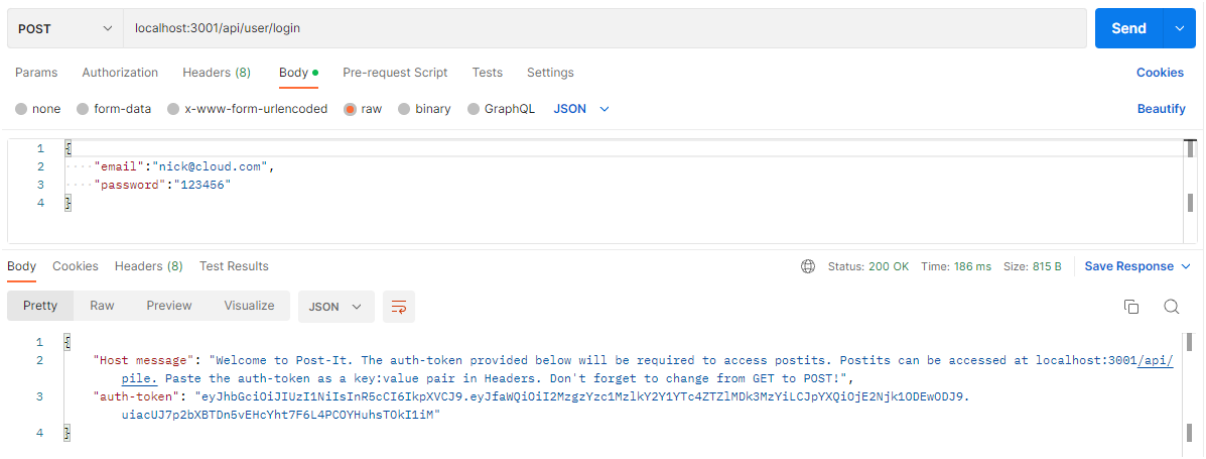

Figure 6. Example showing Nick logging on to Post-It via Postman. Nick receives a welcome message from the host with instructions on how to reach the pile. Nick also receives an auth-token to provide him with verification access to the pile.

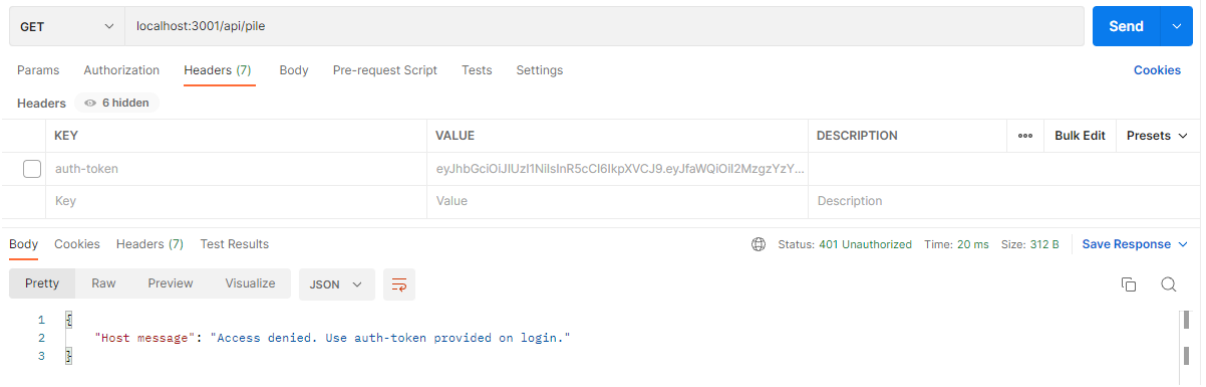

Figure 7. Olga tries to access the pile without using an auth-token and receives an access denied message from the host.

Olga then retrieves her token and is authorised to view the pile. She decides to add a post-it to the empty pile using her token and is successful. She only posts her name, the post-it tile and the content of the post-it. She is returned the full details of the post-it including date, likes (0), comments (none) and post-it unique identifier. She also receives a host message confirming that the post-it has been added to the pile (Figure 8).

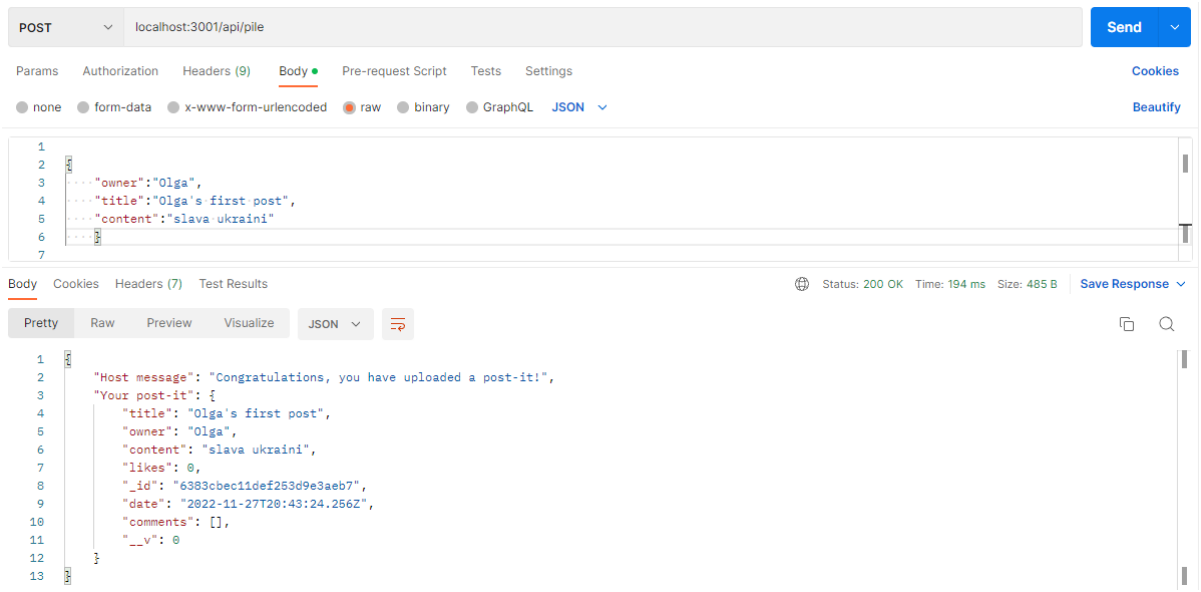

Figure 8. Olga successfully posts a post-it to the pile using her token

Nick and Mary also successfully post to the pile user their authorisation tokens. Nick then browses the pile and sees the post-its in chronological order, with the first post-it at the top of the pile (Figure 9). Mary also browses the pile and sees the same view.

Nick and Olga decide then to comment on Mary's post-it. Nick posts first, followed by Olga. They comment by using Mary's post-it ID, together with their comment content and their name (owner). They both receive confirmations of their comments having been posted, and see Mary's post-it followed by their comments, shown in chronological order (Figure 10). Mary sees these comments in chronological order (Figure 11) and tries to comment back on her post-it but is unsuccessful, as post-it owners cannot comment on their own post-it (Figure 12)!

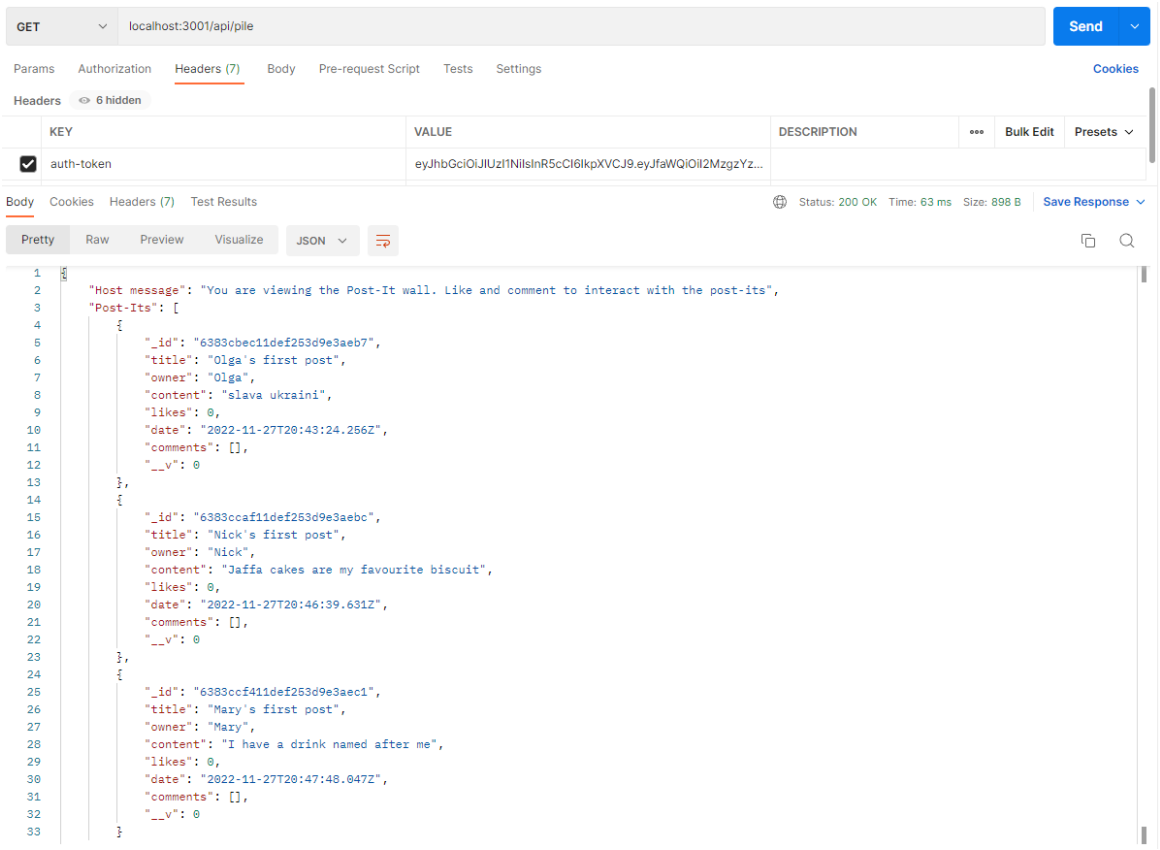

Figure 9. Nick browses the pile in chronological order, seeing the first post-its on the pile first.

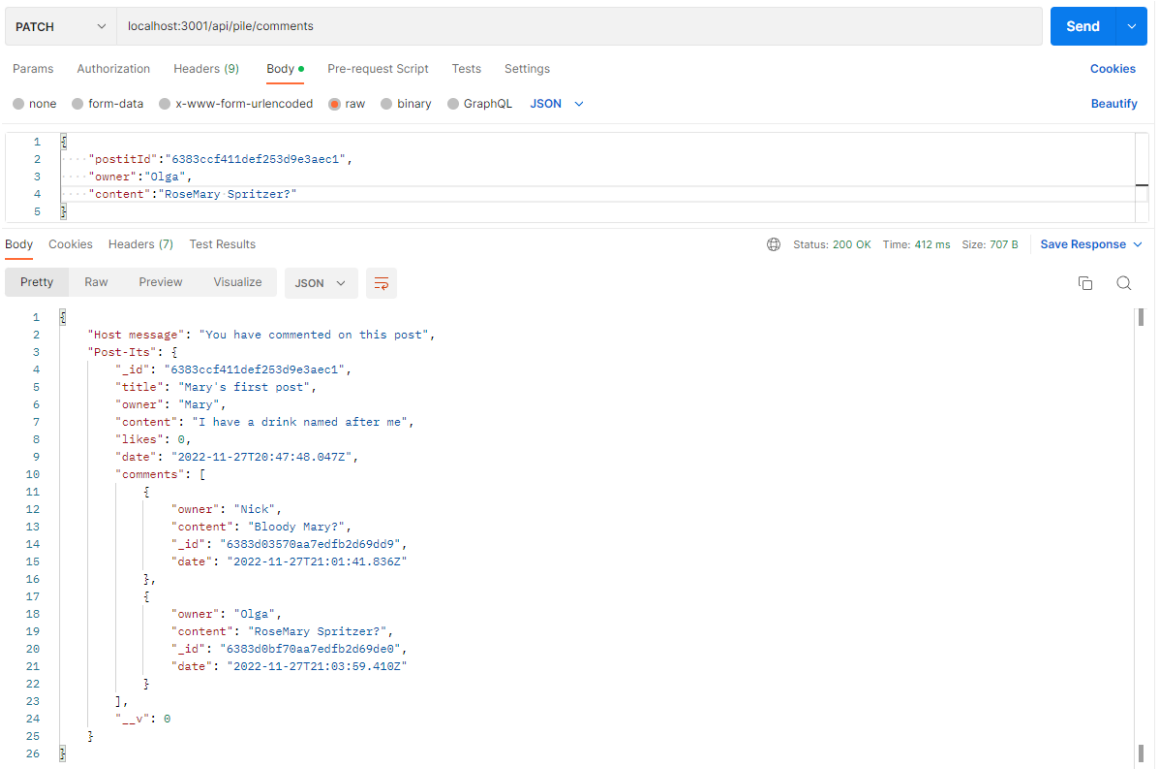

Figure 10. Nick and Olga comment on Mary's post-it in round-robin fashion. Olga's comment is shown in this example.

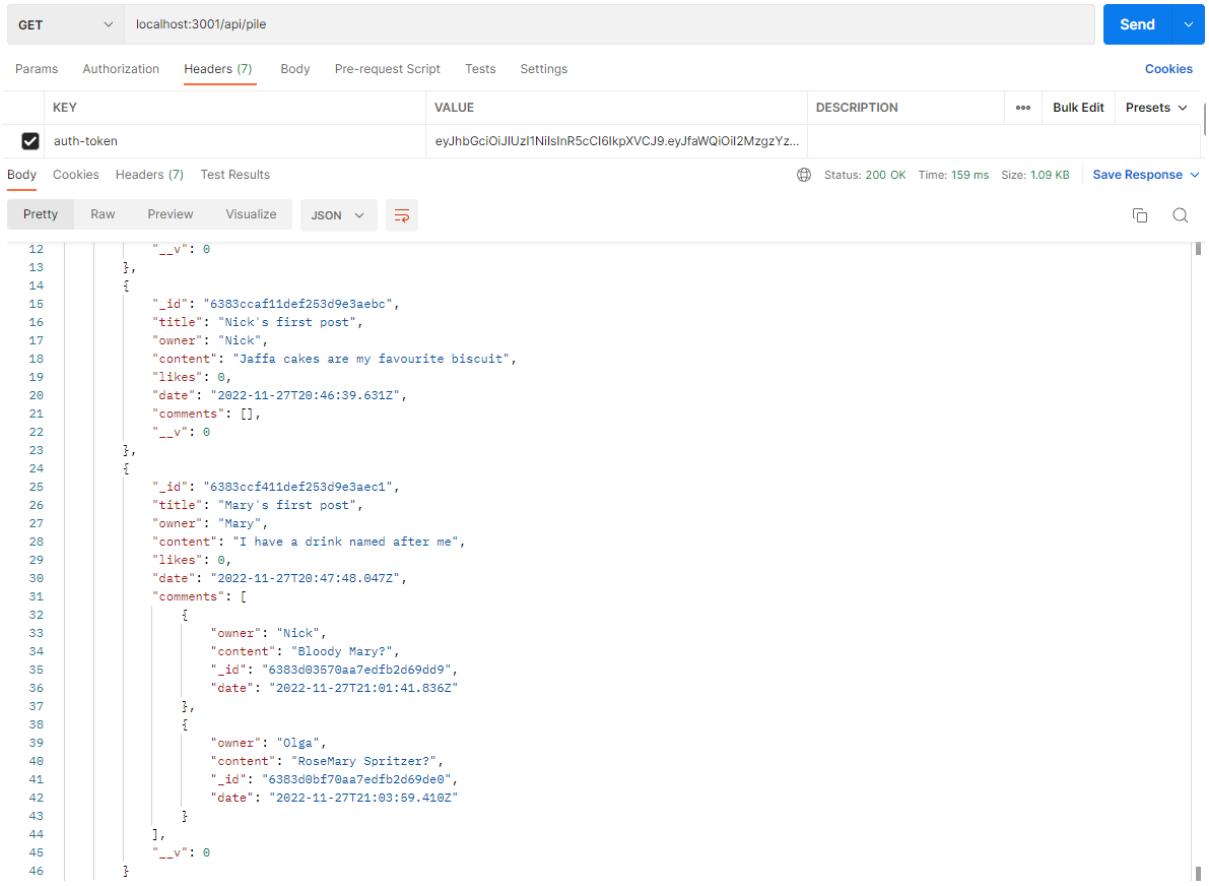

Figure 11. Mary can see the comments posted onto her post-it and can view the post-its in chronological order.

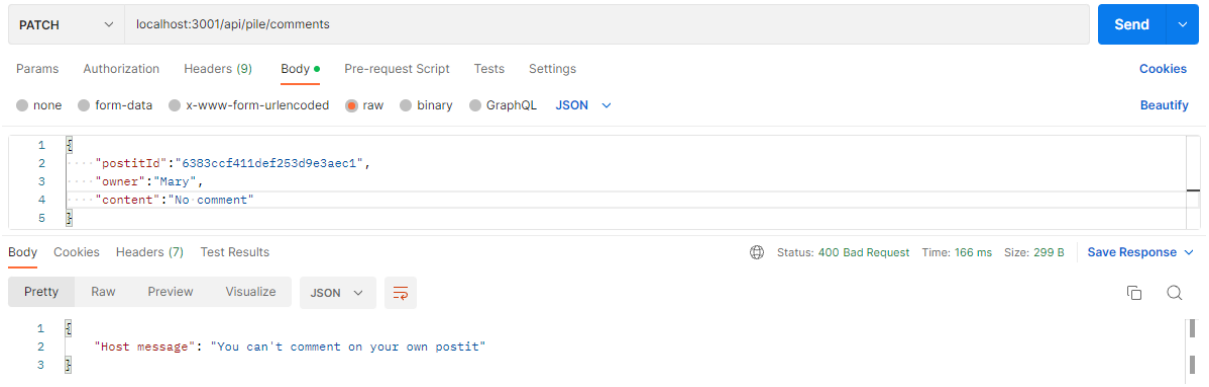

Figure 12. Mary tries unsuccessfully to comment on her own post.

Once they comment on Mary's post-it, Nick and Olga decide to like Mary's post-it. They like the post by submitting their name and the ID of Mary's post-it. They each like the post-it once, giving the post-it a total of two likes (Figure 13). Once they like the post-it they receive a host message informing that the like was successful and showing the post-it in full with comments and likes. Mary sees these likes and also tries to like her post-it. Unfortunately for Mary a post-

it owner cannot like their own post-its! Mary receives a status: 400 and a host message informing her of her error (Figure 14).

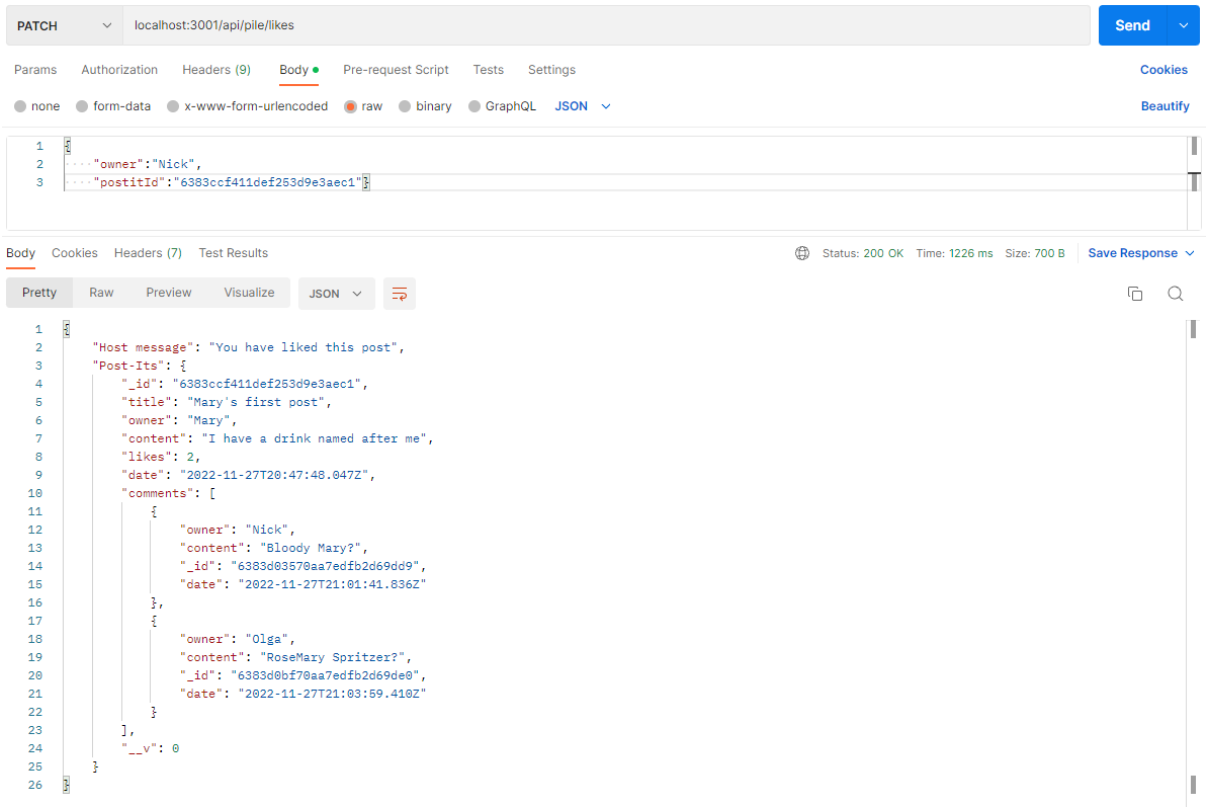

Figure 13. Nick and Olga both like Mary's post, as indicated by the 'likes' field incrementing by 2.

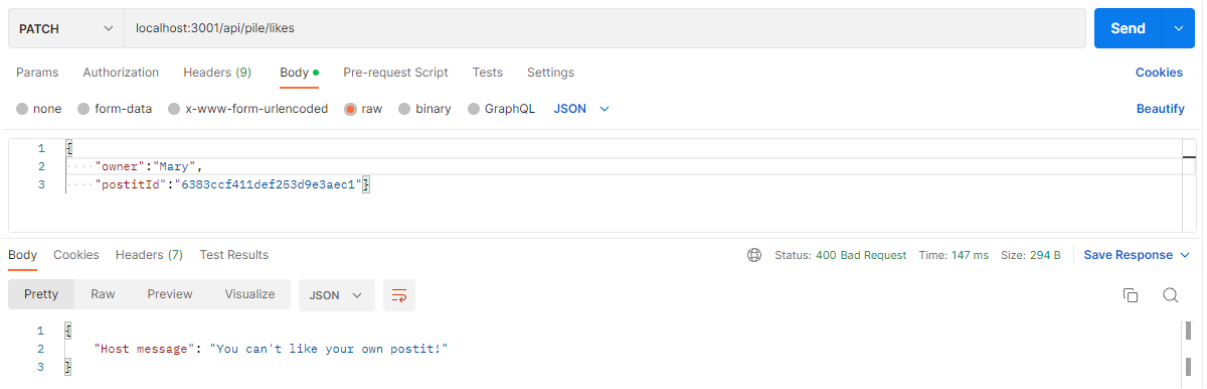

Figure 14. Mary tries to like her own post, but is unable to!

Finally, Nick decides to view the pile. He sees Mary's post at the top of the pile, since it has garnered the most likes, and sees the comments associated with the post-it. He then sees Olga's post, followed by his, as these are shown in chronological order.

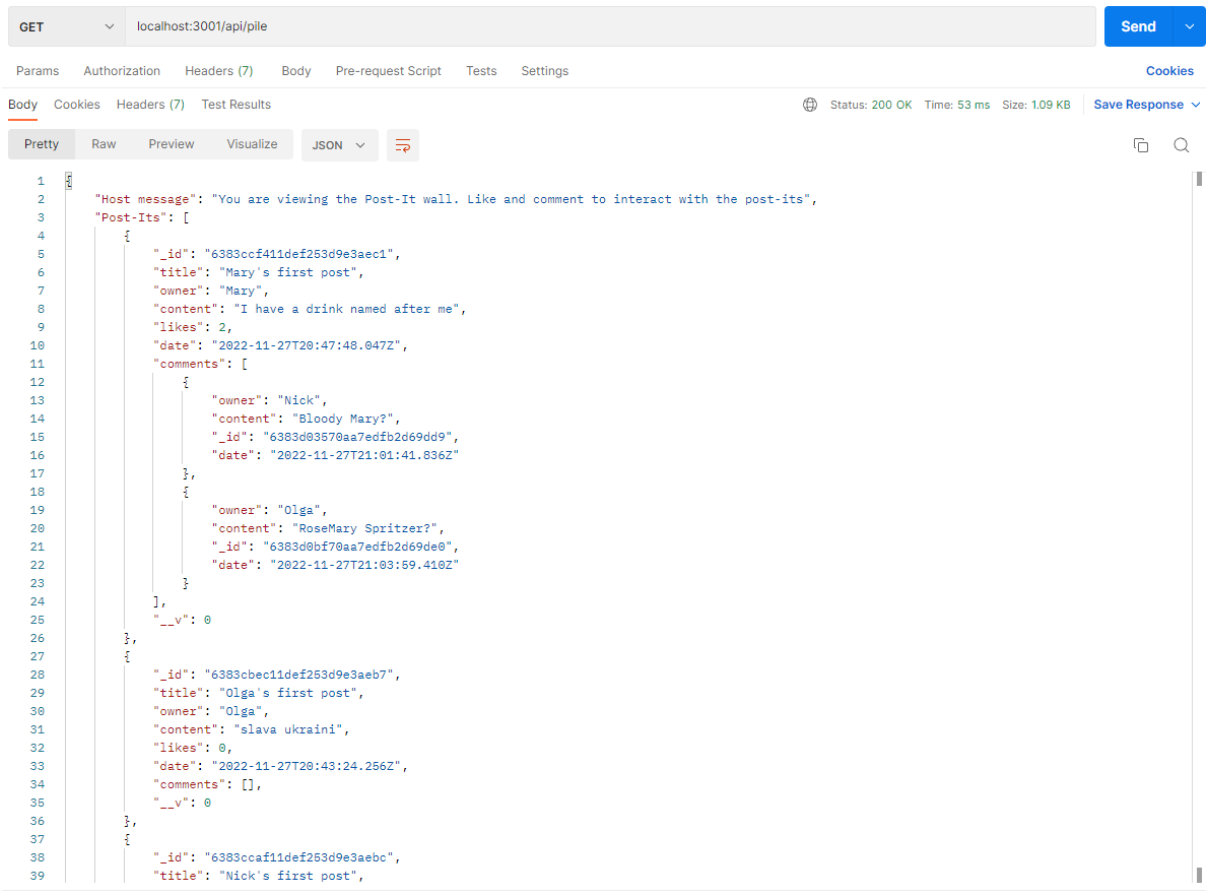

Figure 15. Nick views the pile and sees that Mary's post-it with 2 likes is at the top of the pile. The other comments are listed afterwards chronologically.

Conclusion

In this technical report, I have described the implementation of Post-It, a Cloud Software as a Service application that allows users to post information into a shared space. Users can interact with the posts, adding to them comments or likes. Firstly I describe the dependencies of the software and explain how to initialise the environment and install the libraries. I then detail how authentication and user verification is carried out in the software. In Phase C I explain how to use Post-It in a brief guide and how the software uses REST API to store and return data between the server, nodejs and mongoDB. Finally I demonstrate the use case of Post-It with users Olga, Mary, and Nick, who register, login and post, like and comment using the software.

I hope you enjoy using Post-It!

References

1. Express - Node.js web application framework. Available at: http://expressjs.com/. (Accessed: 27th November 2022)
2. nodemon. Available at: https://nodemon.io/. (Accessed: 27th November 2022)
3. Getting Started with MongoDB & Mongoose | MongoDB. Available at: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/. (Accessed: 27th November 2022)
4. dotenv - npm. Available at: https://www.npmjs.com/package/dotenv. (Accessed: 27th November 2022)
5. body-parser - npm. Available at: https://www.npmjs.com/package/body-parser. (Accessed: 27th November 2022)
6. joi - npm. Available at: https://www.npmjs.com/package/joi. (Accessed: 27th November 2022)
7. bcrypt - npm. Available at: https://www.npmjs.com/package/bcrypt. (Accessed: 27th November 2022)
8. jsonwebtoken - npm. Available at: https://www.npmjs.com/package/jsonwebtoken. (Accessed: 27th November 2022)