PIXSTAR

WEBSITE REPORT

PINTERST CLONE

By Syed Nasiruddin

Introduction

Within this report, I will meticulously cover the different tabs of the website, various processes involved (sign up, post creation, etc.), the webservices used, client and server side testing, HTML and CSS validation, security, privacy and legal issues.

Through a detailed analysis, I aim to convey the different functionalities offered by the website

Overview

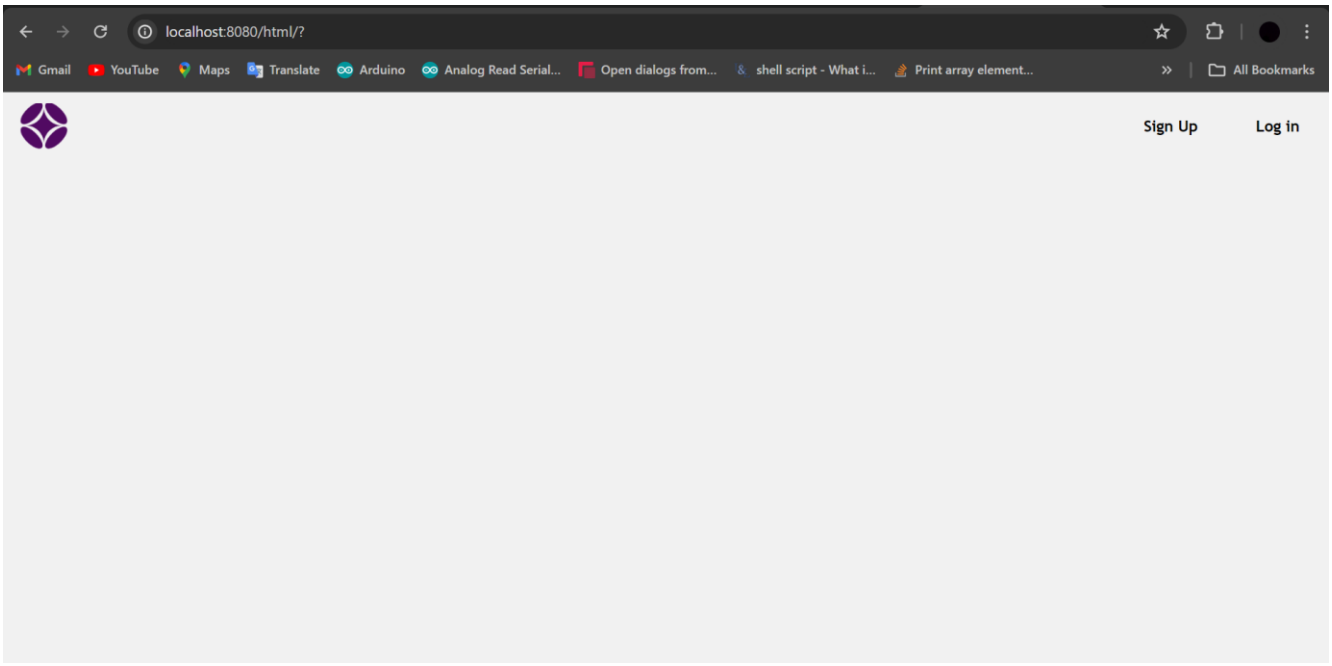
Pixstar is an image-sharing platform that offers a diverse range of functionalities designed to enhance the social networking experience.

Users can connect with friends and followers by following each other's profiles. Once connected, users can create and view posts, allowing for seamless expression of ideas and experiences. It also enables users to explore content from both followed and unfollowed users, broadening their horizons and fostering discovery of potentially new friends! Additionally, users can engage with posts by liking and commenting on each other’s posts, sparking meaningful conversations and interactions between each other. The website offers powerful search capabilities as users can easily find posts based on keywords or topics of interest, ensuring they never miss out on relevant content.

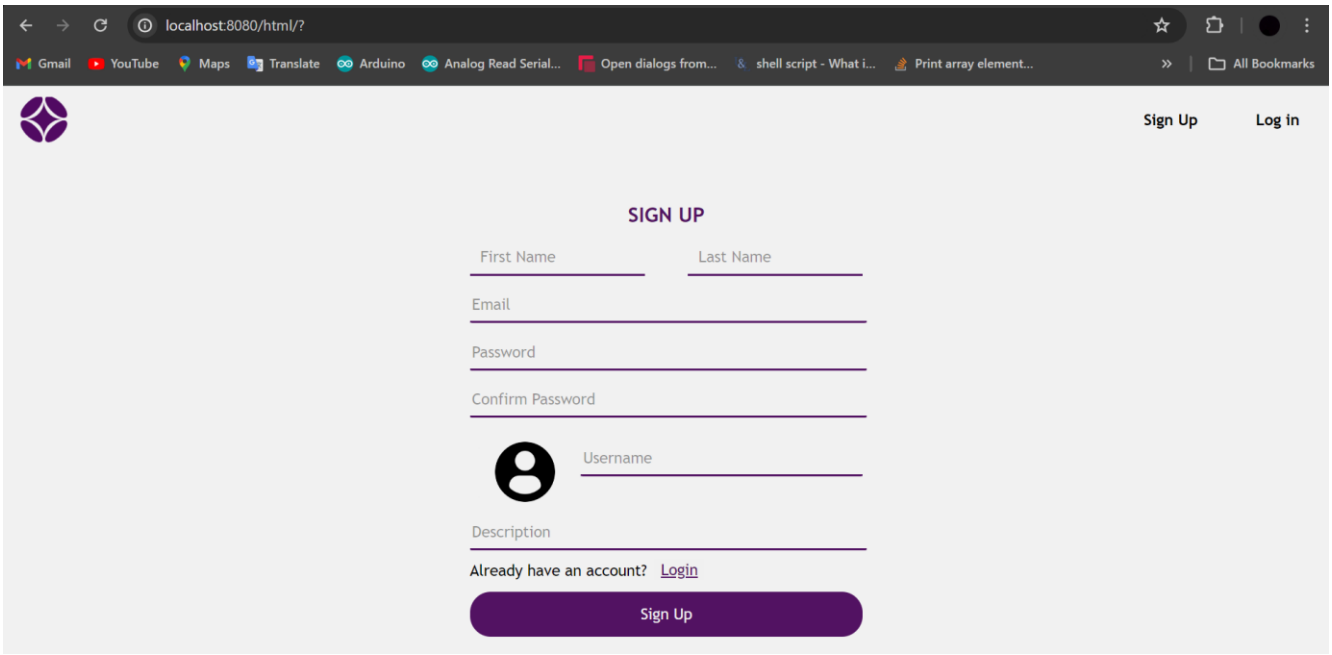
Whether its seeking inspiration, connection, or entertainment, Pixstar provides the tools and features to elevate the social networking experience.

Website Tabs

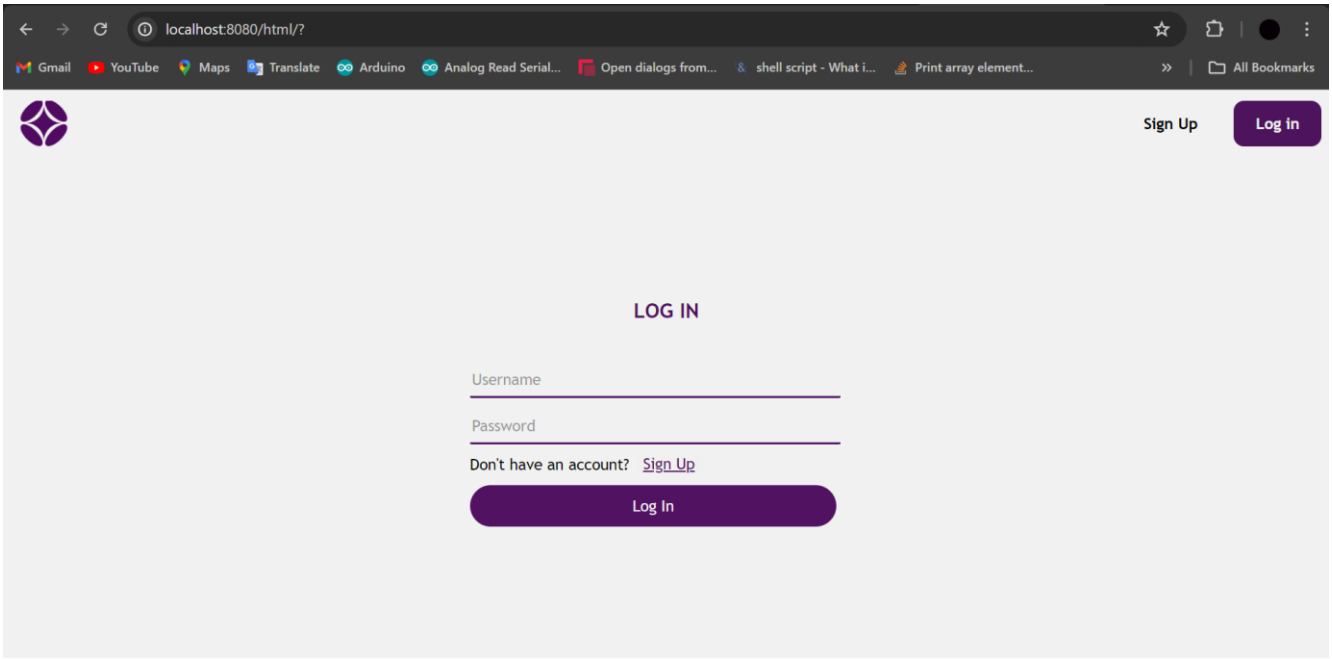
When the user first loads the website, this is what they see:



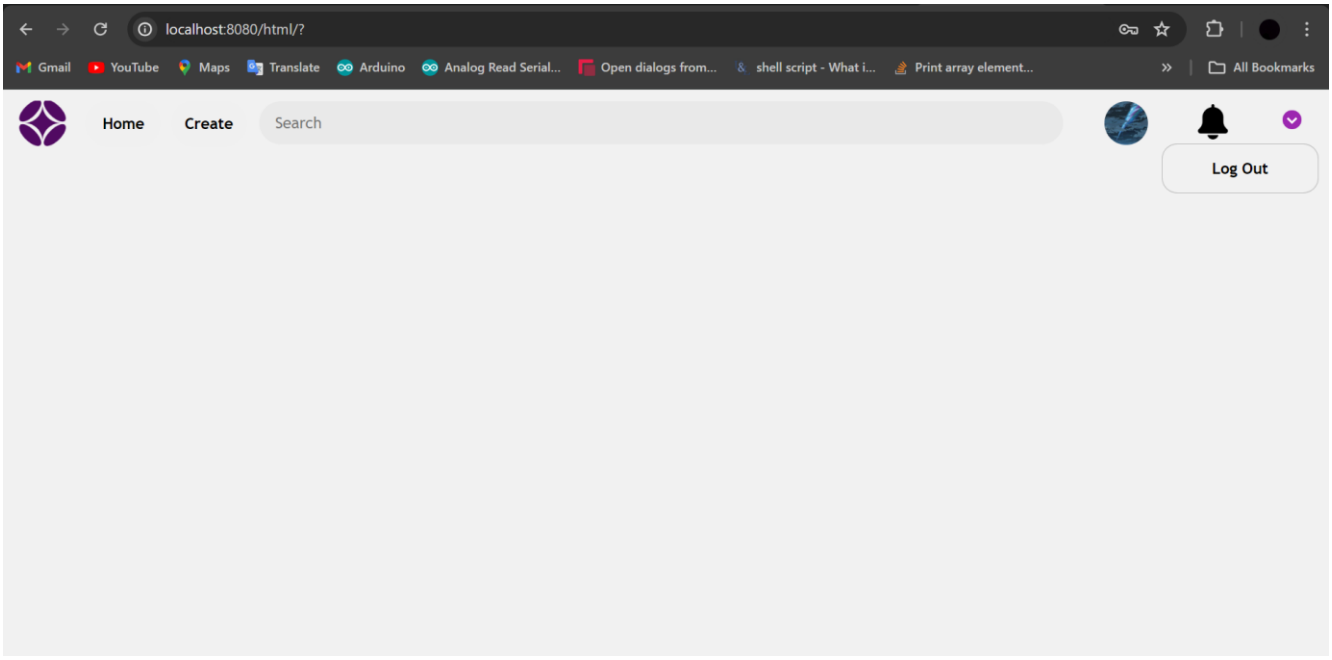
They have the option to sign up and log in, the sign up page is as seen below.



And below is the login page.



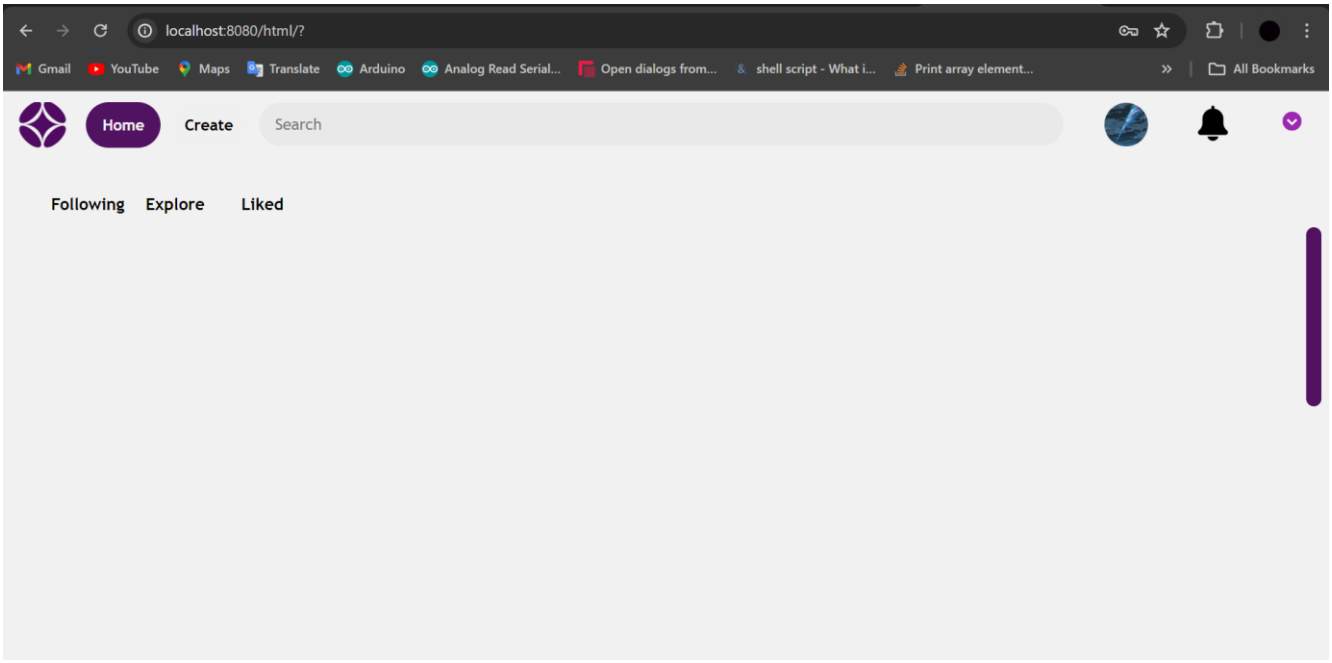
Once the user has logged in, they will be show this screen:



The page now has a lot more options, in the top left corner are the buttons that lead the user to the home page and the post creation page, in the middle is the search bar where the user can search users in the format “@username” or they

can search posts by title by just typing in the title, on the top right is profile photo of the user which upon clicking it leads the user to the account tab, there is notification tab as well that shown when the user clicks on the bell, and a dropdown that allows the user to log out as and when they require.

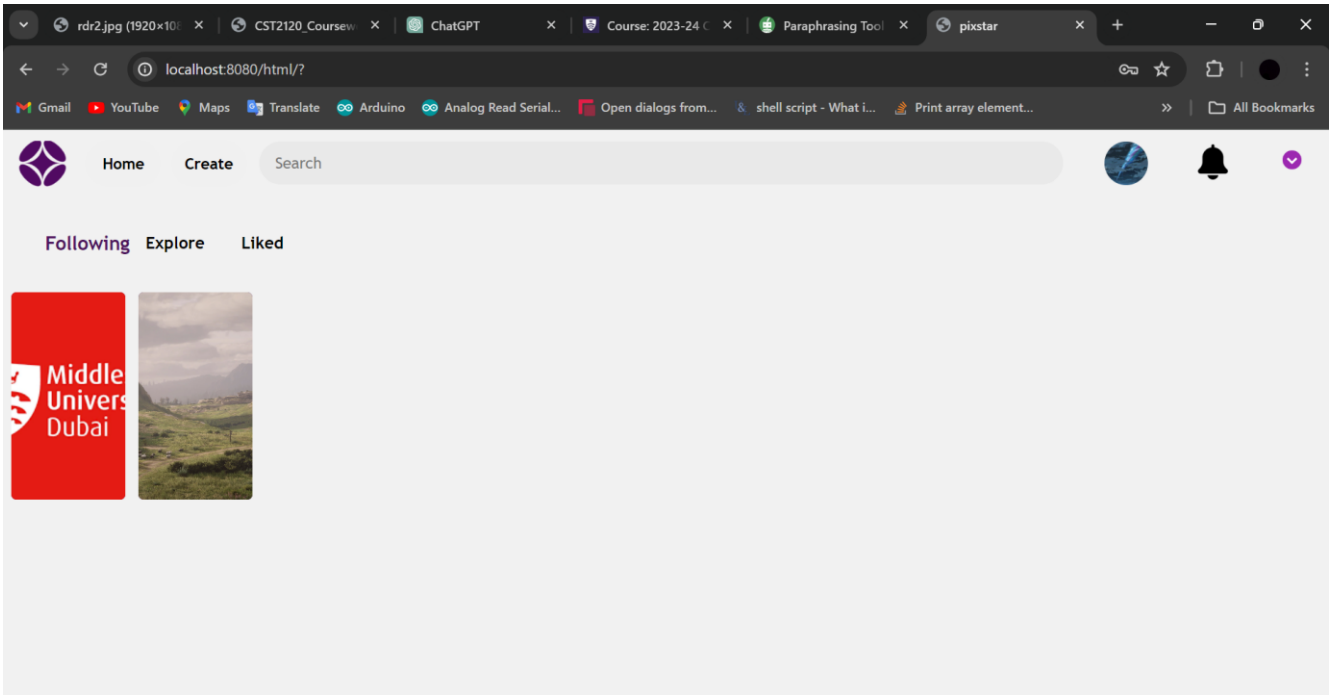
Starting with the home page, once the user clicks the “home” button, the below is shown



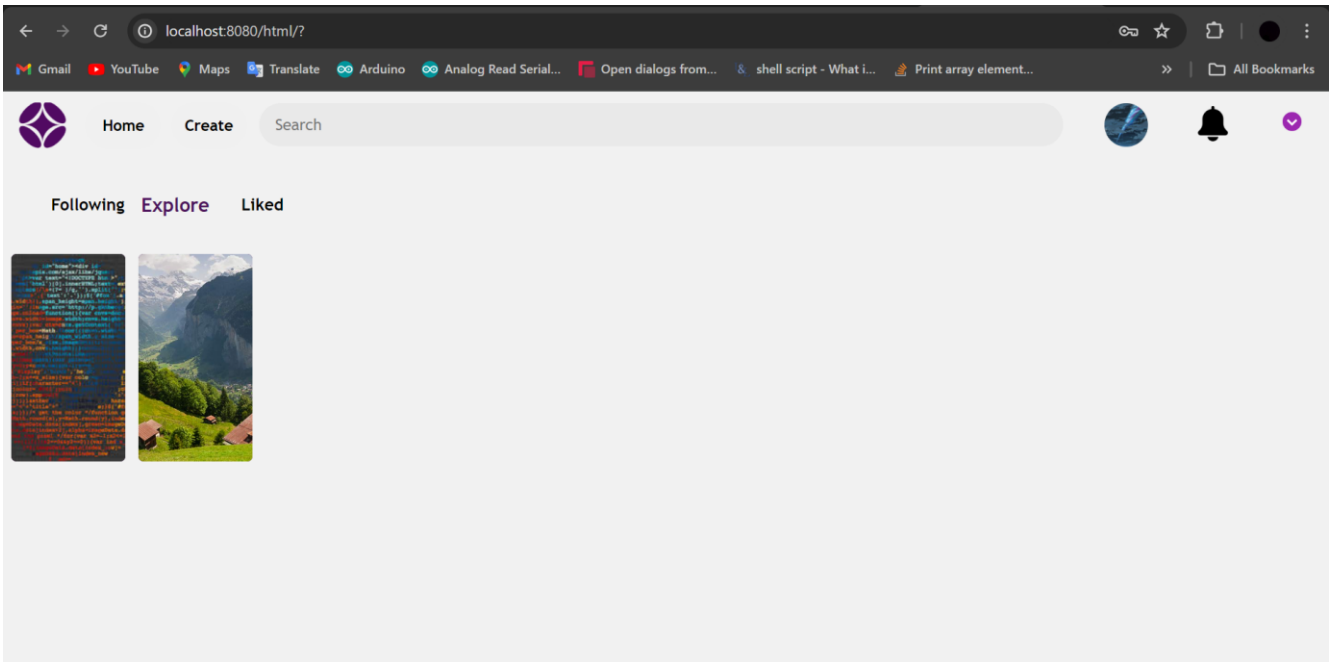
The user can choose which of the three sections they want to view, the following tab consists of all the users the current user is following, the explore tab consists of users that are not followed by the user, and the liked tab includes all the posts the user has liked.

The tabs can be seen below,

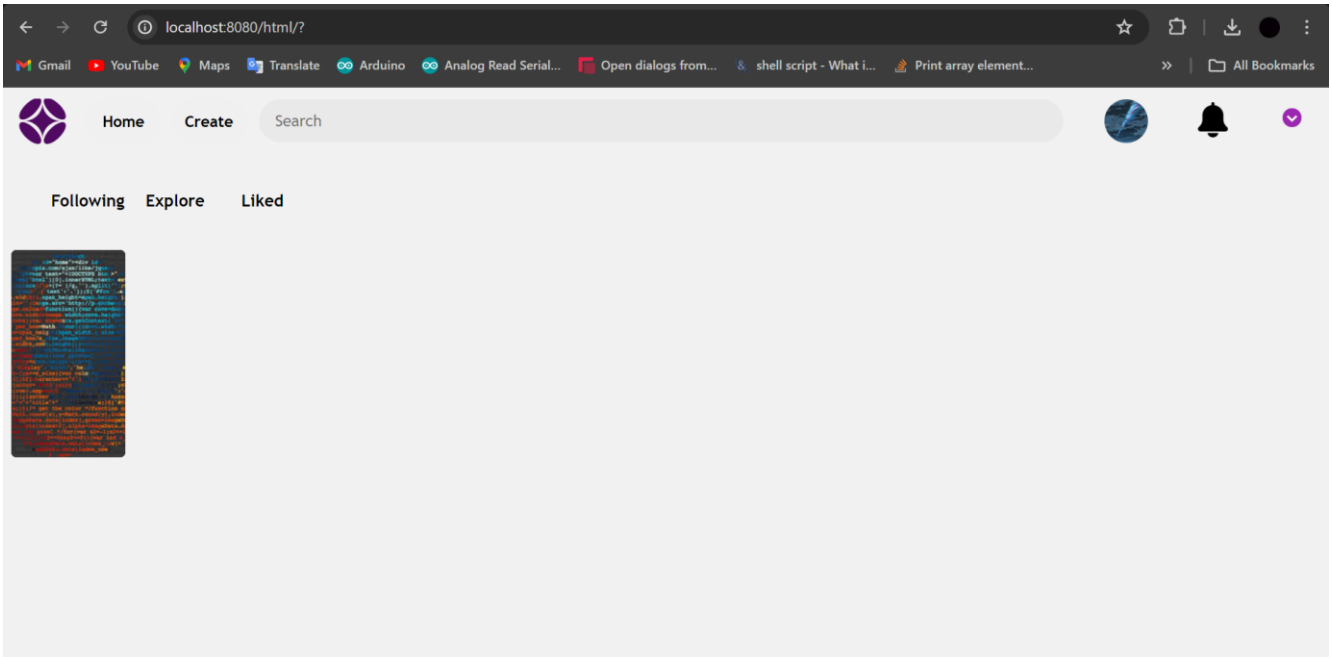
The following tab:



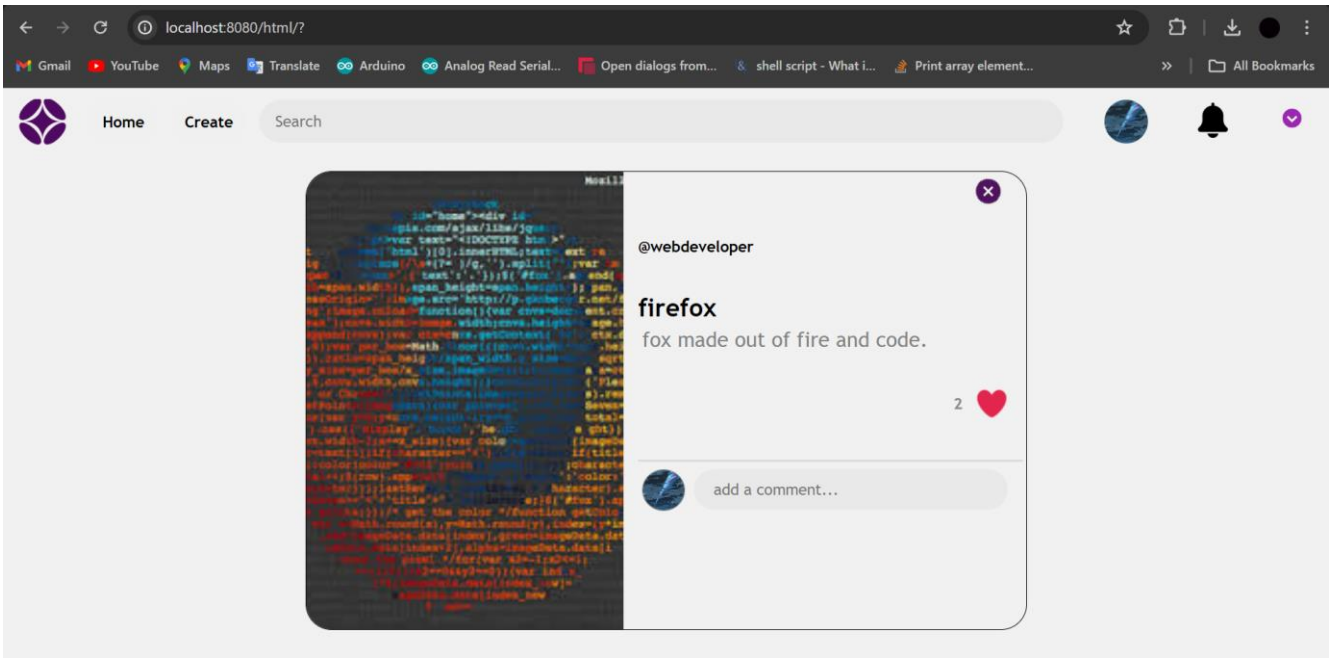
The explore tab:



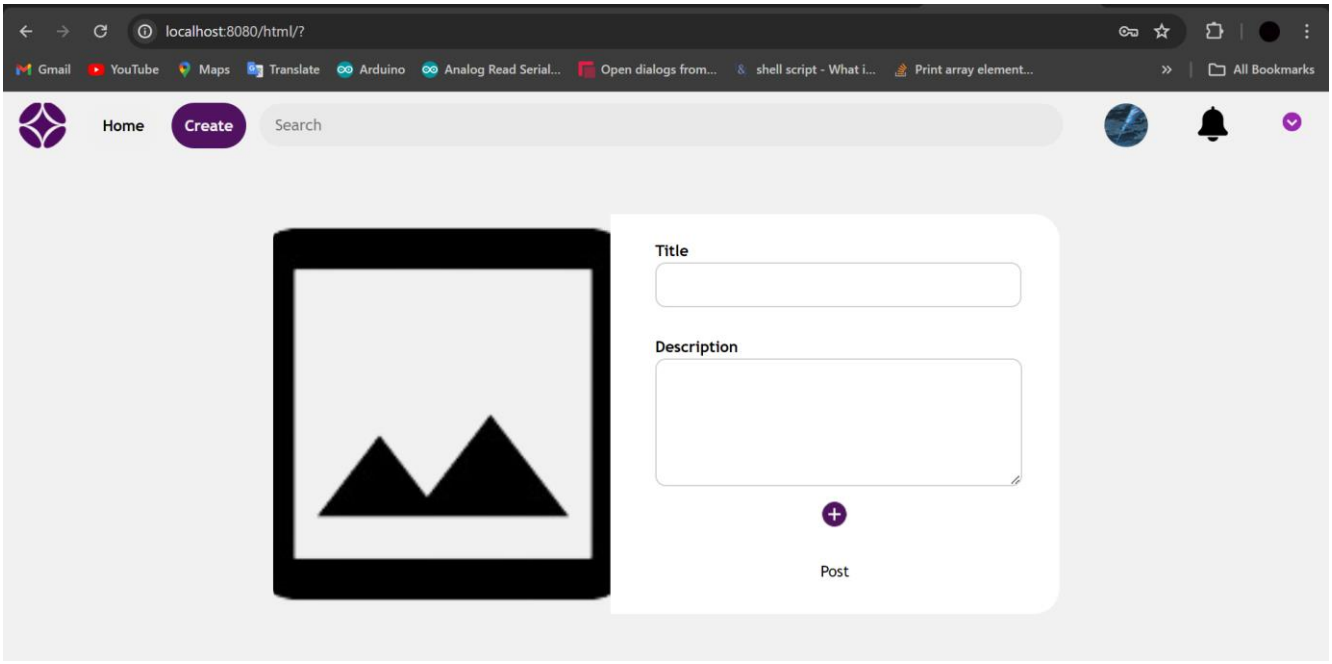
The liked tab:



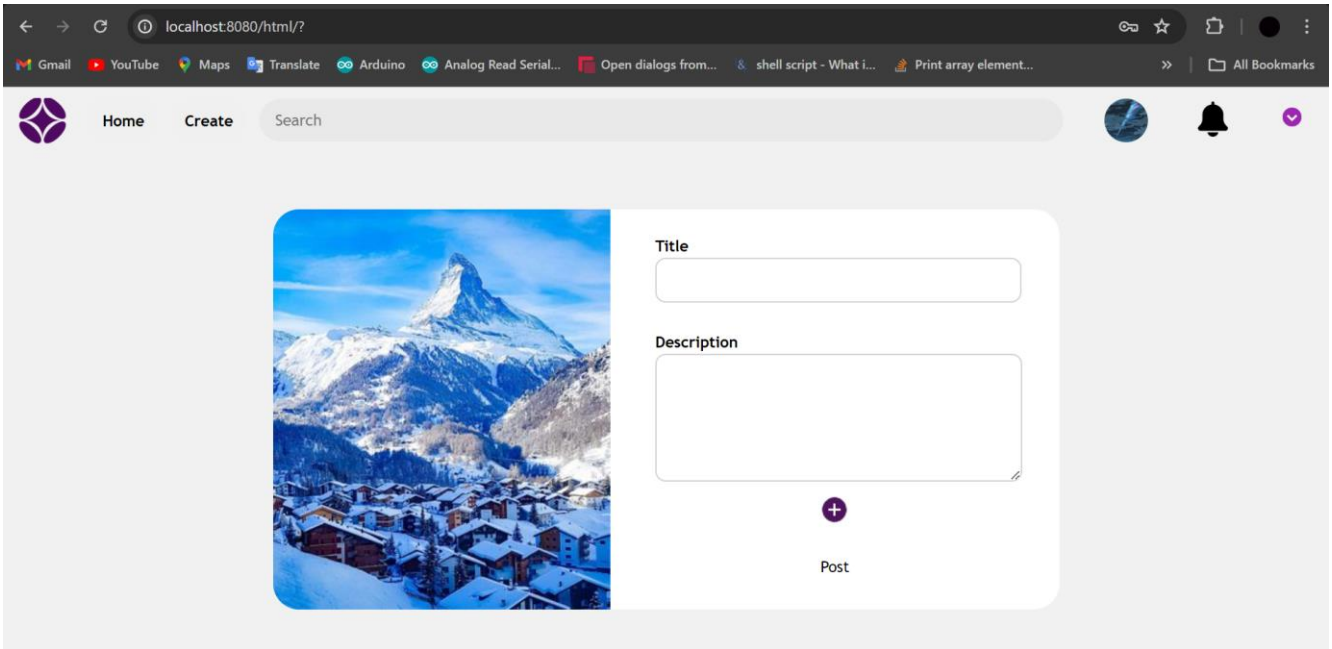
The user can click on the post to view more details, and interact with it by liking and commenting as well.



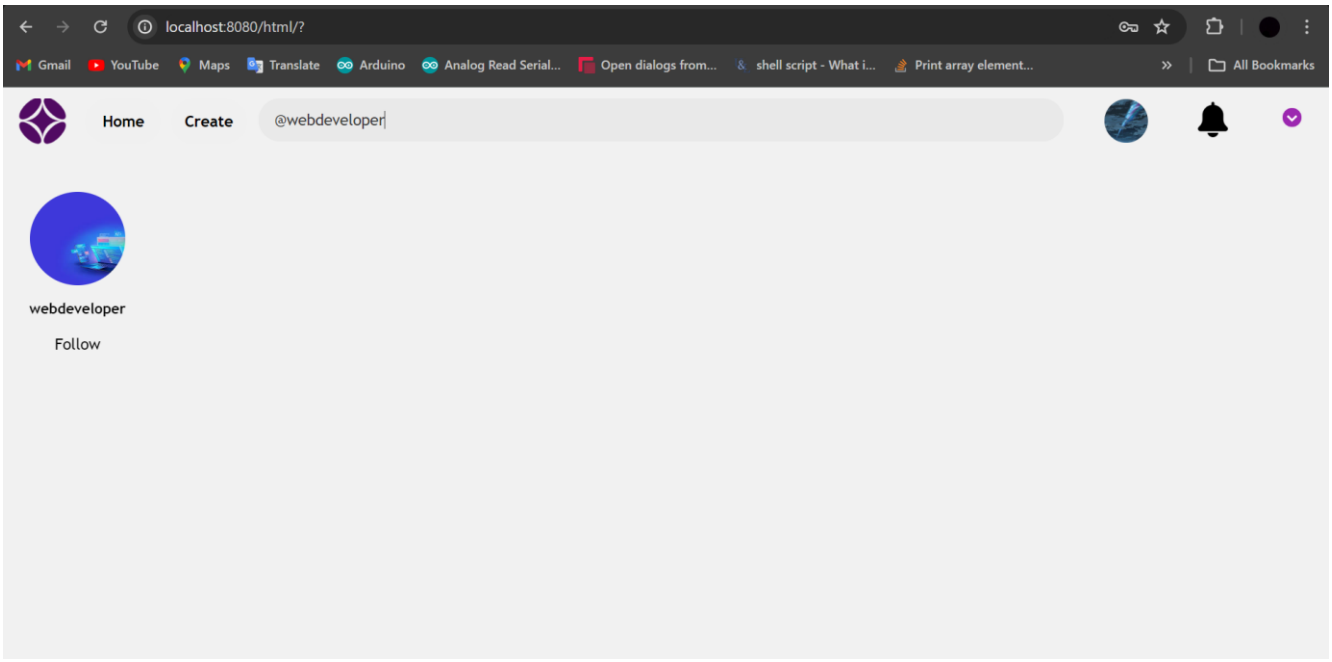
Moving on to the create tab,



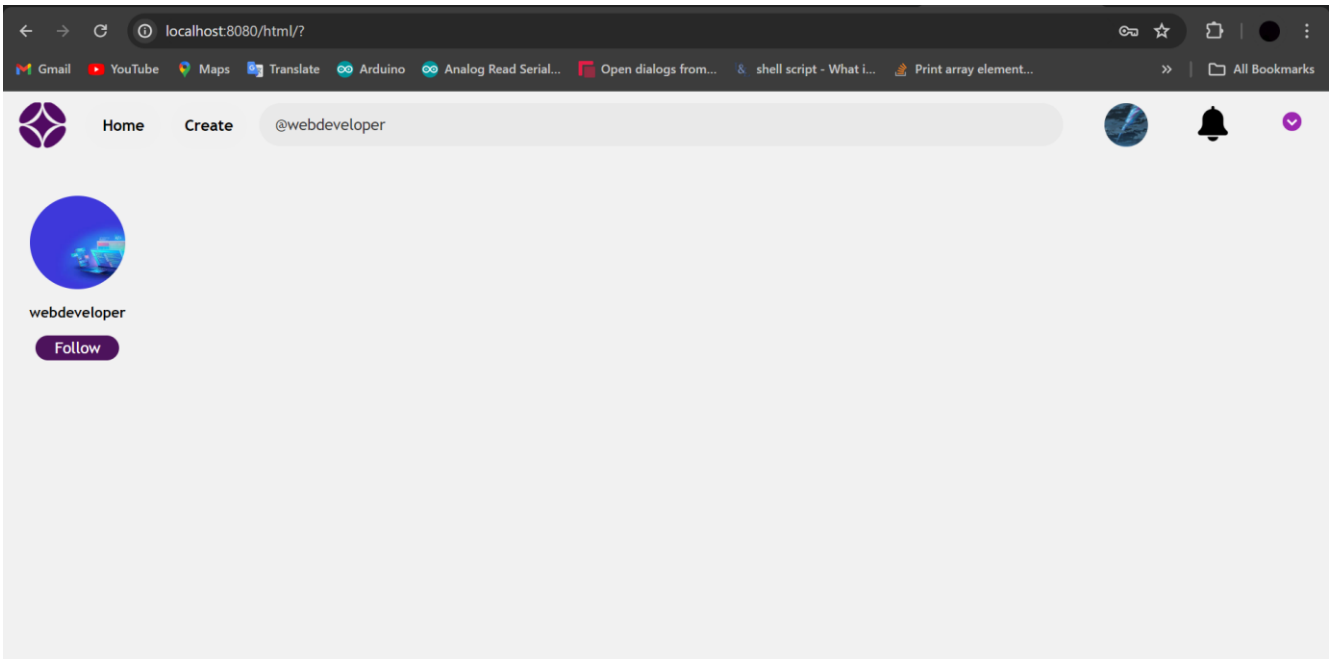
It allows the user to assign a title, description and an image (that should be less than 1 MB). The image is attached by clicking on the purple “plus” symbol, a preview is shown on the left.

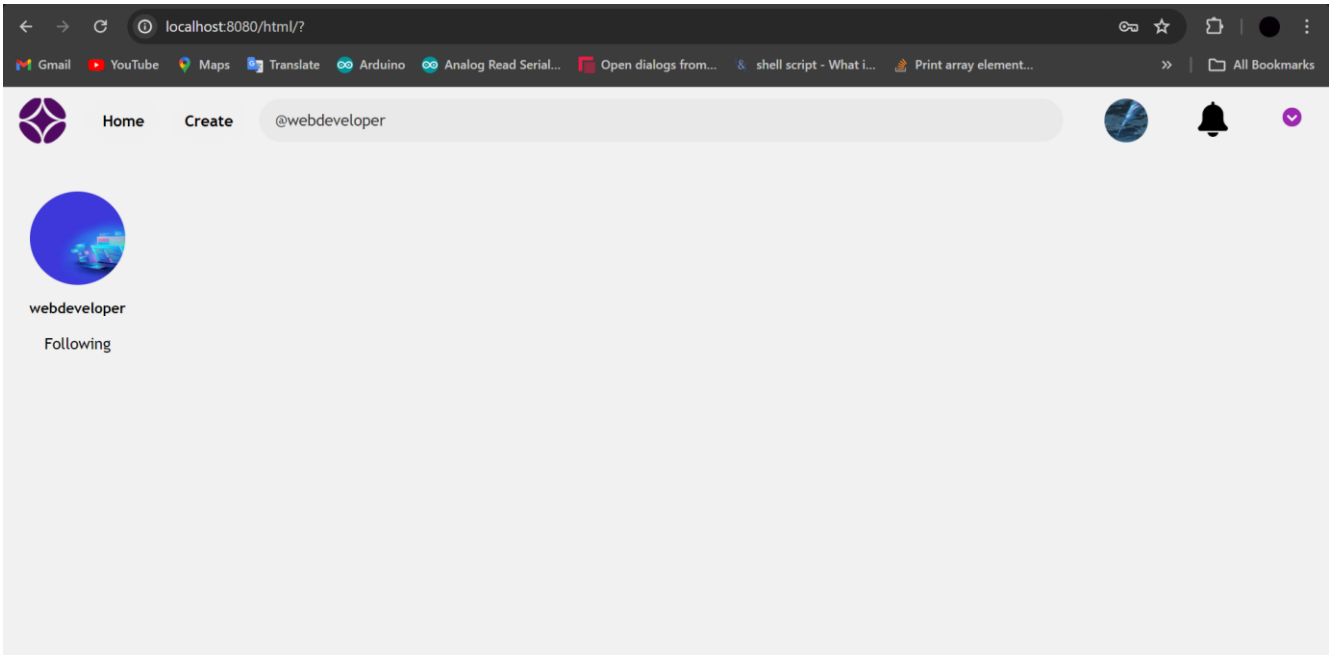


The searchbar provides both user and post searches, the below is the result of a user search,

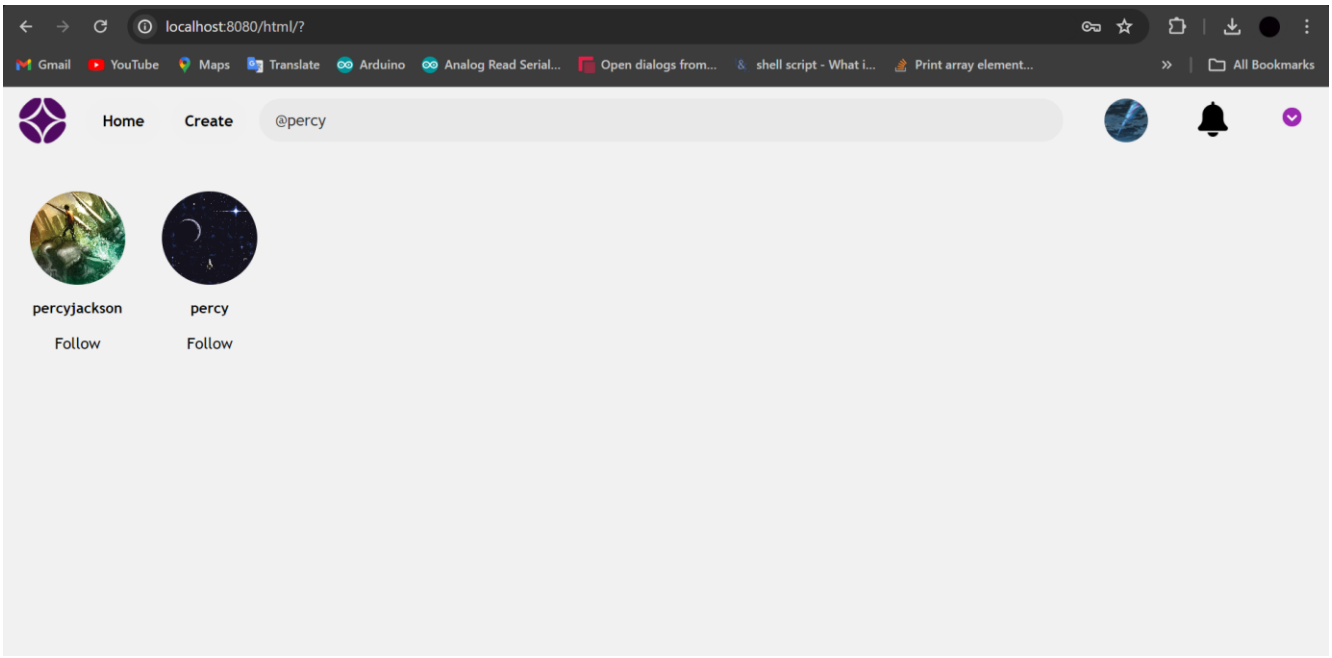


The user can follow or unfollow users after searching them,

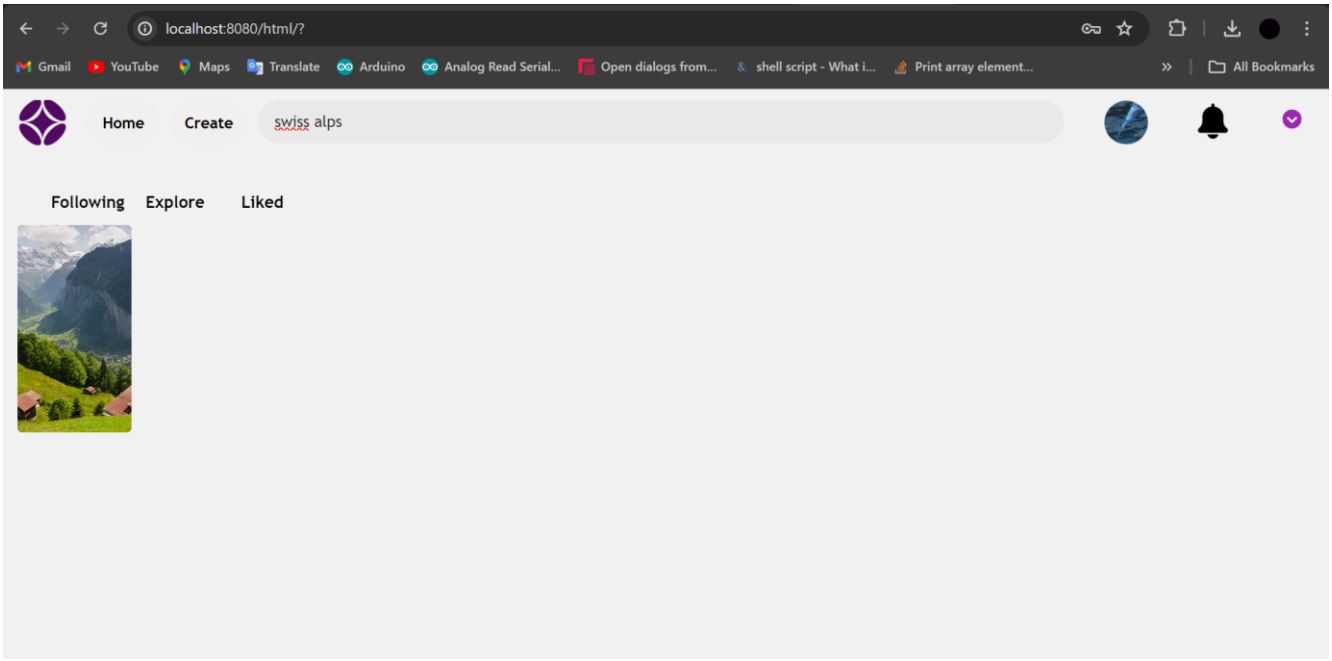


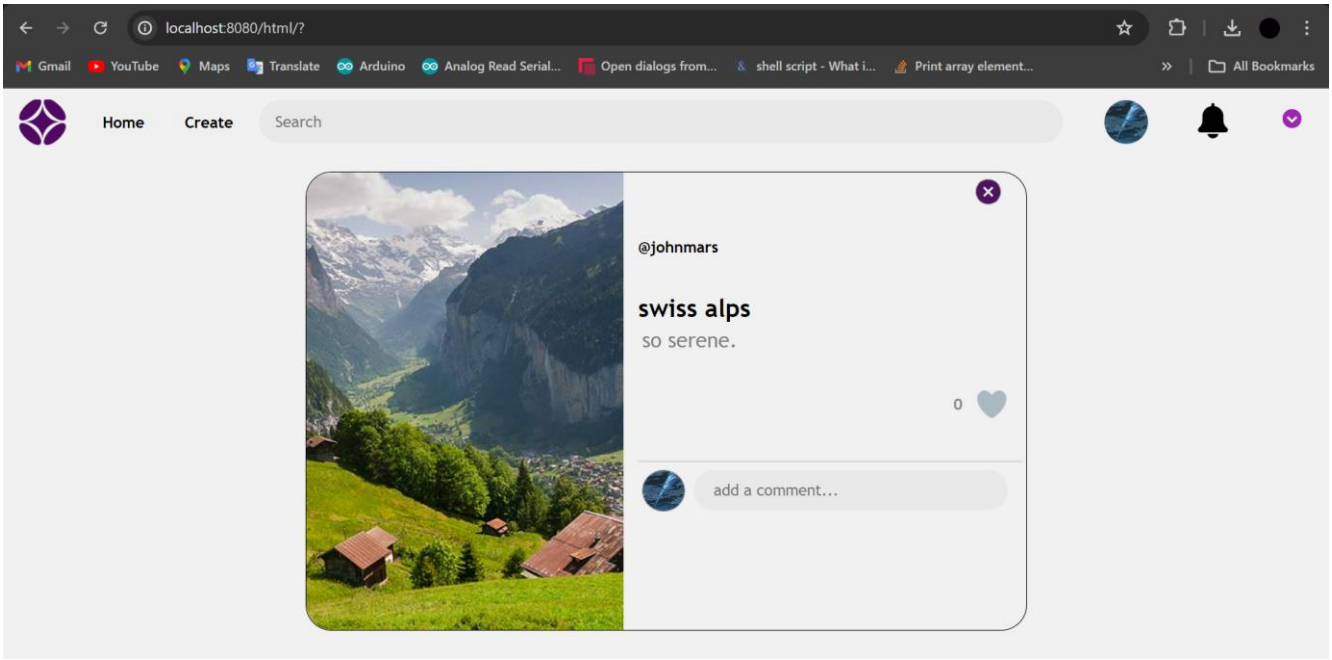


If a few users have a part of the same name, both users are shown:

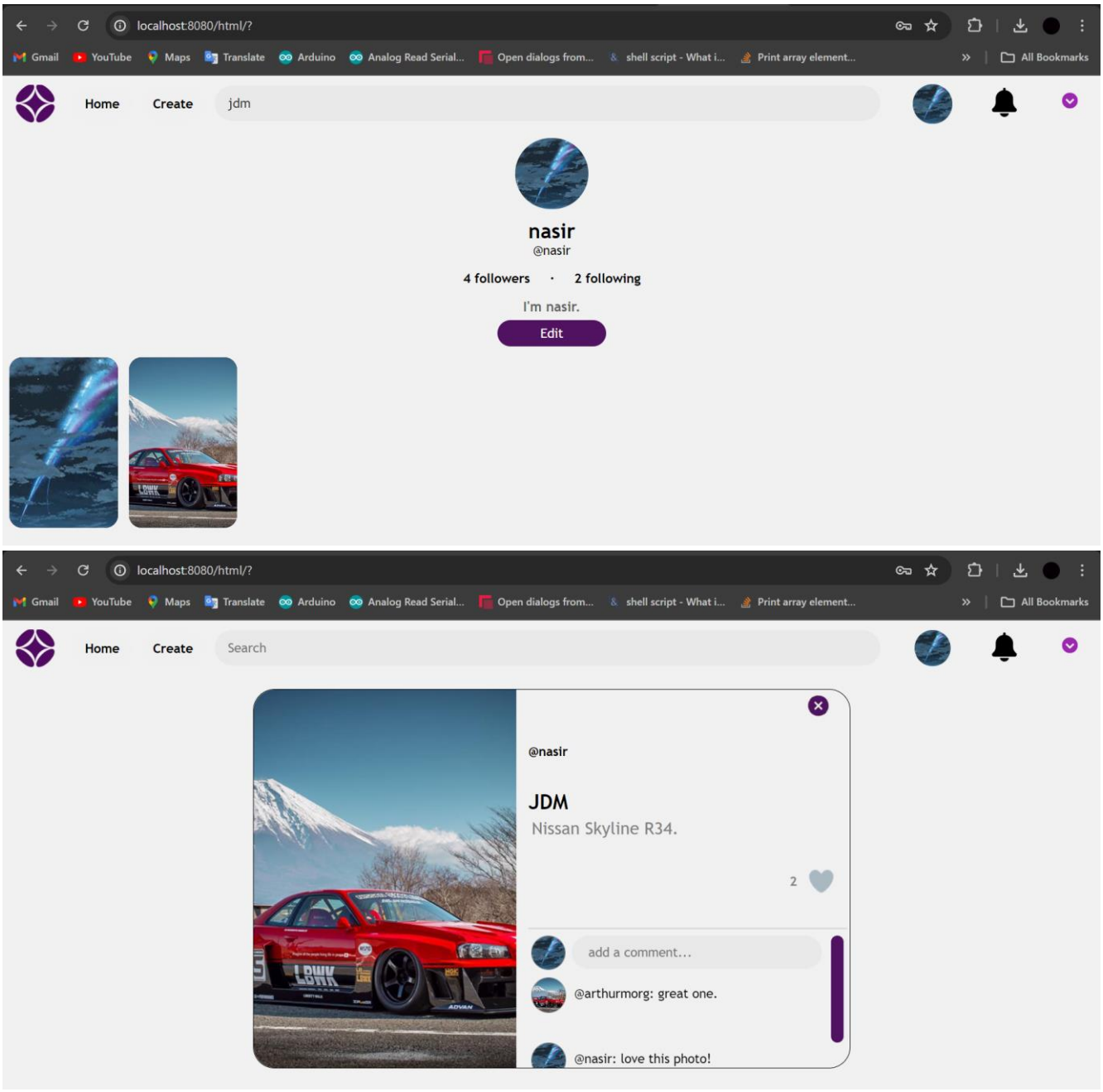


posts can be searched by searching key words



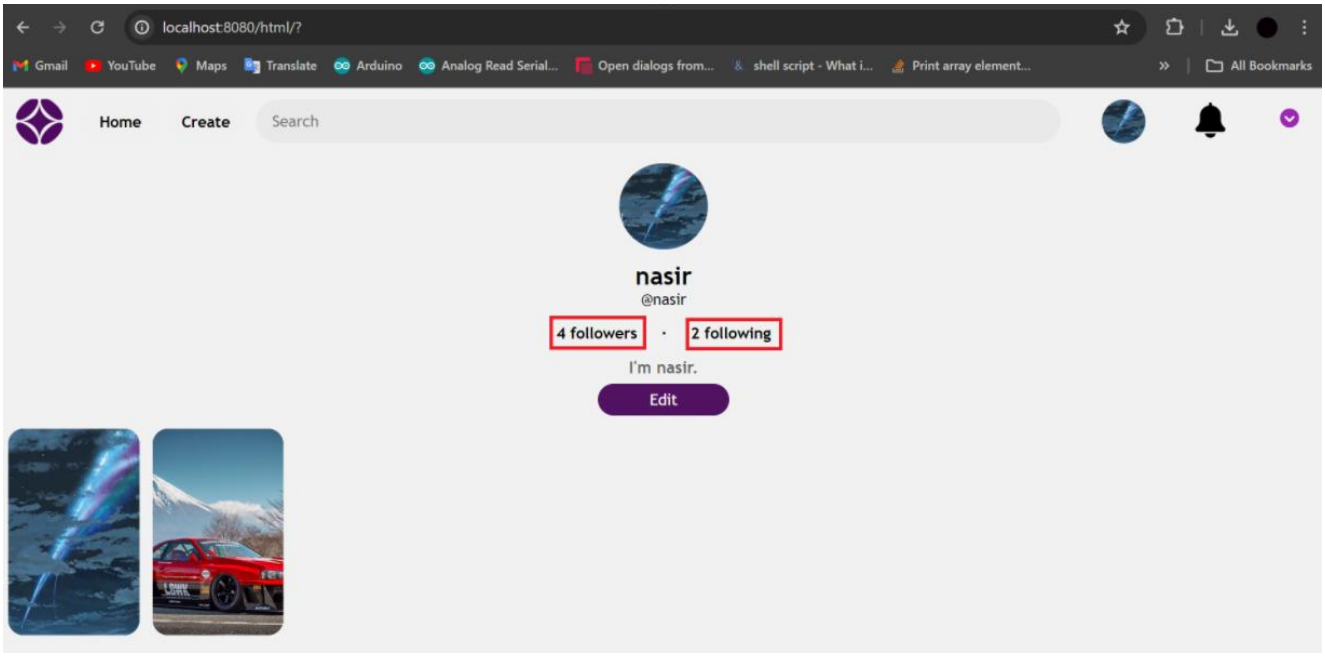


By clicking on the user's profile picture, they can view the account tab, which allows them to see their own posts, followers, and following. They are also given the option to edit their profile.

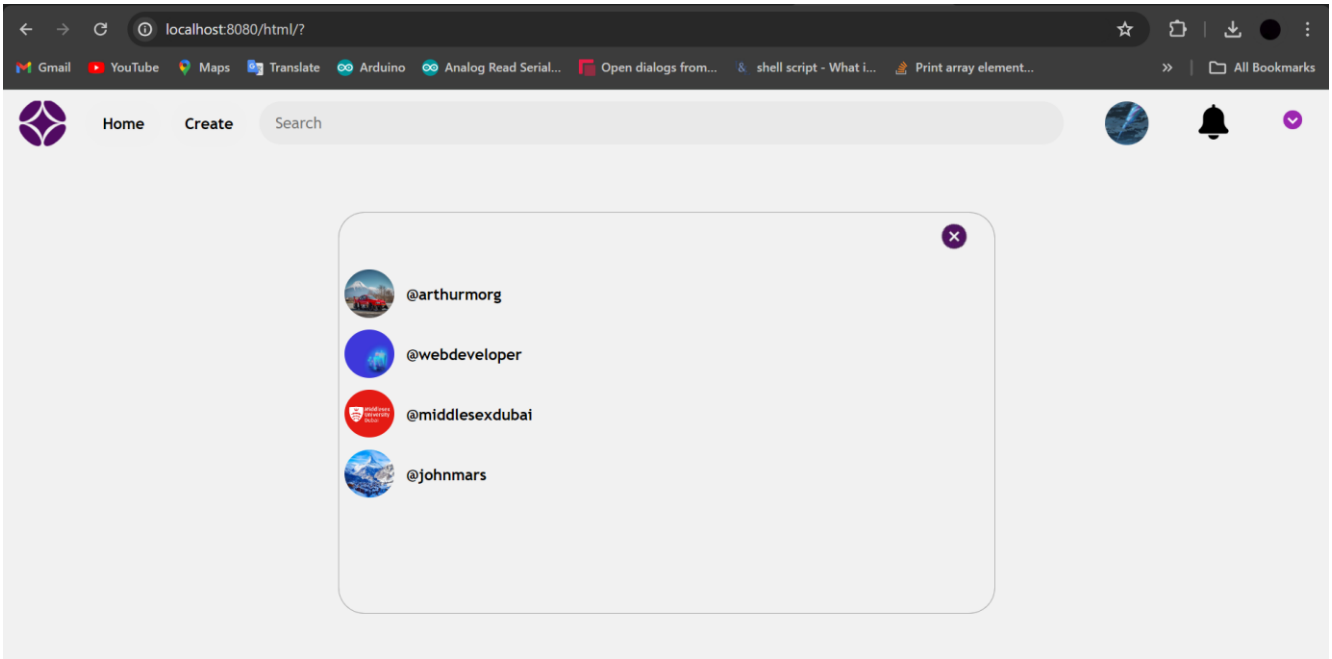


The newer comments appear at the top of the comment section.

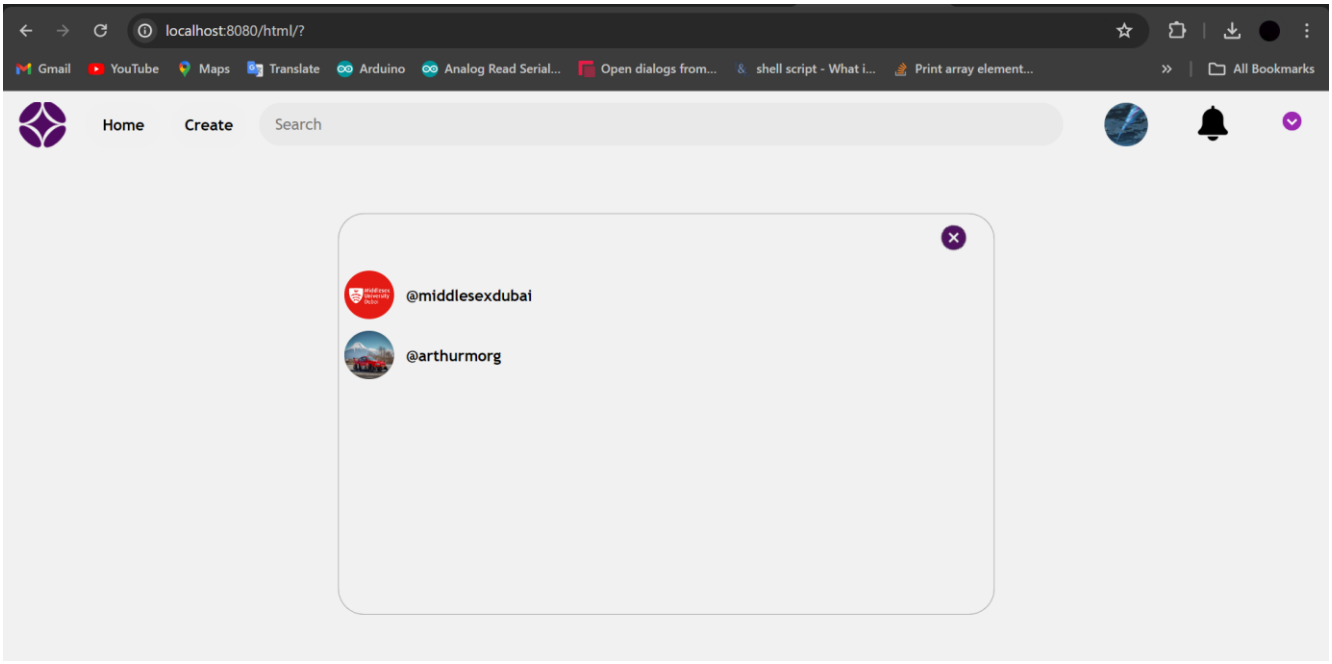
Followers can be viewed by clicking on “followers” on the account tab, similary, following can be viewed by clicking on “following”, the buttons have been highlighted in red for infomative purposes.



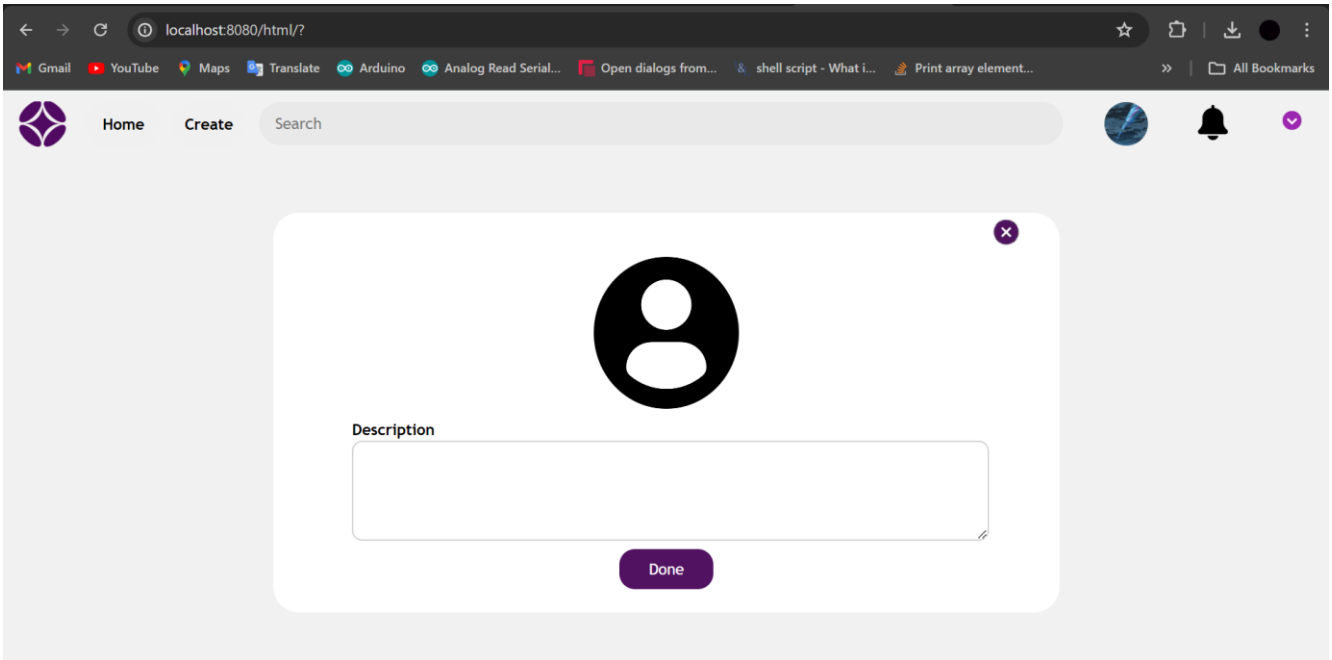
by clicking on the “followers” button, the followers username and profile pictures are shown.



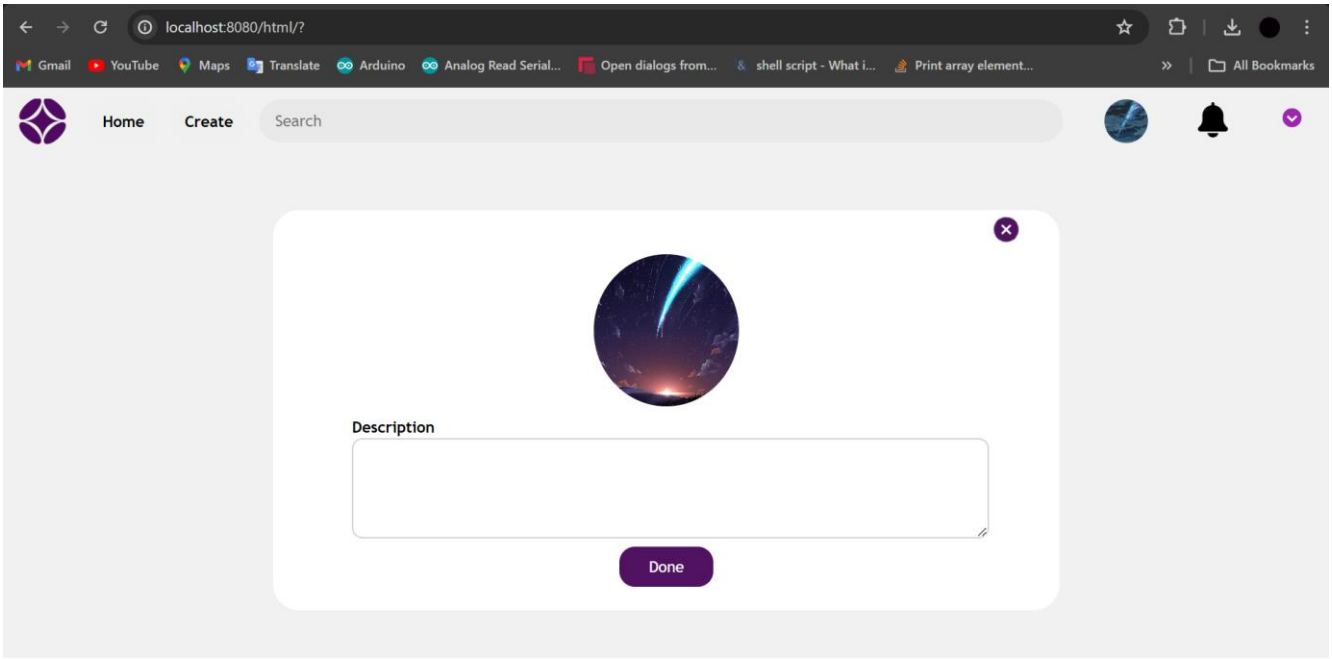
Same for the following button,



the edit profile section can be seen if the user clicks on the “edit” button,

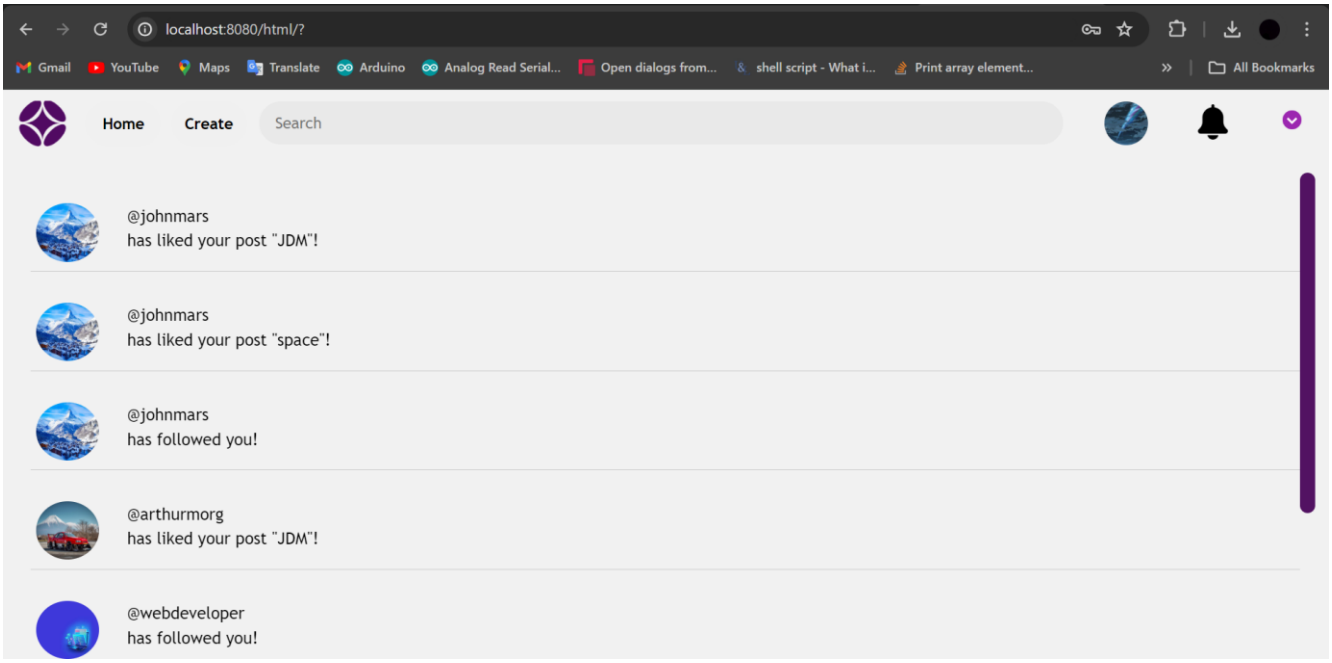


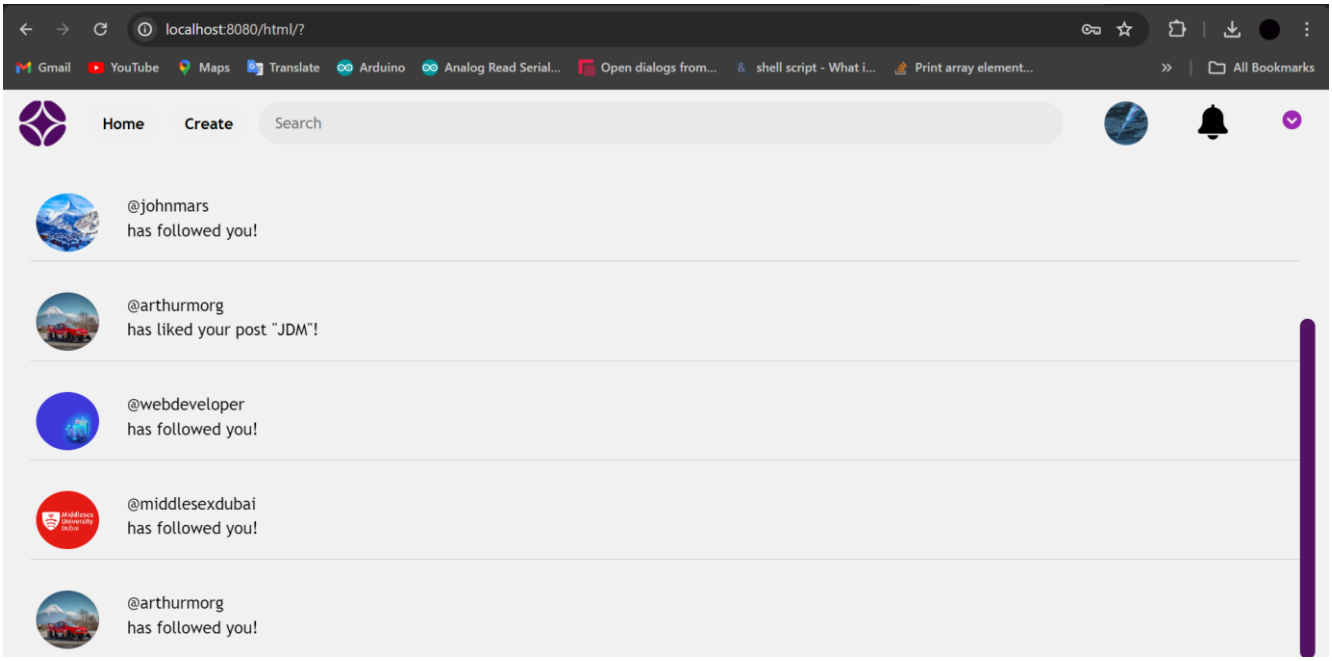
by clicking on the placeholder profile photo, they can attach the new profile photo and it shows a preview as well.



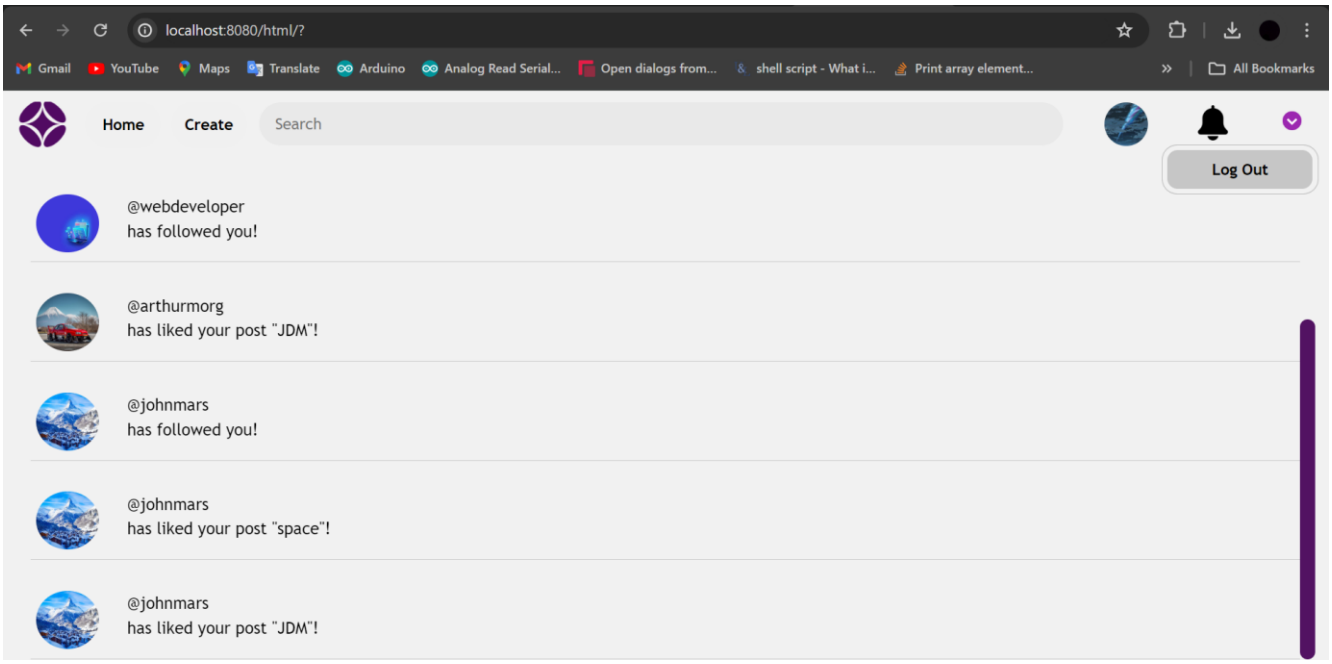
The user can exit by clickong the “x” button on the top right of the window of both the edit profile section and the expanded posts.

The viewer can get notifications about who followed them and liked their posts, if a person unfollows or unlikes their post, this notification is deleted. These notifications are scrollable and the newer notifications appear the top of the page.

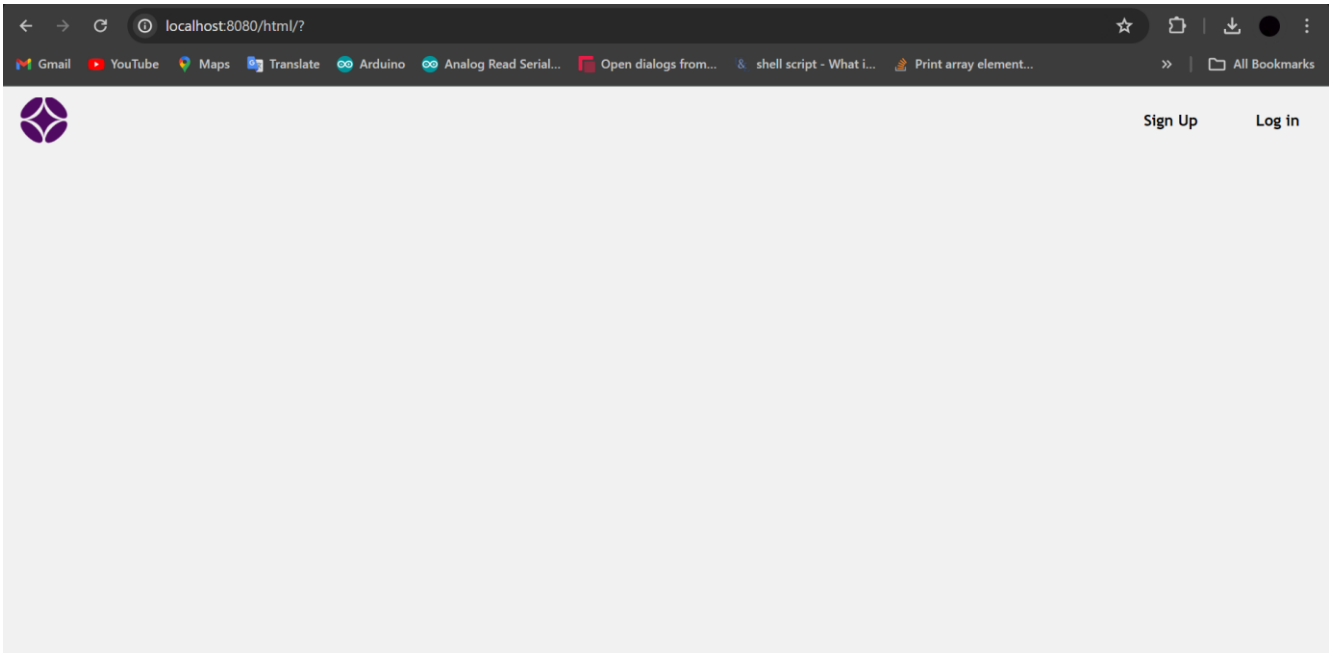




Finally the user an log out by clicking the button in the dropdown,



and the user is redirected back to the first page,



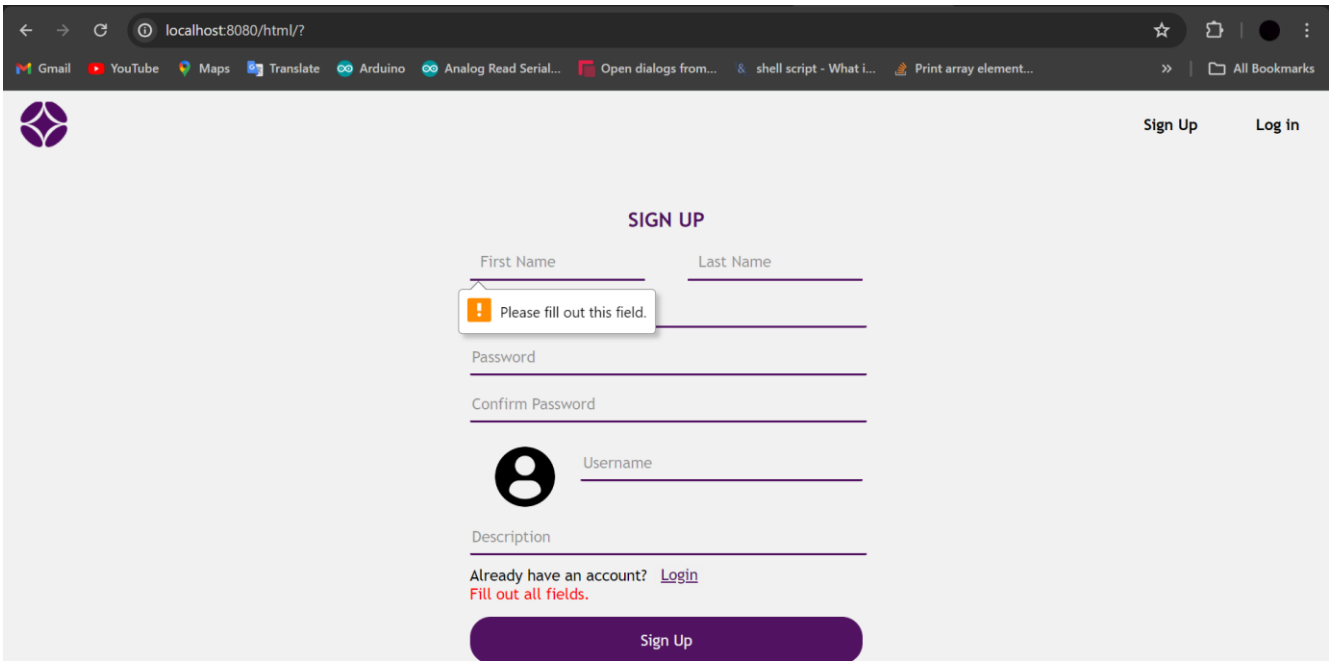
Processes

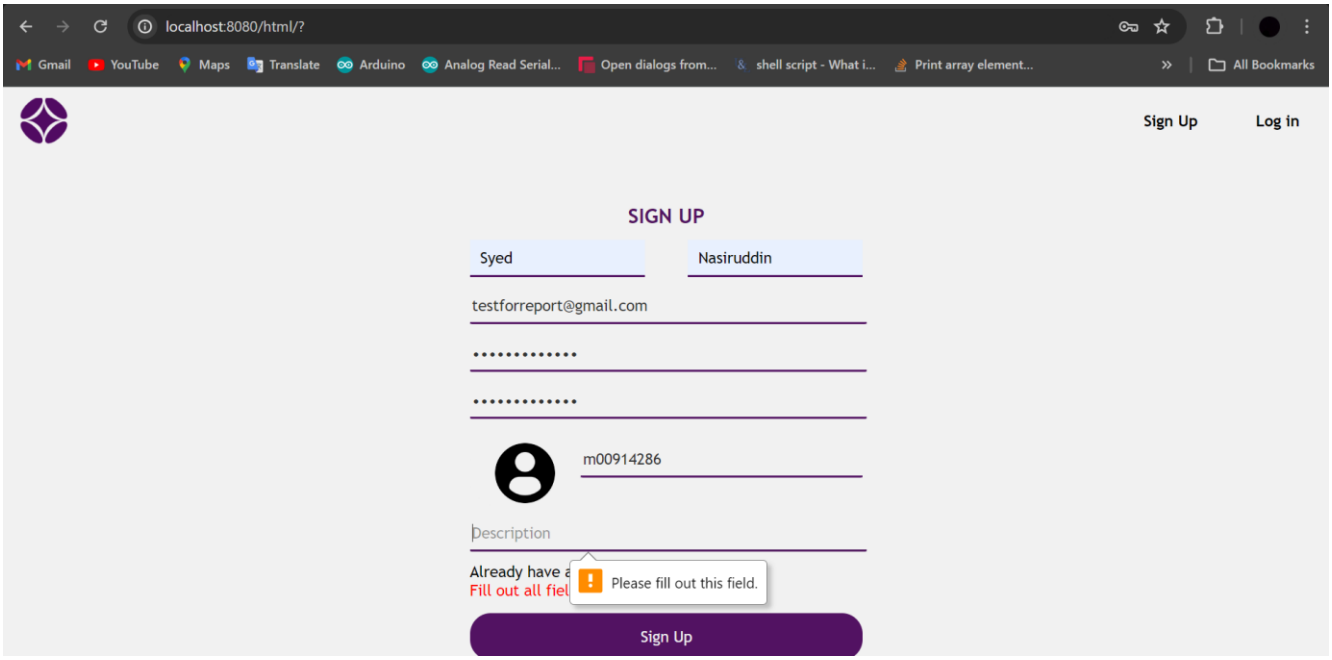
Sign Up

The user is required to put in their first name, last name, an email with a valid domain (outlook.com, yahoo.com, gmail.com), password, confim password, profile photo, username, and description.

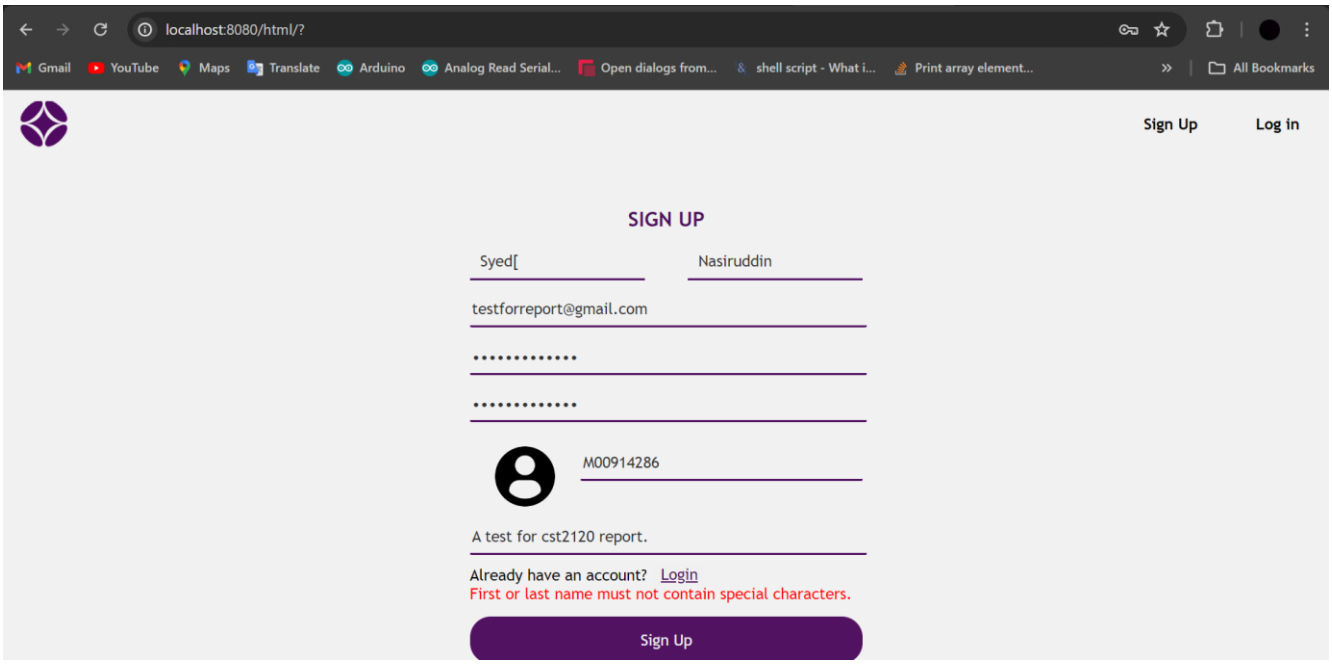
Before the form is submitted, validation is performed and below are the different types of errors that the form catches.

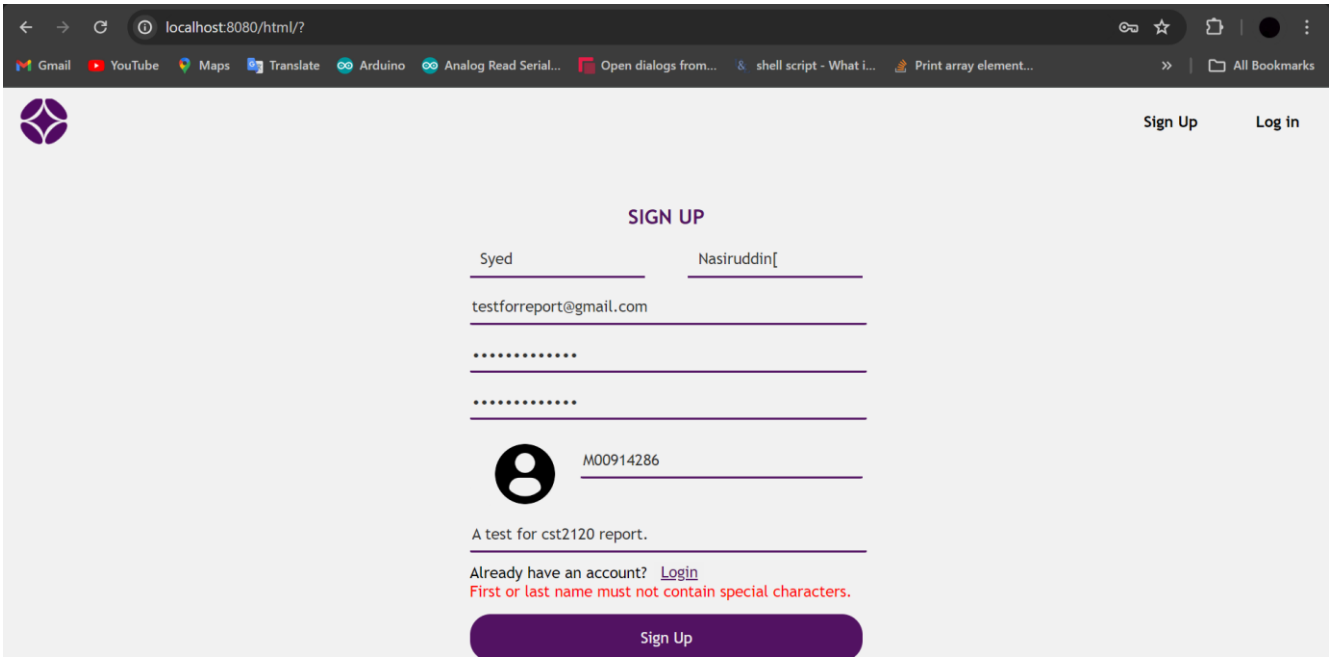
1) Fields are not filled out.



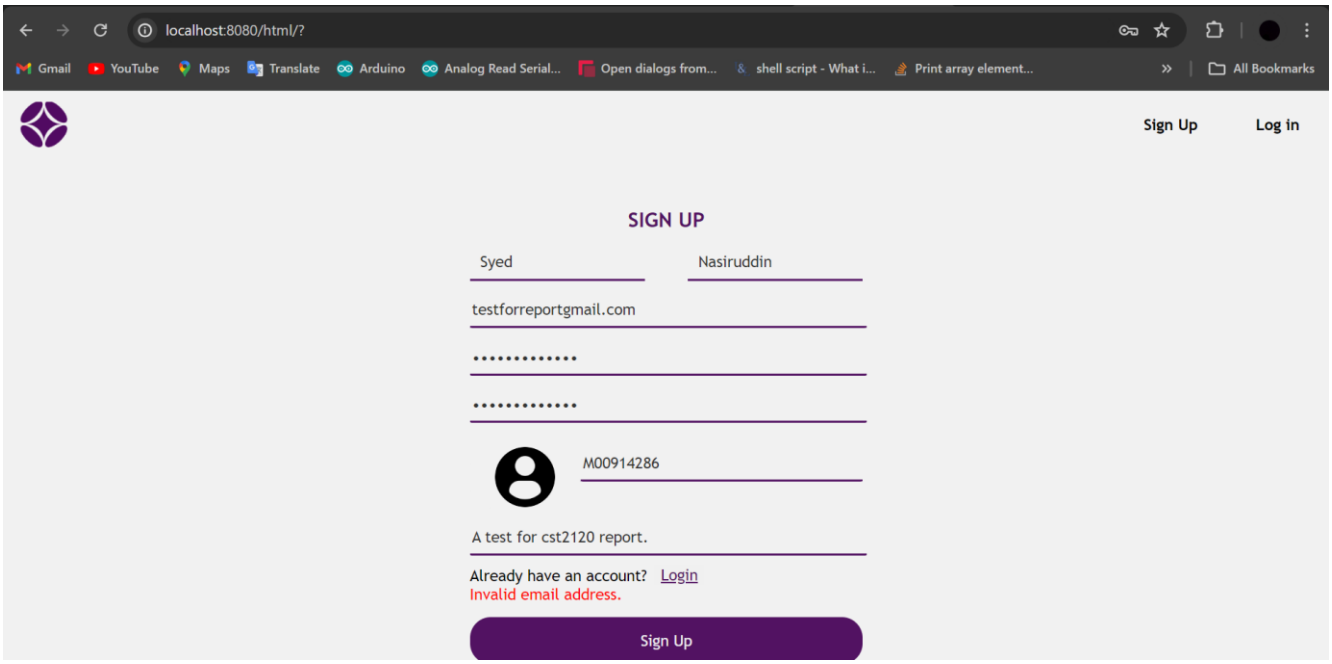


2) Special characters in the first or last name (for example, brackets []).

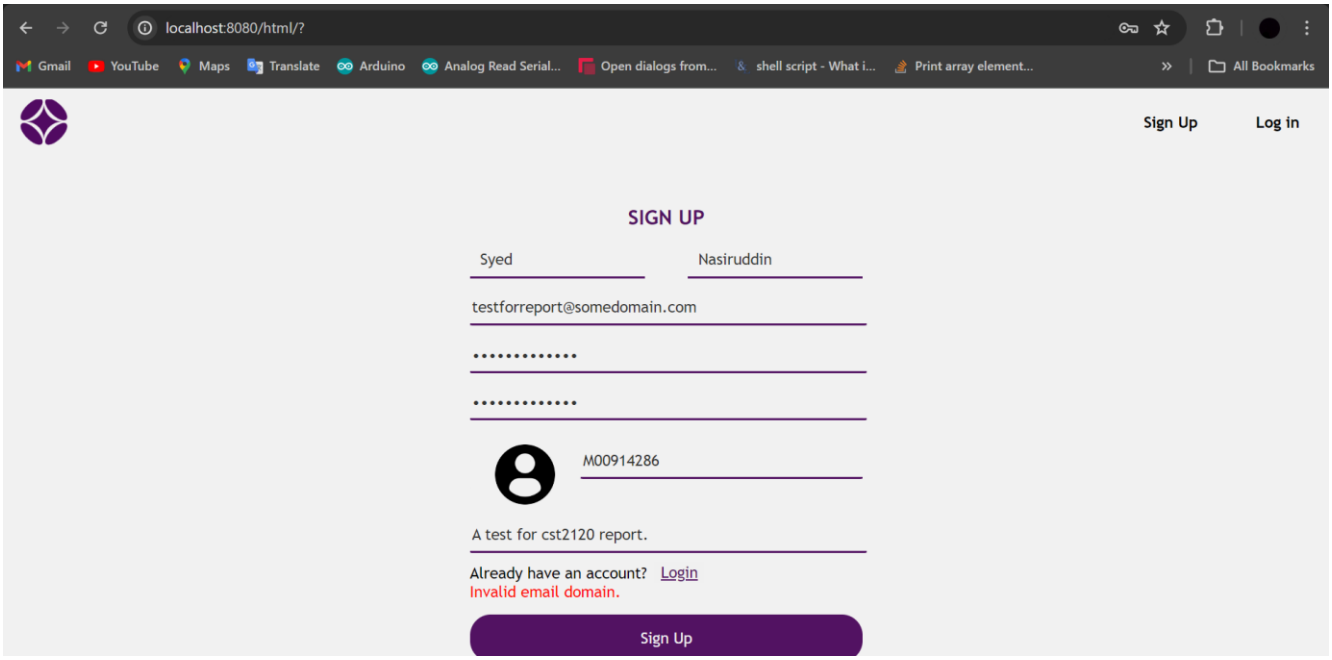




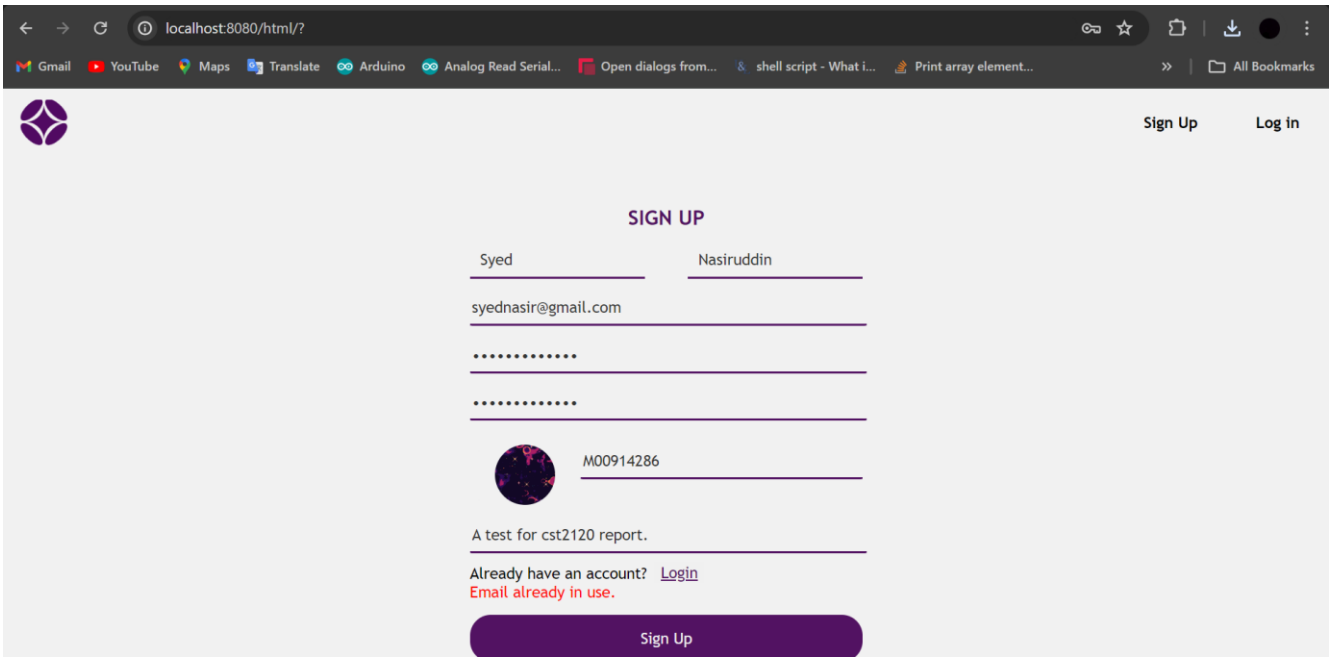
3. Invalid email address (missing @)

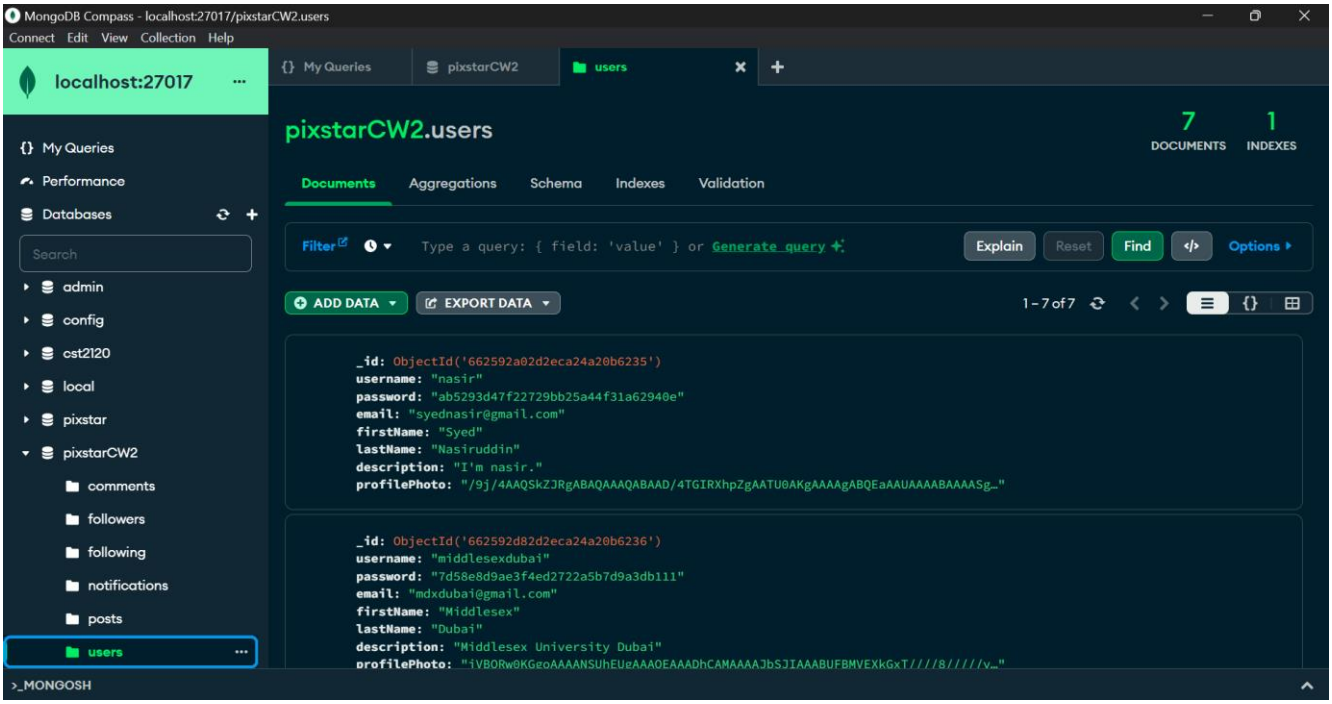


3. Invalid email domain. (not gmail, yahoo, outlook, aol, hotmail, mdx.ac.live.uk)



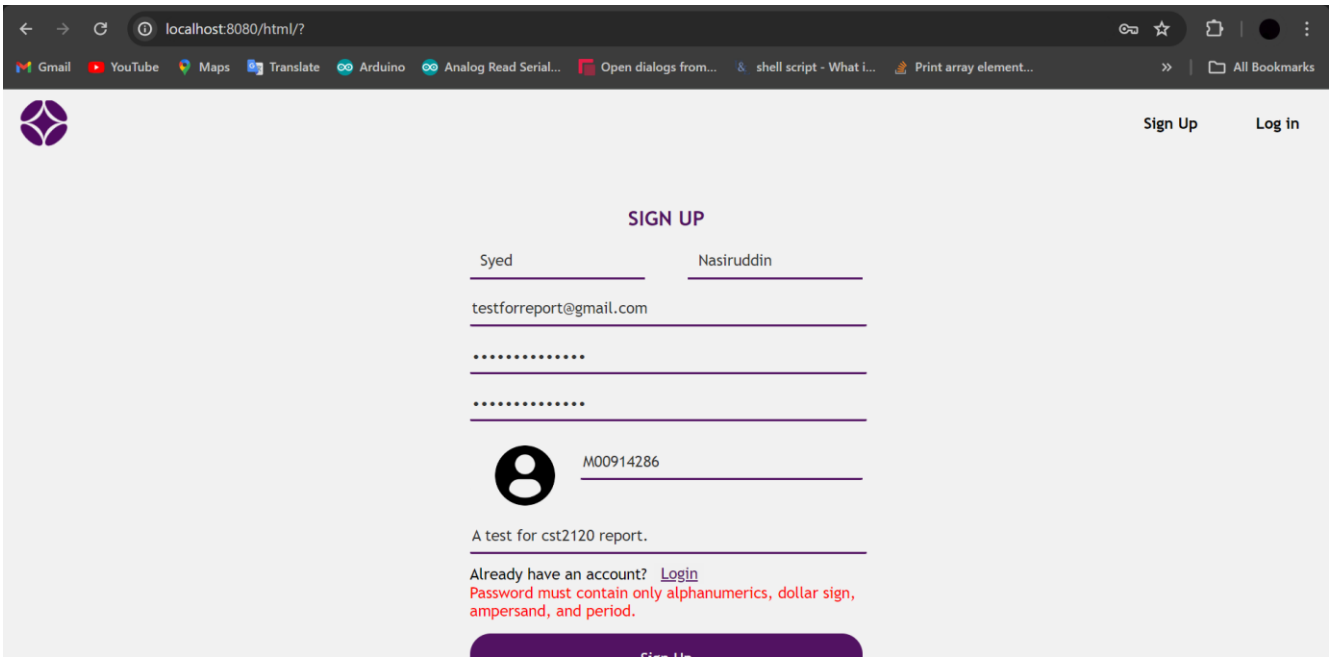
4. Email already taken.



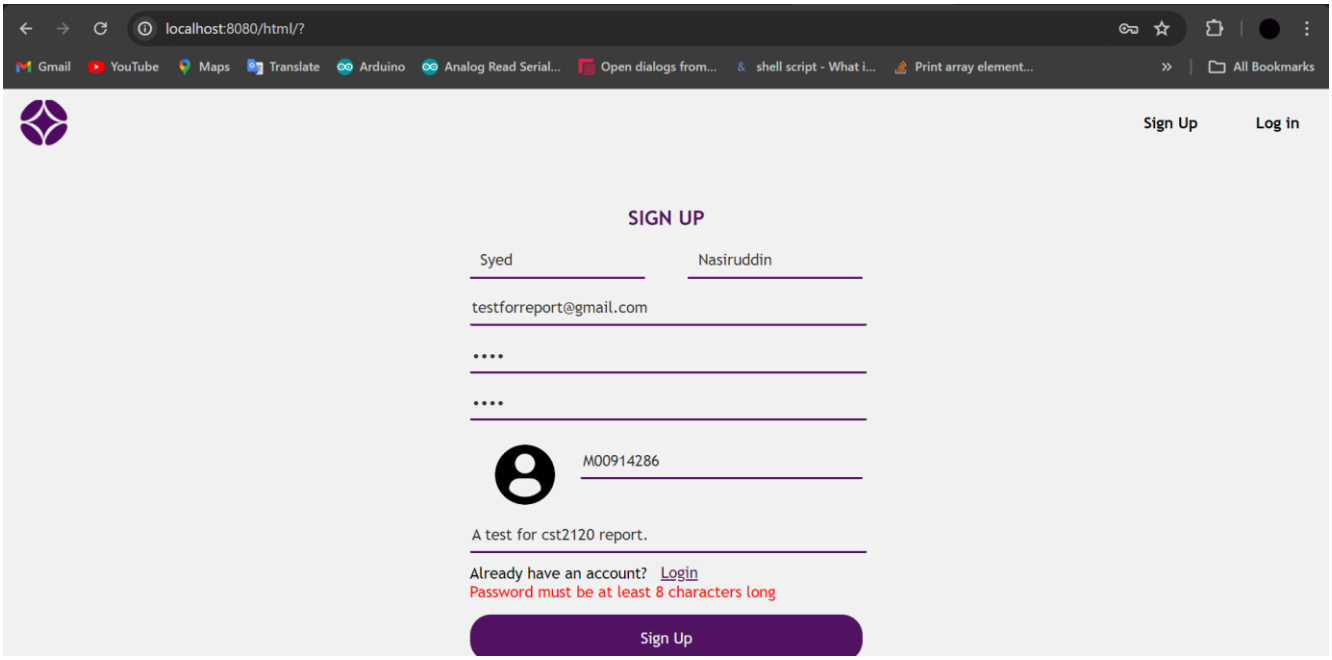


email already used by user “nasir”

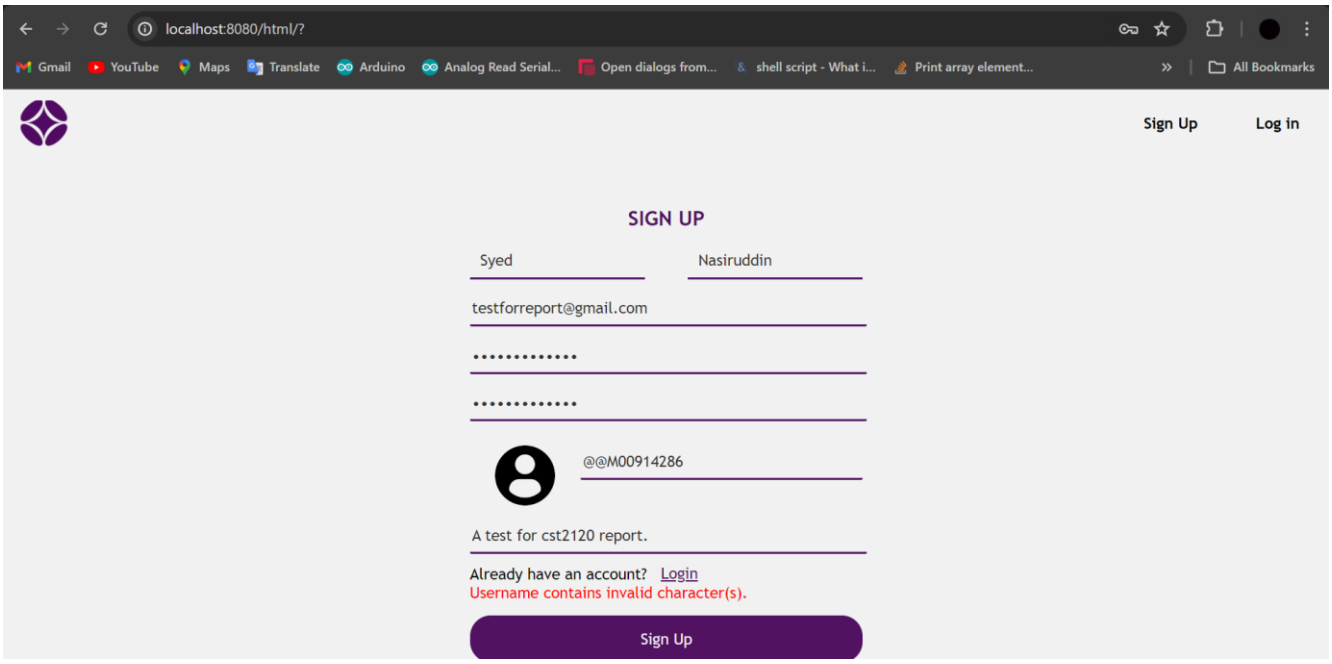
5. Password contains invalid characters. (for example, brackets [])



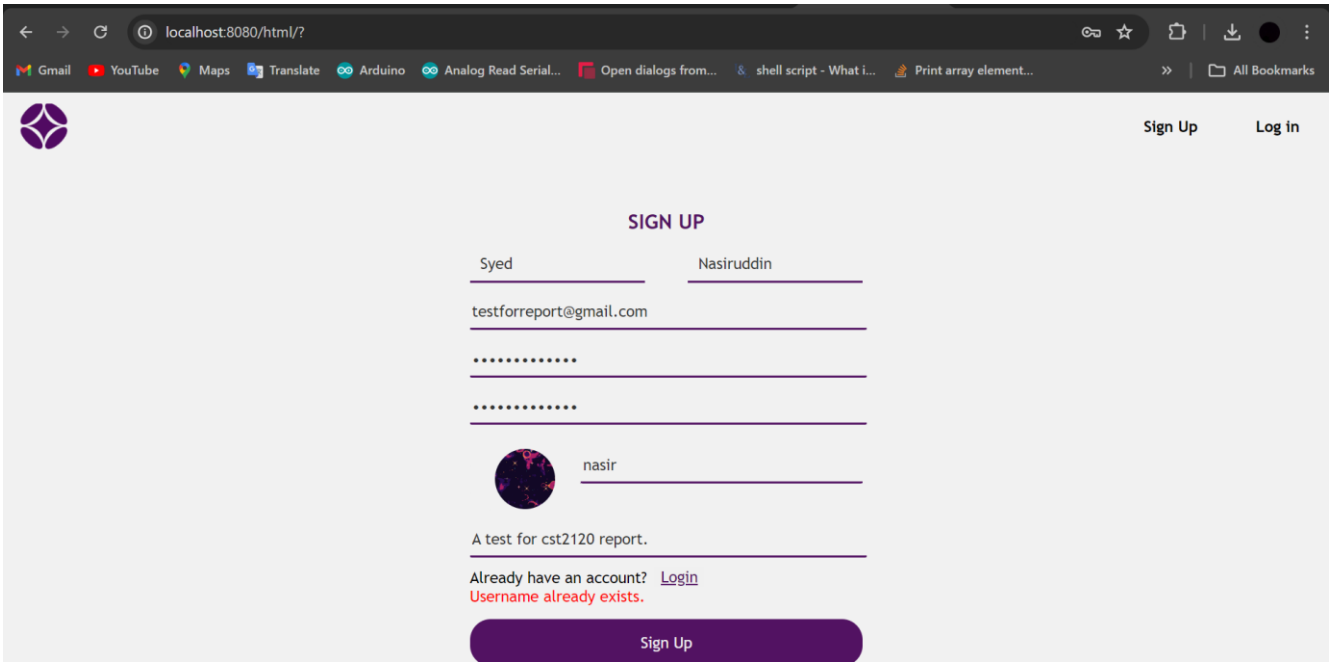
6. Password less than eight characters.

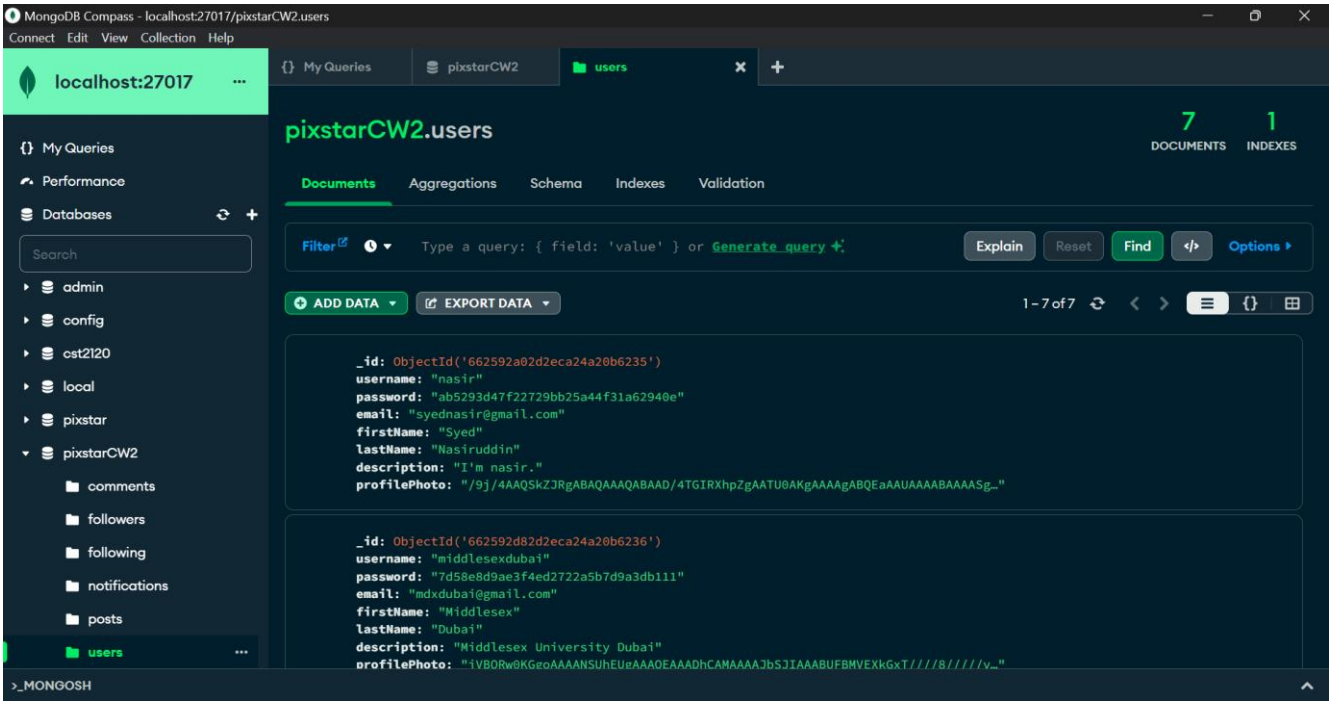


7. Username contains invalid characters.



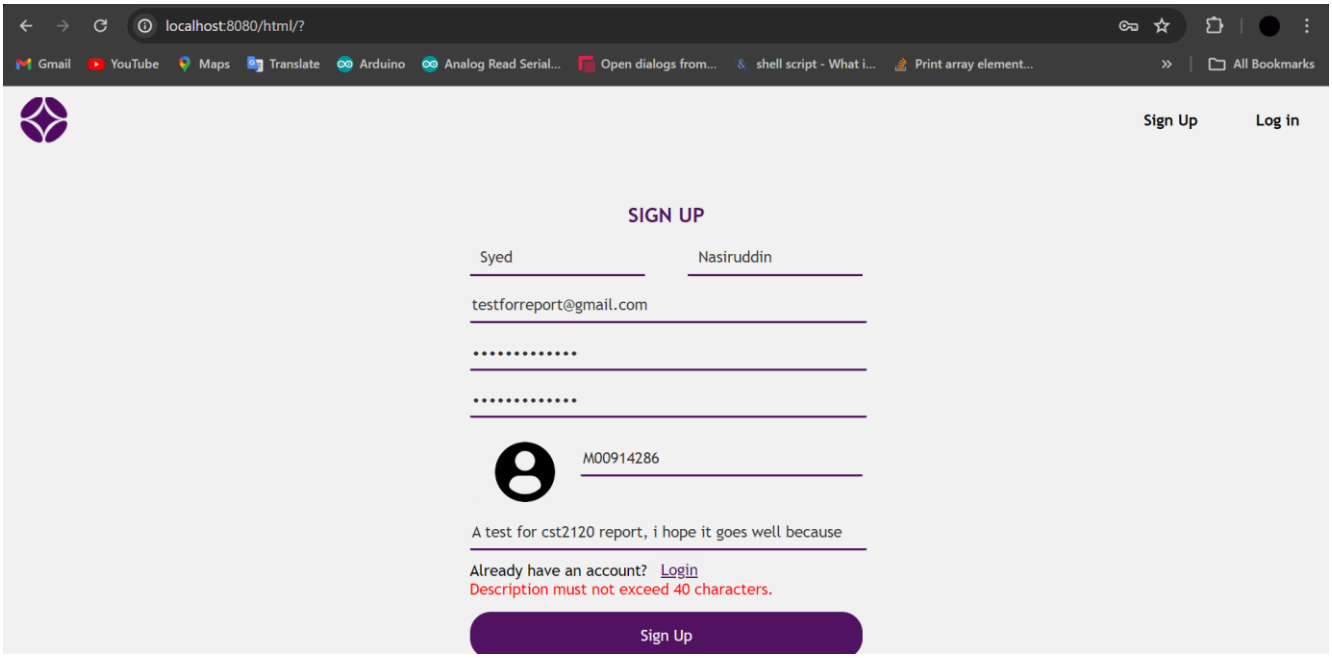
8. Username already taken.



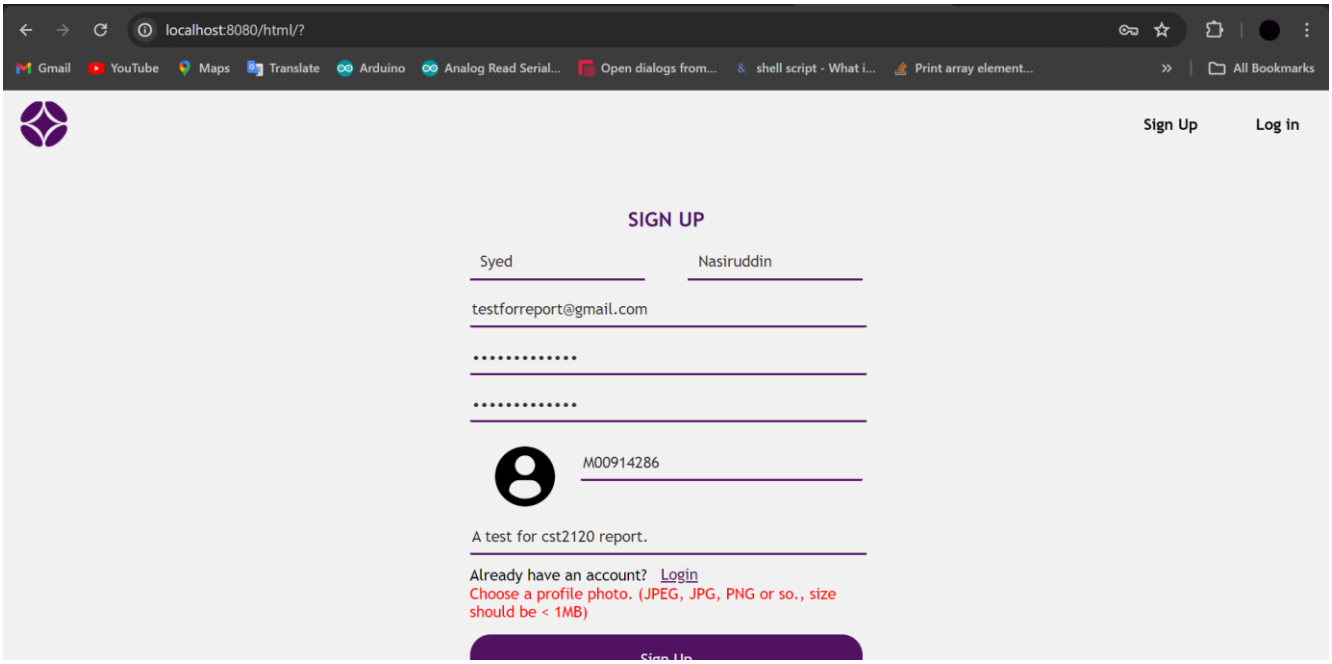


once again, the username is already taken by “nasir”.

9. Description too long.

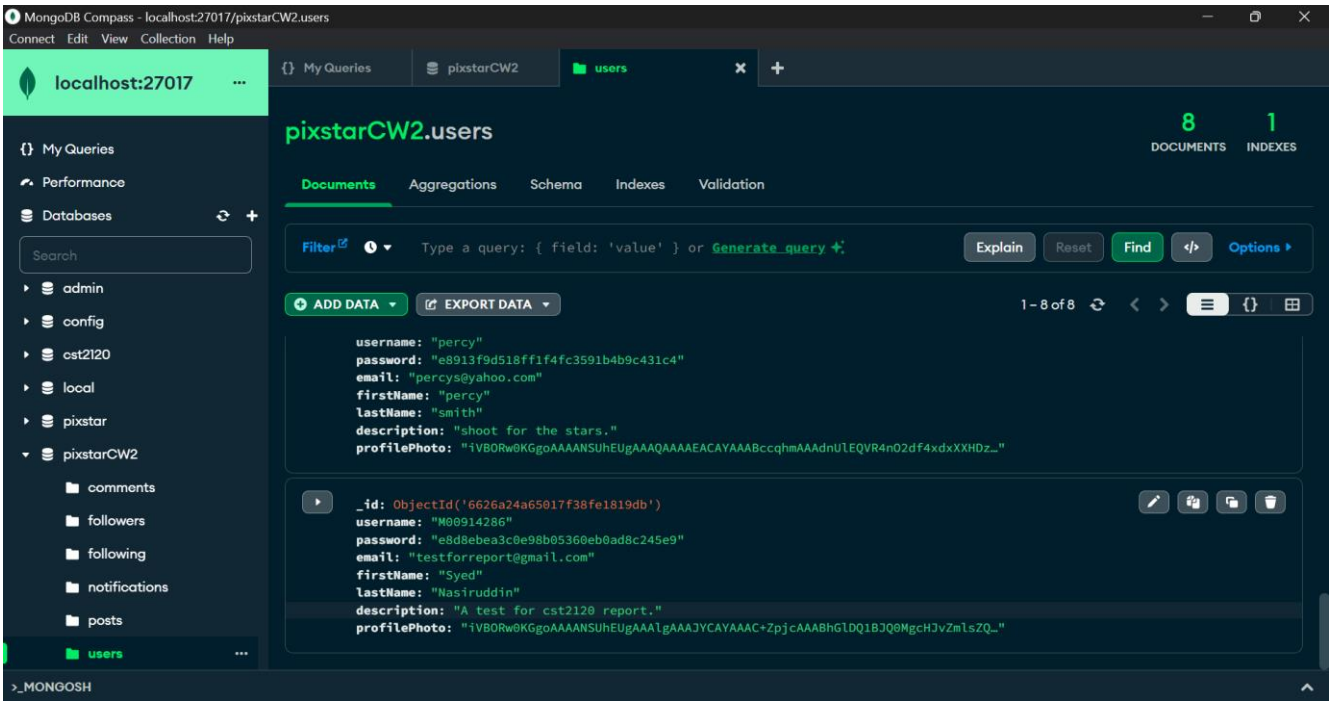


10. Profile photo not chosen.



NOTE: if a user attaches a file that is not an image or exceeds one MB, it is not set and if the user presses submit the same error is shown.

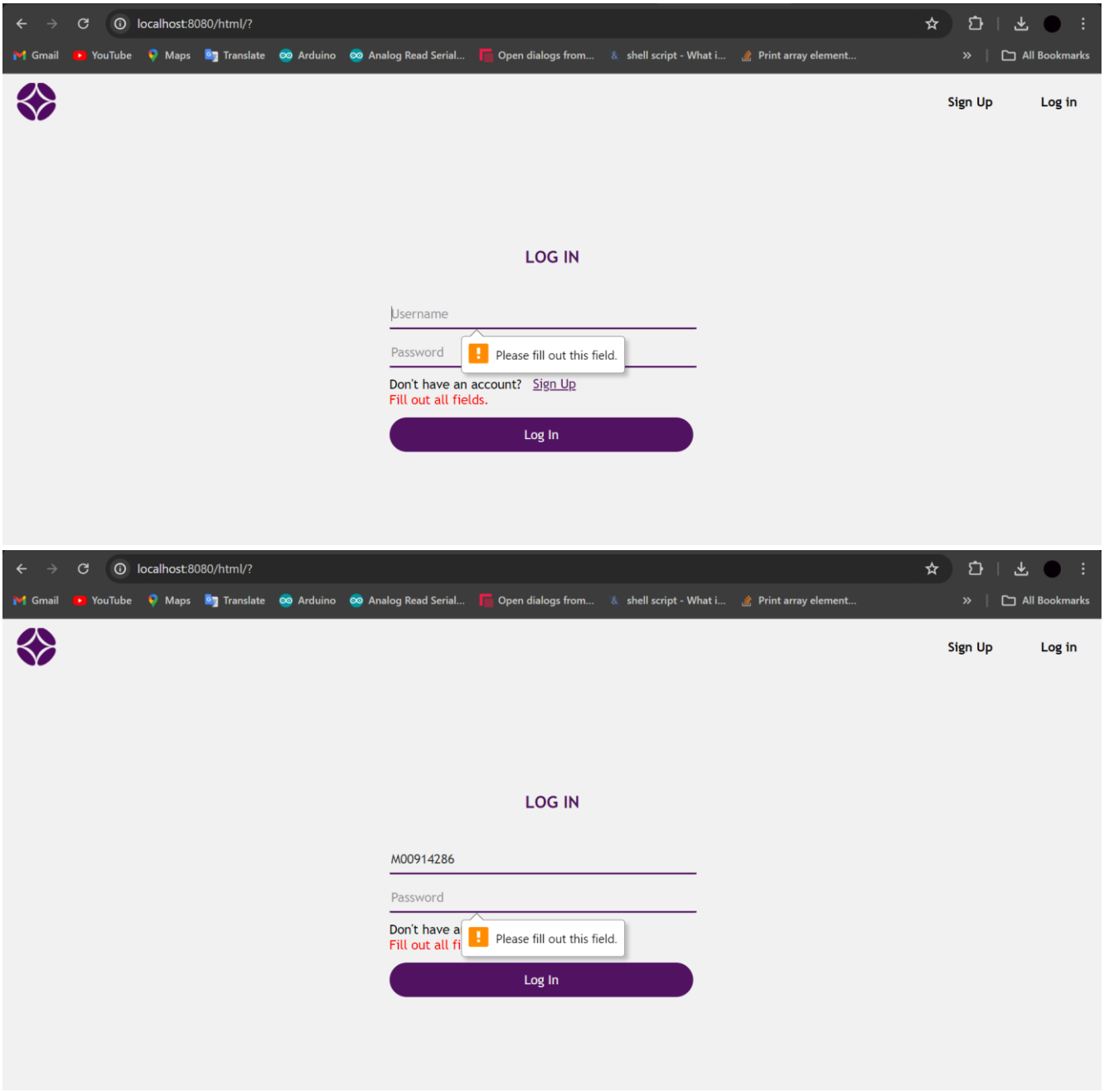
Once all the validation checks are passed, the form is submitted and the users collection is updated.



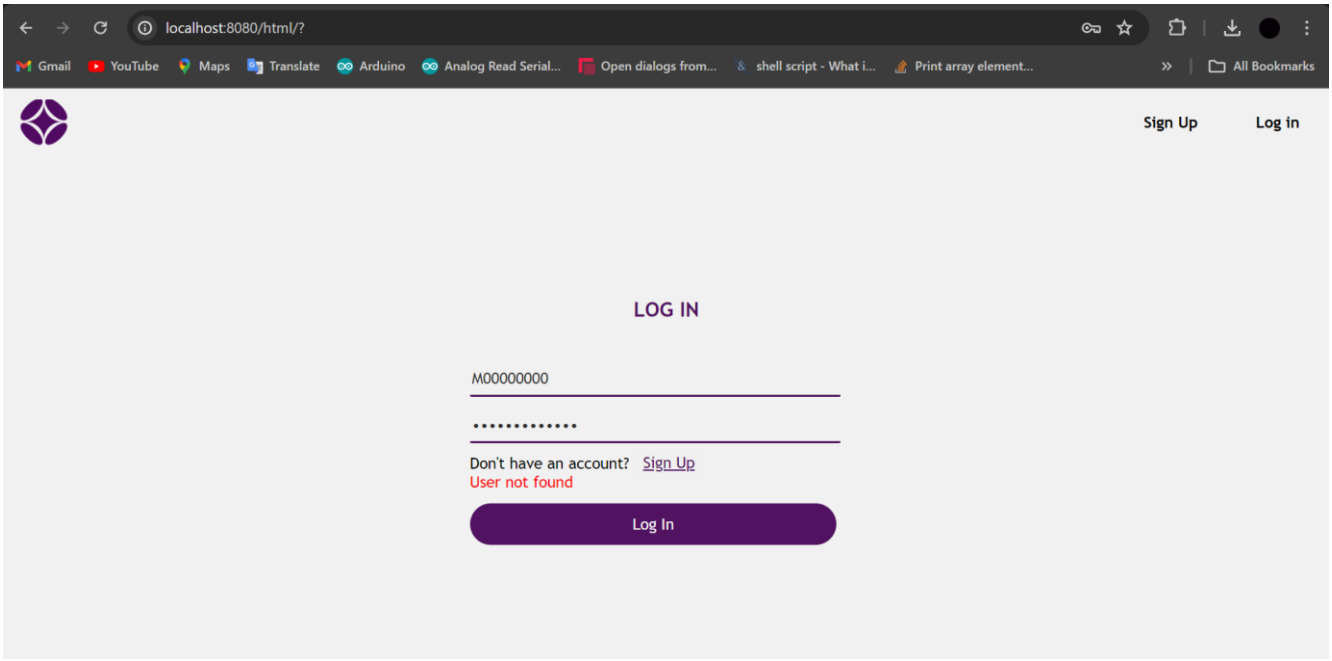
Login

The following outlines the login process, and the validation that takes place.

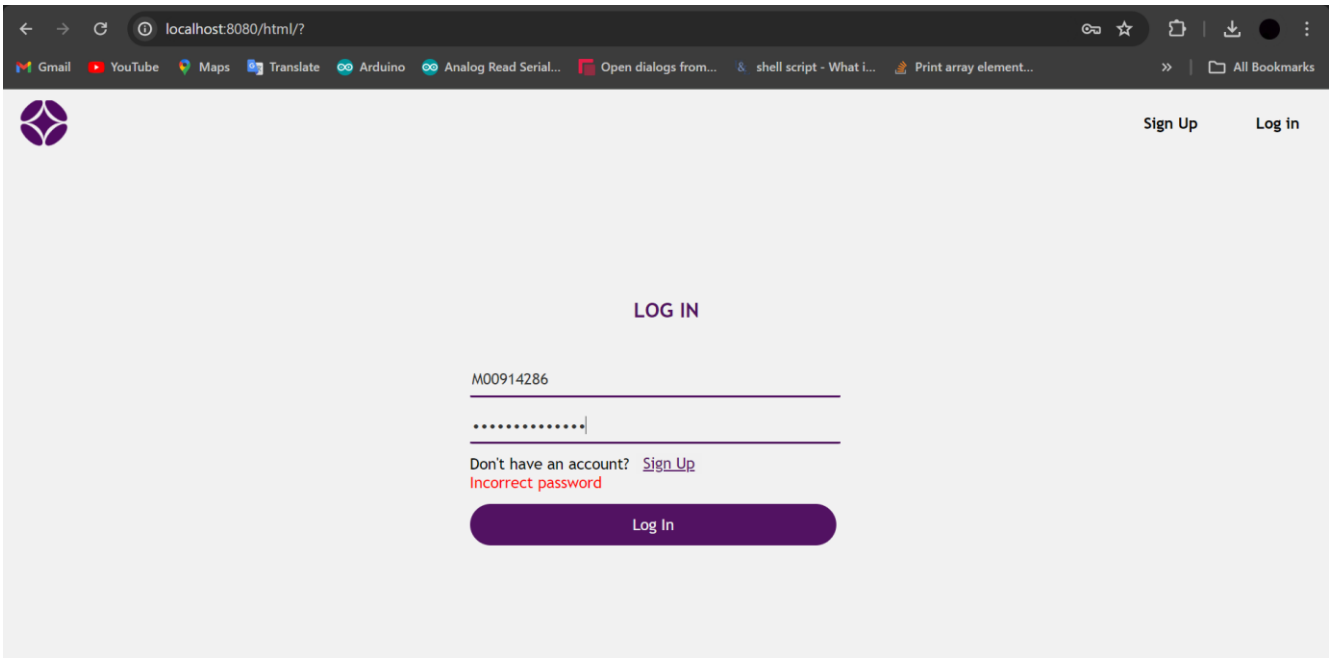
1. If all the fields are not filled out.



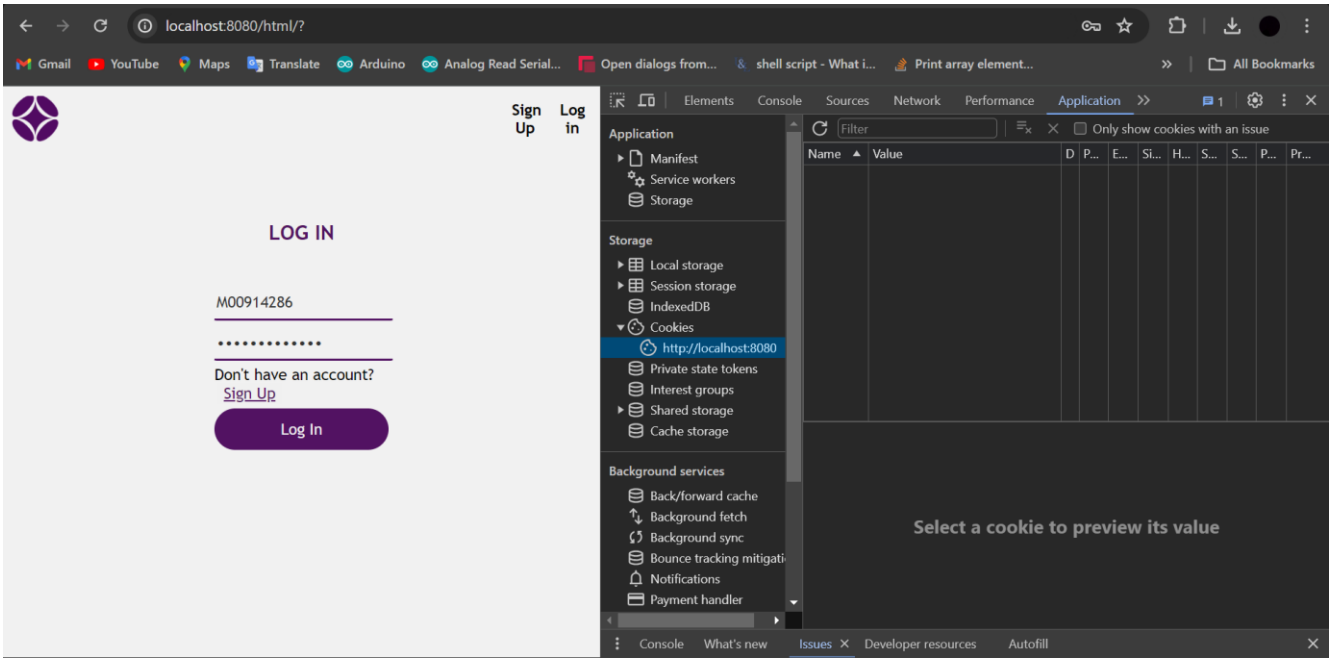
2. If the user doesn’t exist.



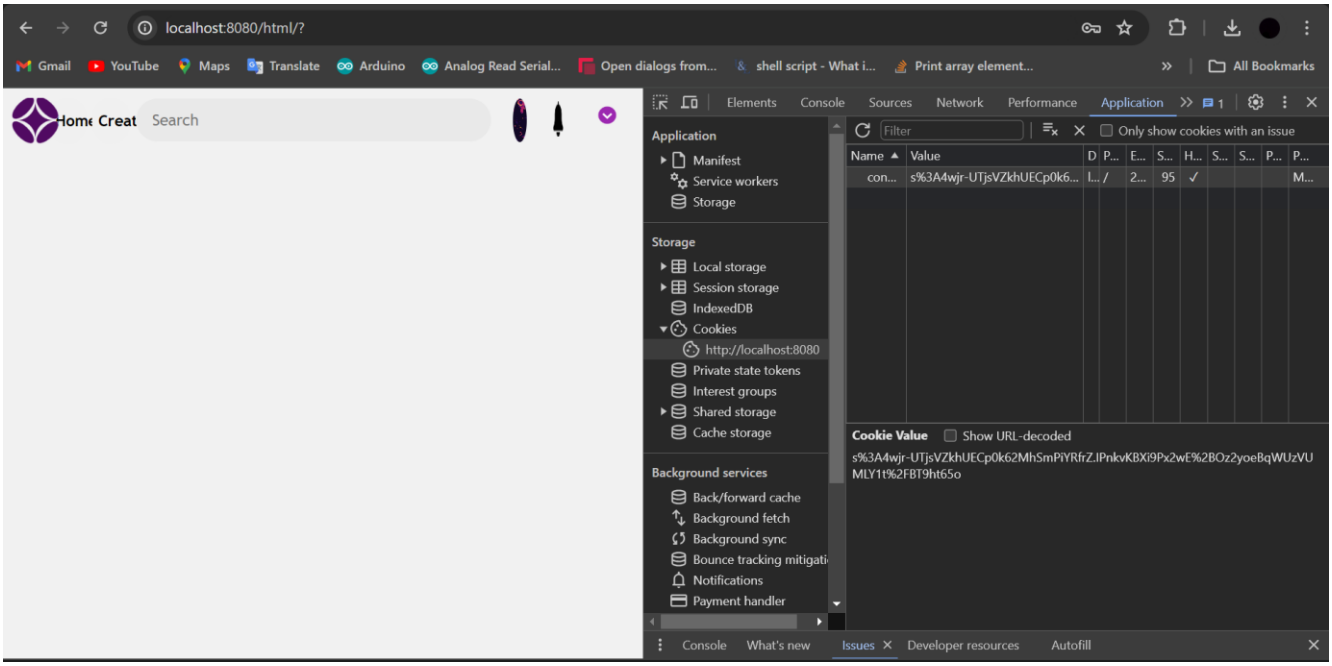
3. If passwords don't match.



Else the user will be logged and a session is created in which the user's username is stored.



NOTE: prior to logging in, no session or cookie is created.

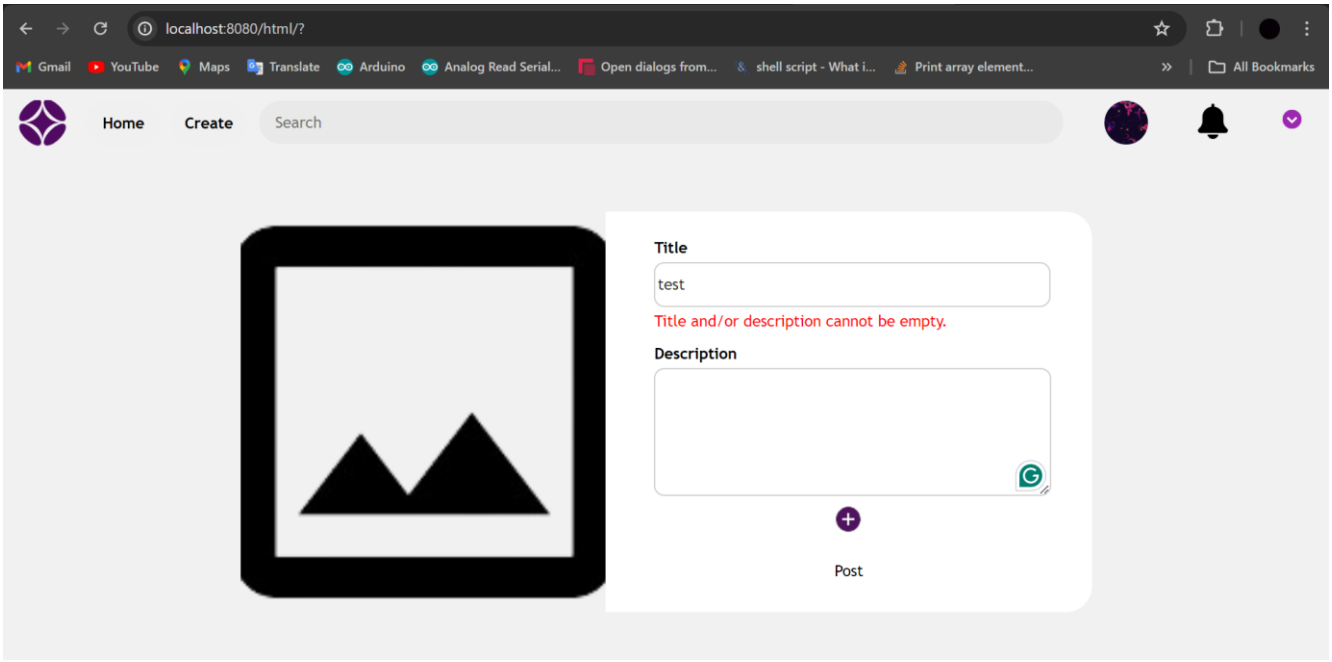


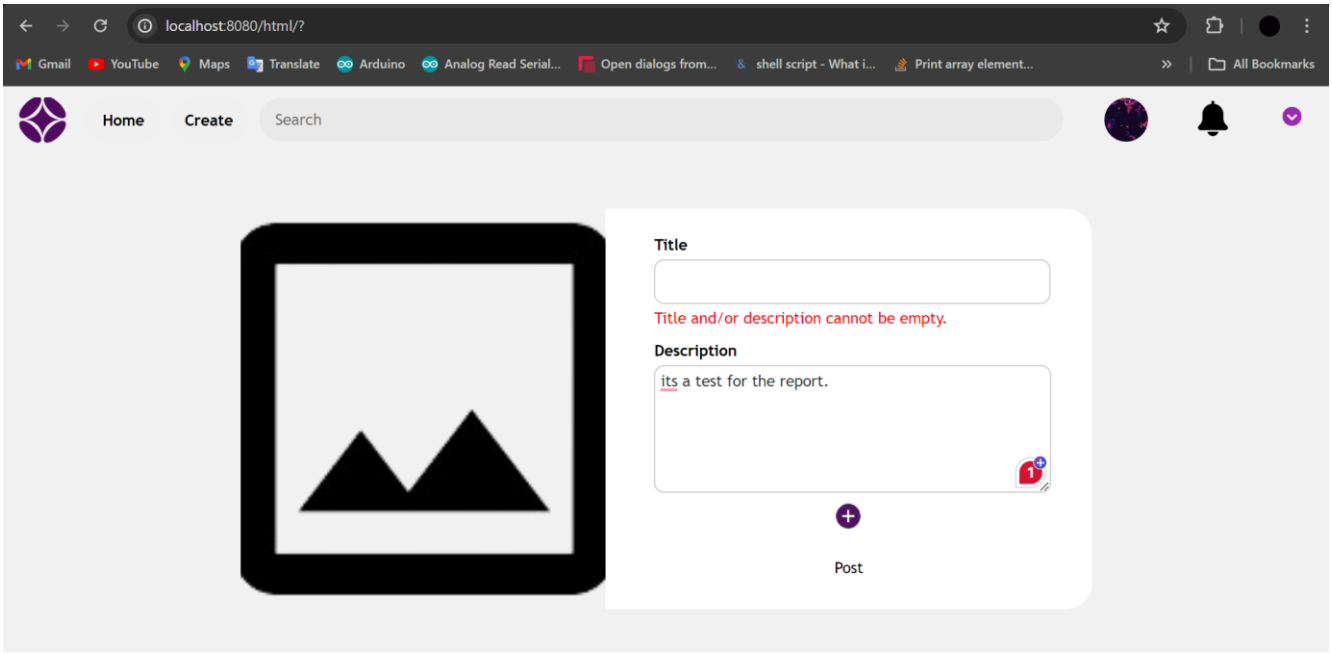
When the user logs in, the session and a cookie is made.

Create Post

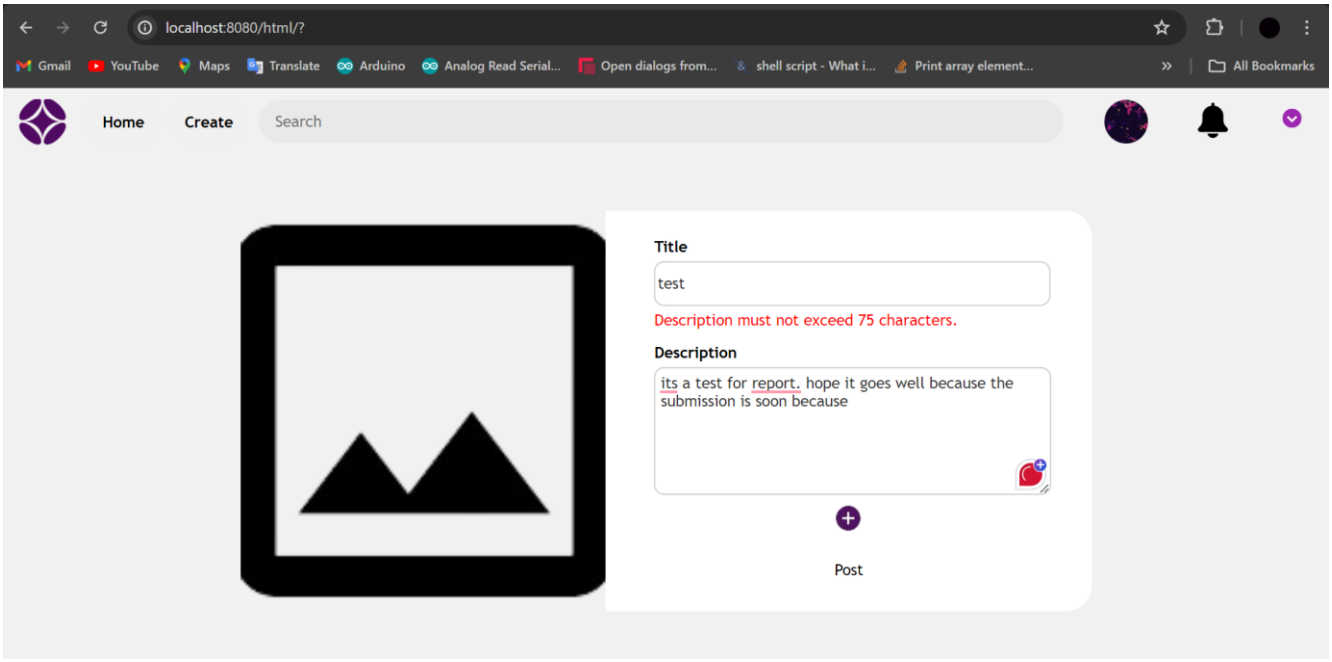
The following outlines the post creation process and the validation that takes place.

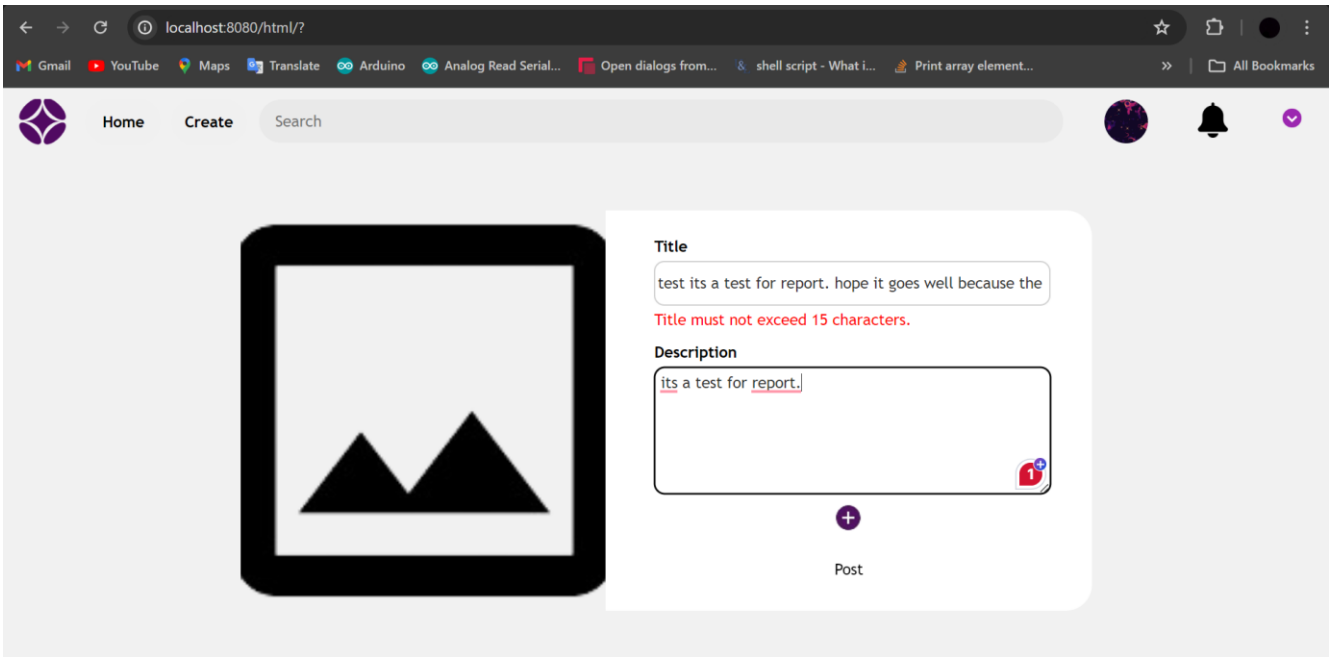
1) If all the fields are not filled out.



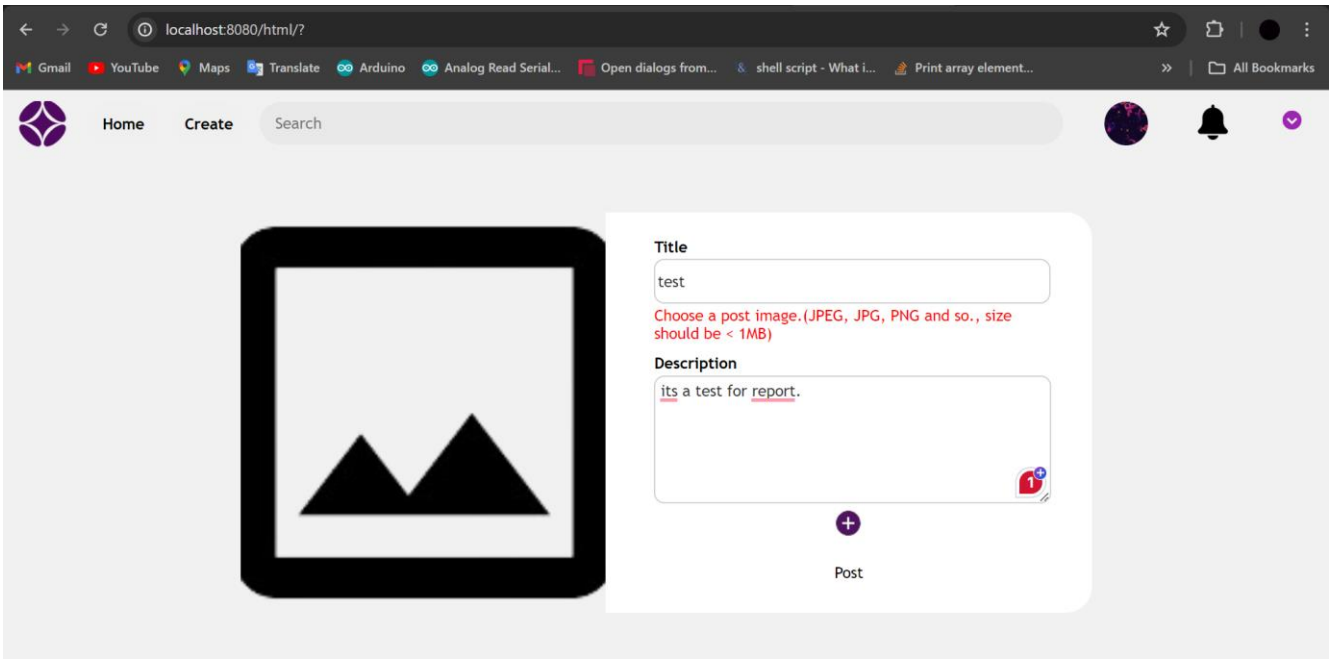


2) if any of the fields exceed the character limit.

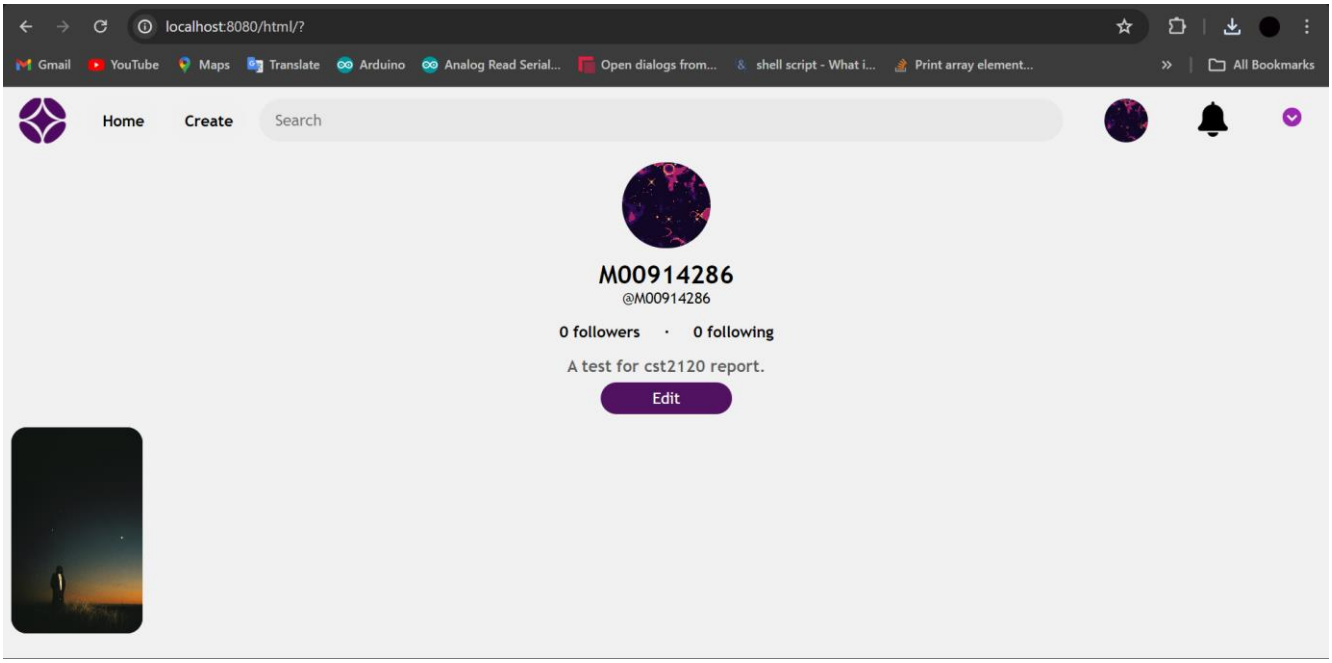


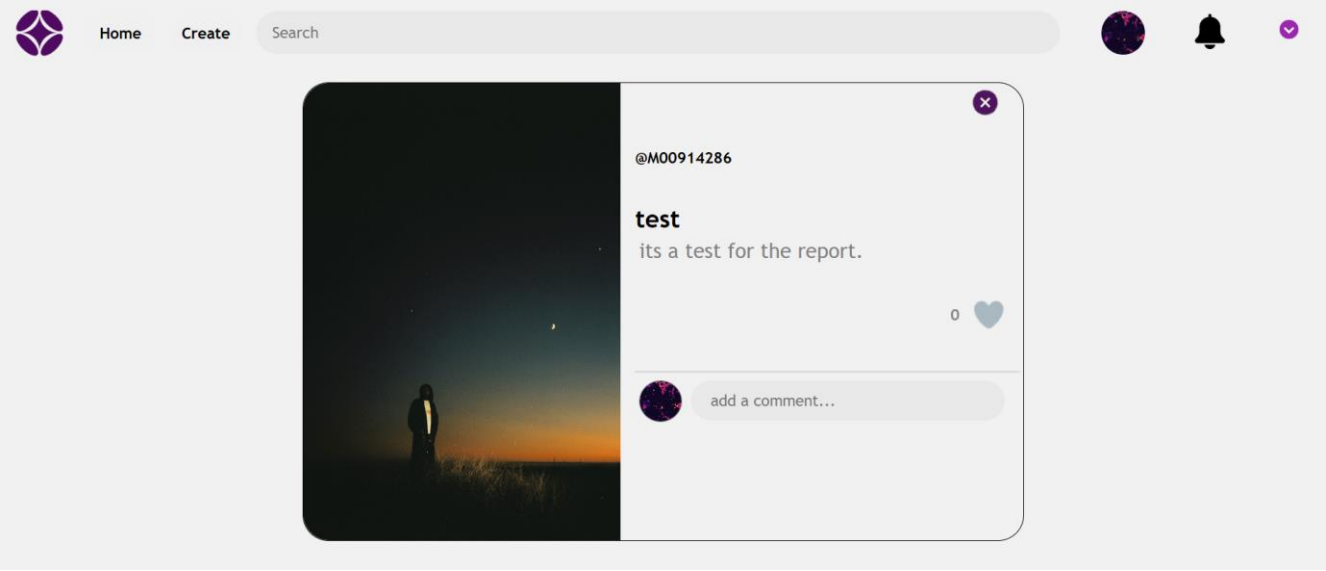


3. If a post image is not chosen.

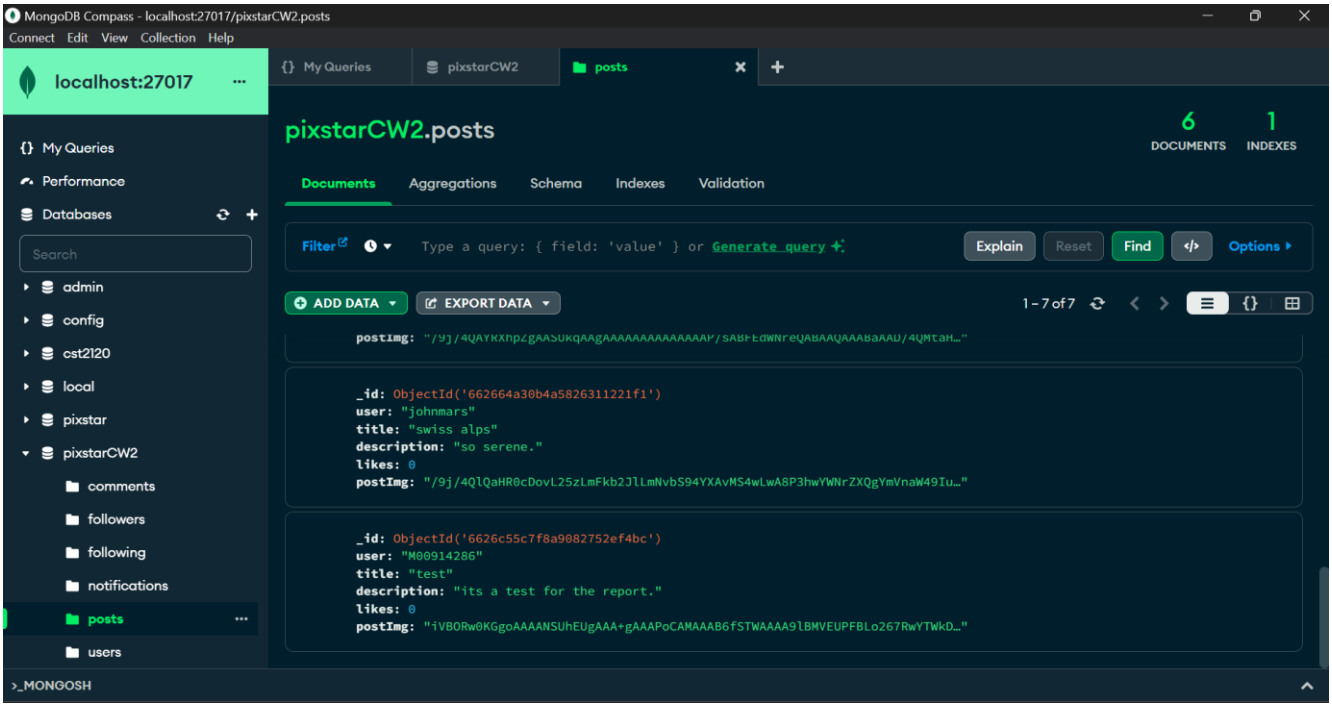


If all the validation checks are passed, the post is made and the user can view it in their account page.

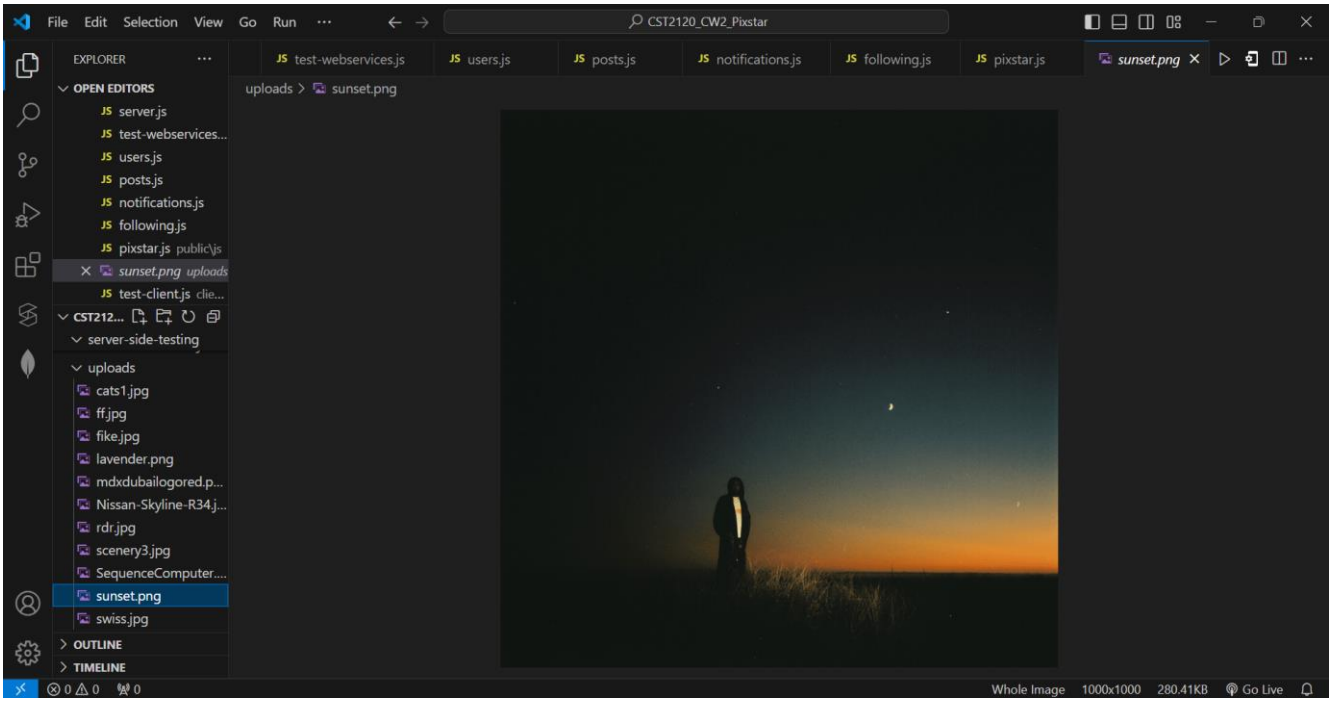




A document is made with the new posts details.

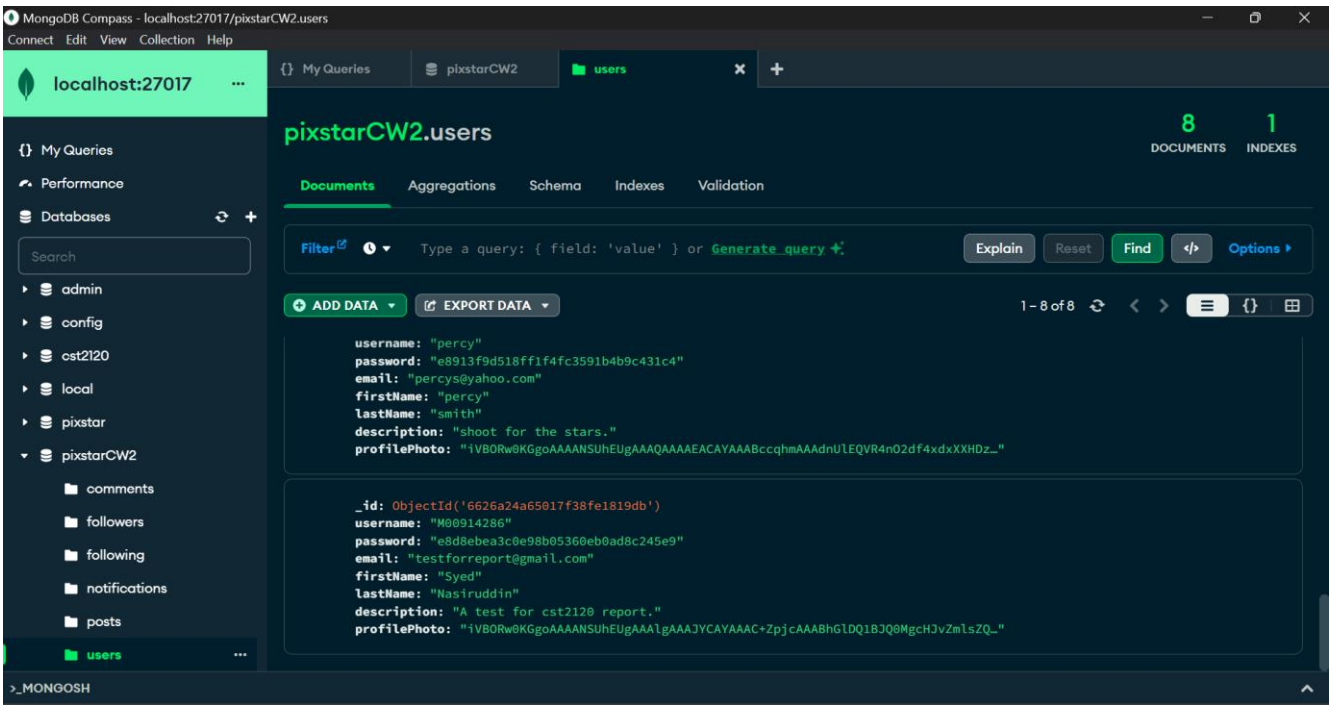


The post photo is uploaded to the “uploads” file in the project.

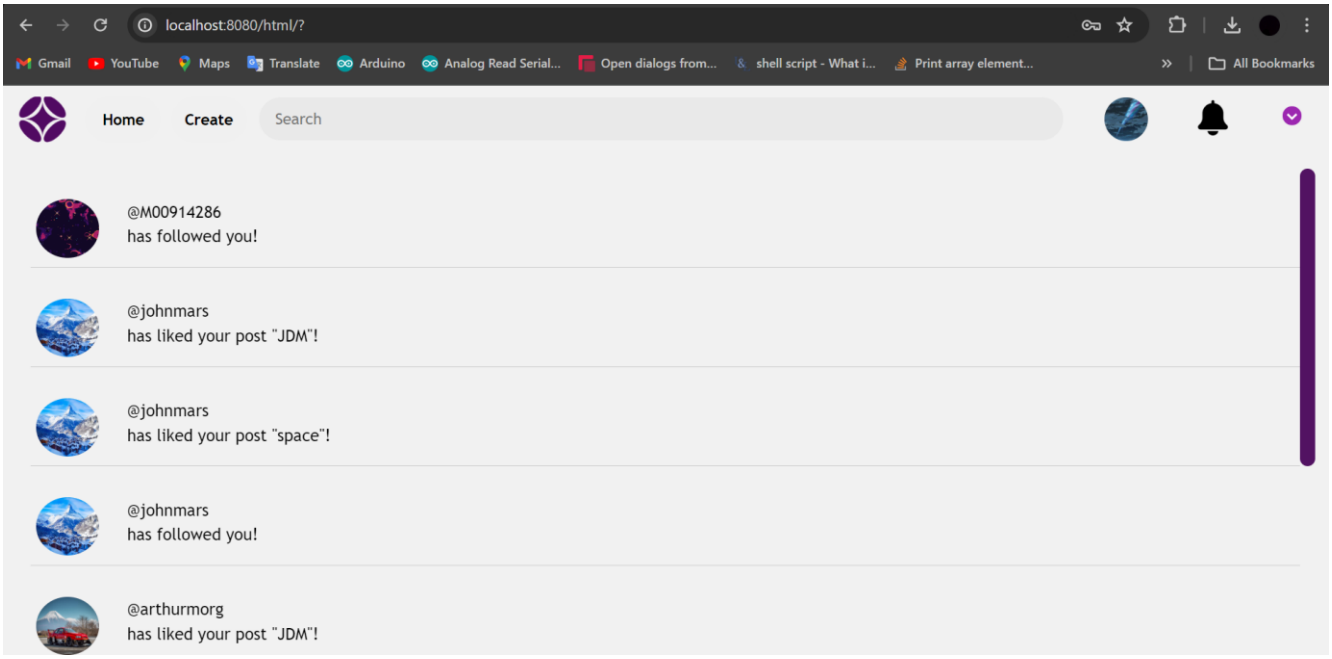


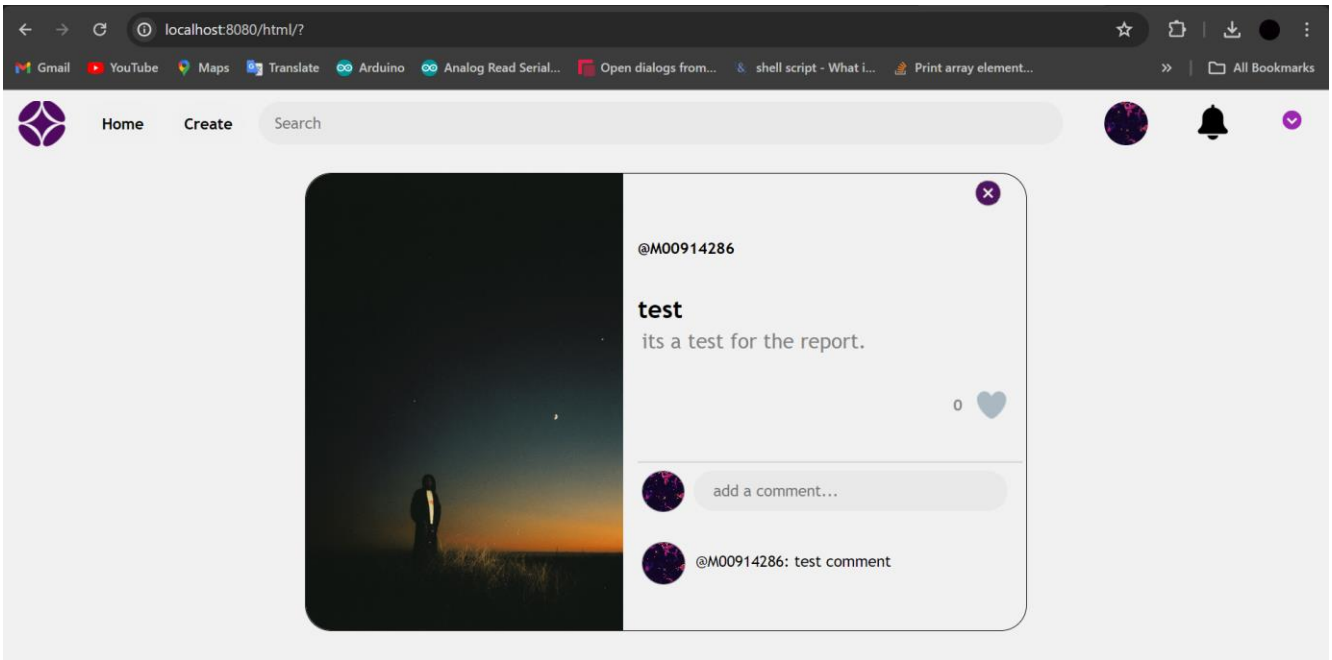
Edit Profile

The following outlines the profile edit process and the validation that takes place.



Above is the document in the “users” collection for the “M00914286” user, while updating the profile photo the description and stored profile photo will be changed.

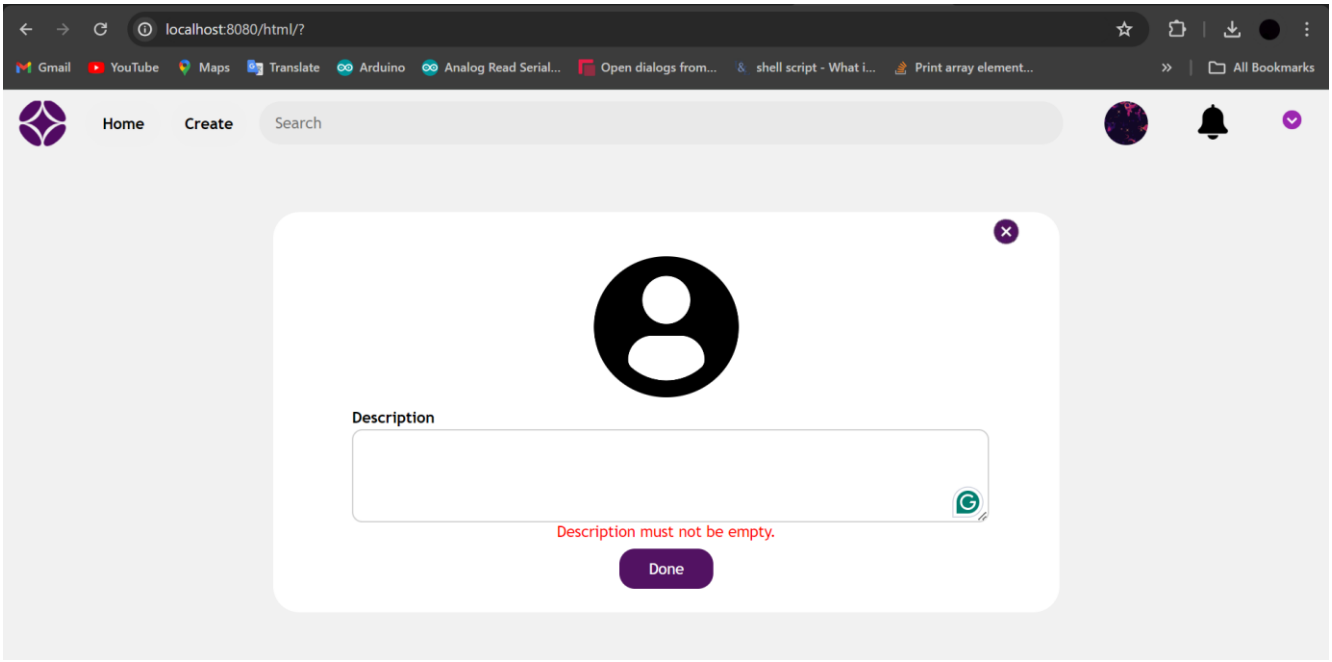




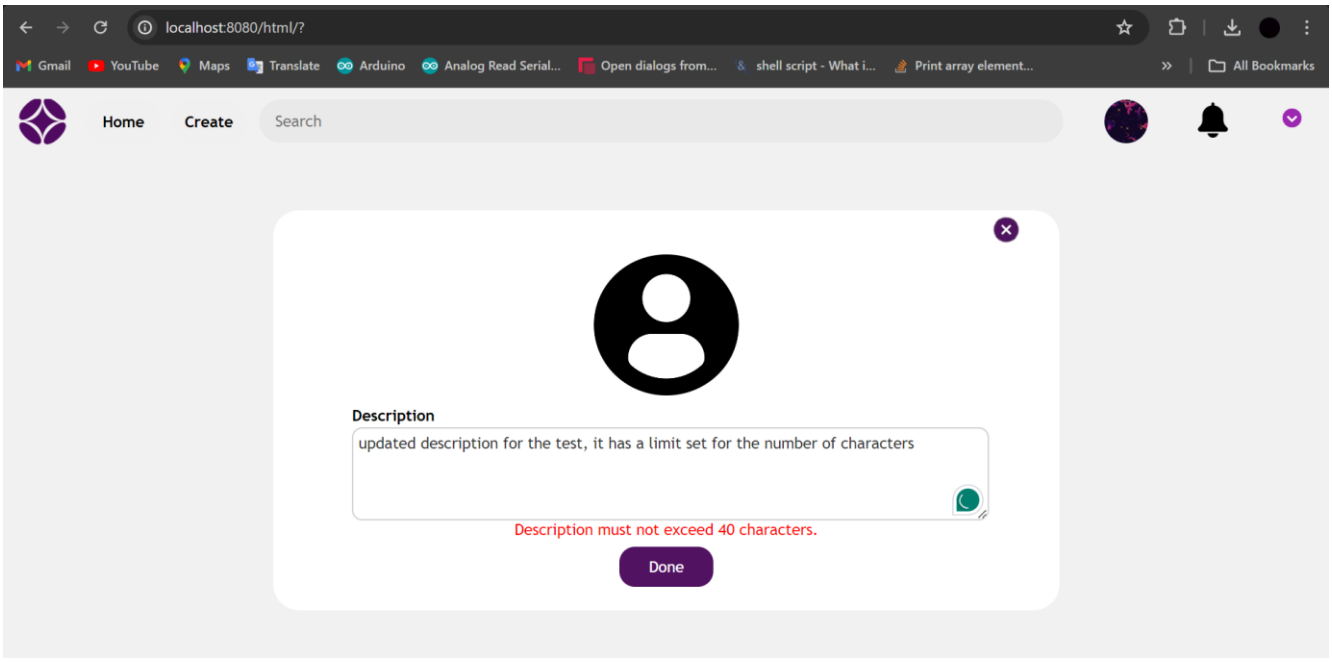
When a user changes their profile photo, all notifcations and comments that

include the user are updated with the new profile photo. The above is before editing the profile.

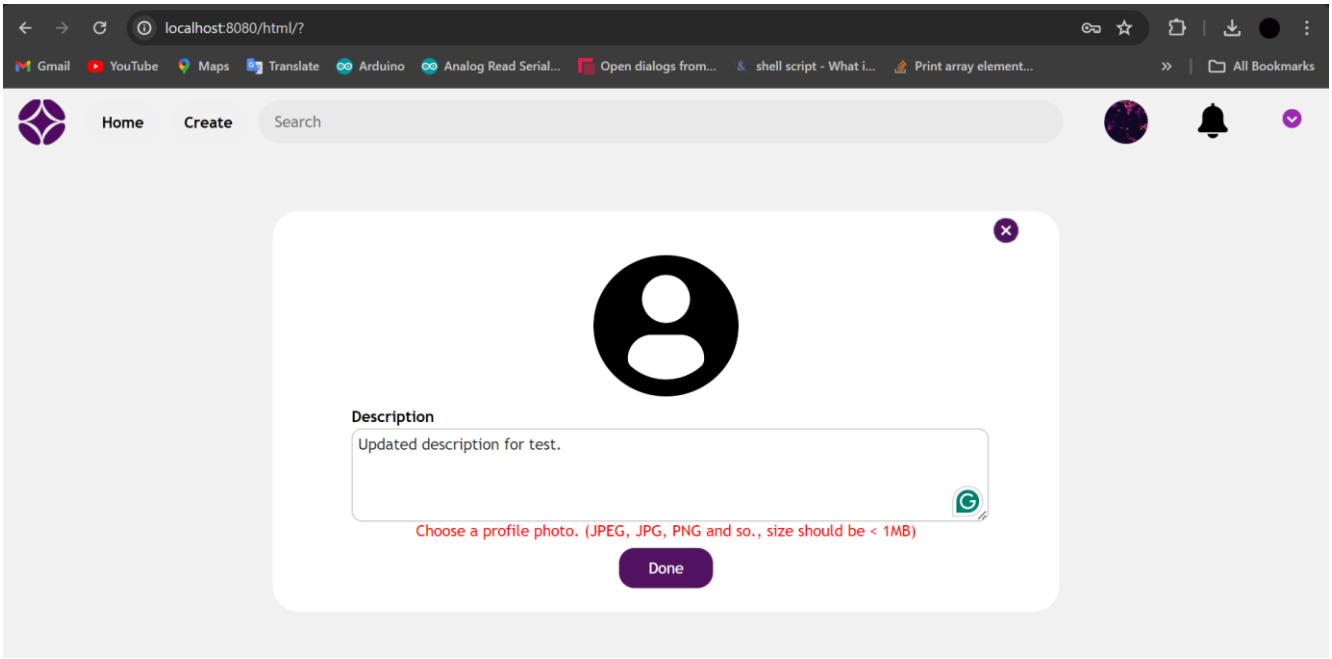
1) If the field is empty.



2) If the character limit has exceeded.

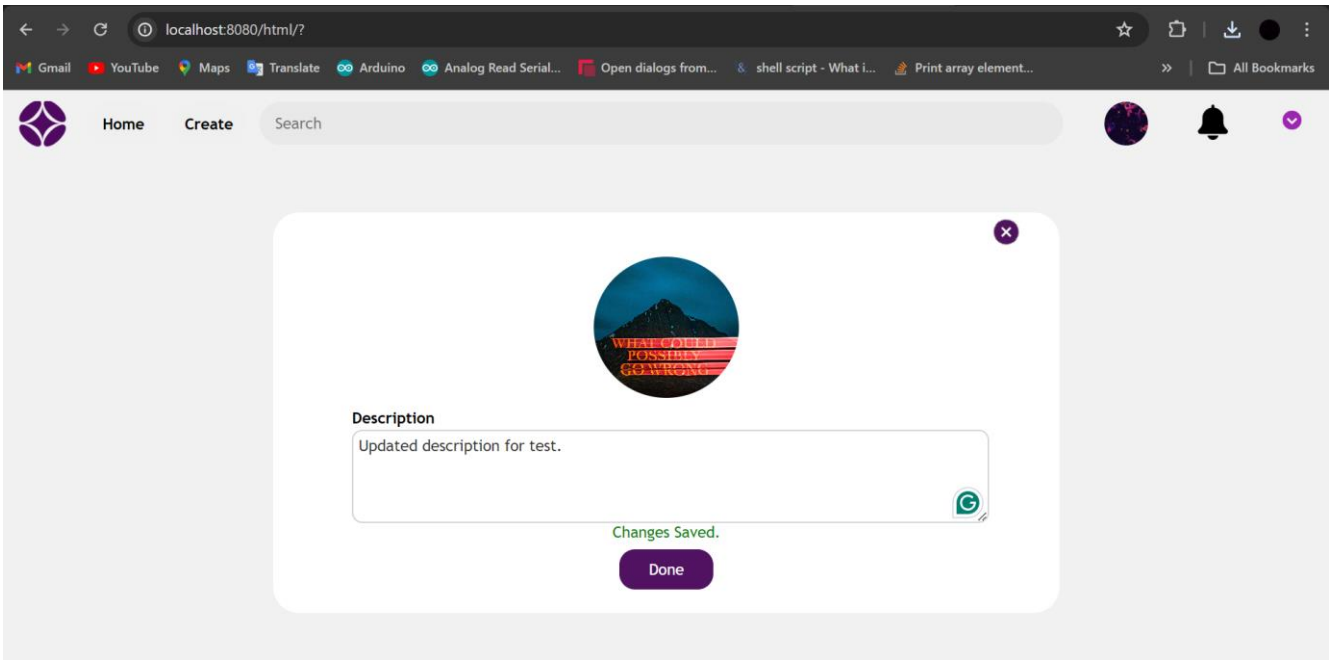


3. Profile photo not chosen.

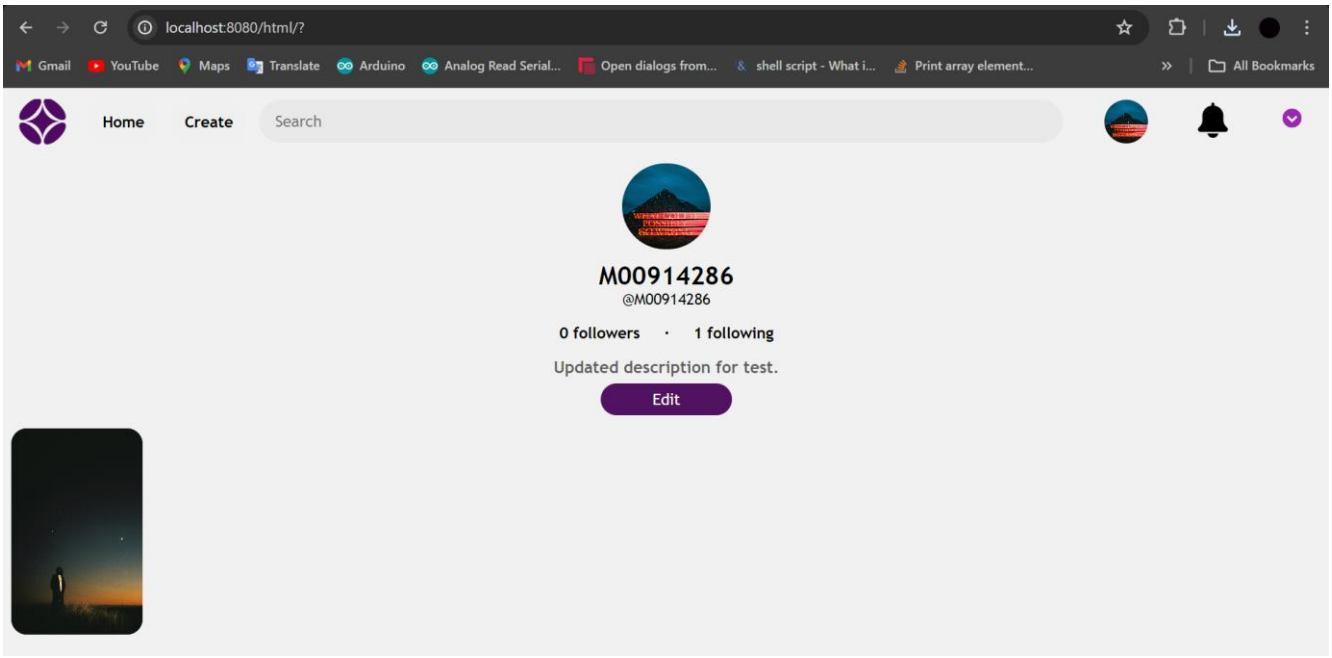


NOTE: if a user attaches a file that is not an image or exceeds one MB, it is not set and if the user presses submit the same error is shown.

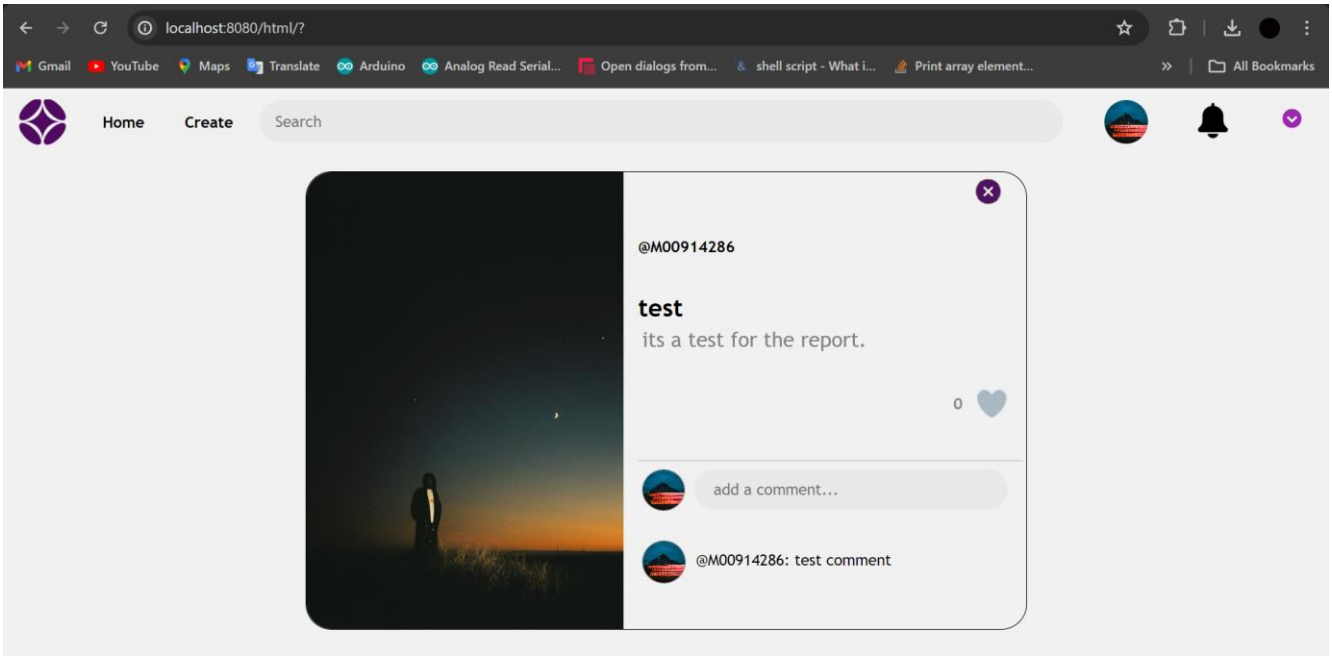
If all validation test are passed the user can change their profile photo and description successfully, any comments or notifications that included the user are updated with the new profile photo and can be seen below.



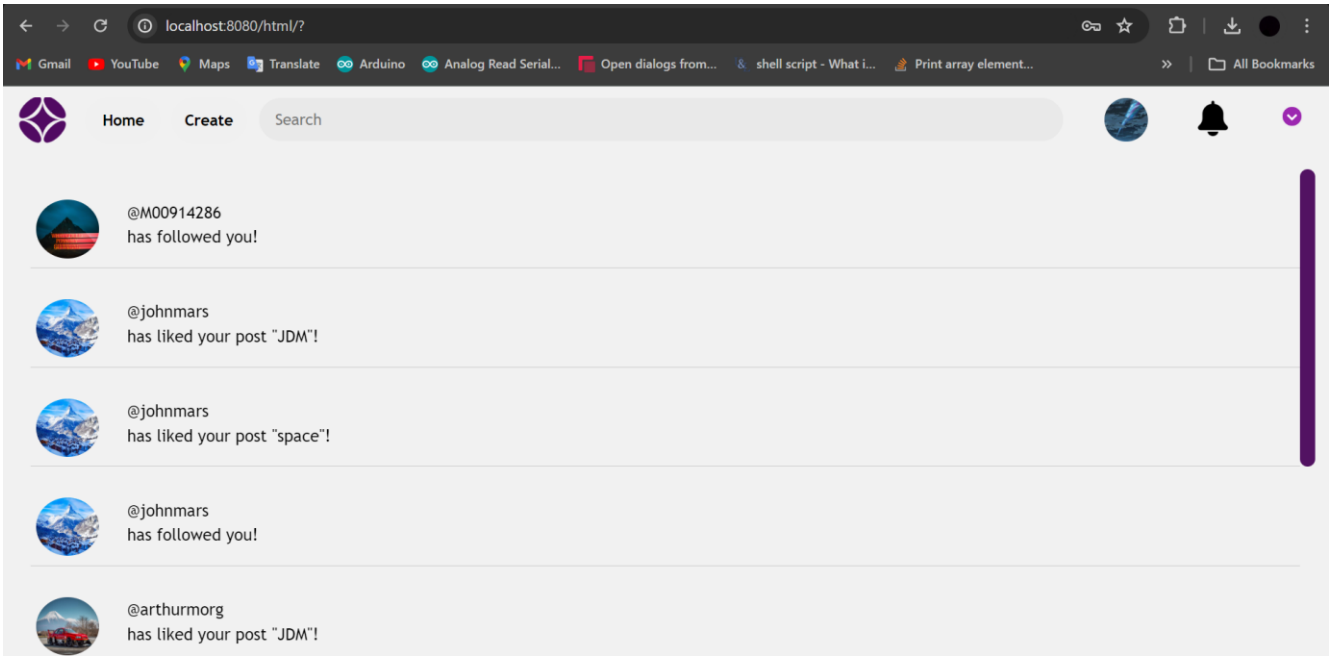
The user can then exit the section by clicking the purple “x” in the top right corner or edit the profile again.



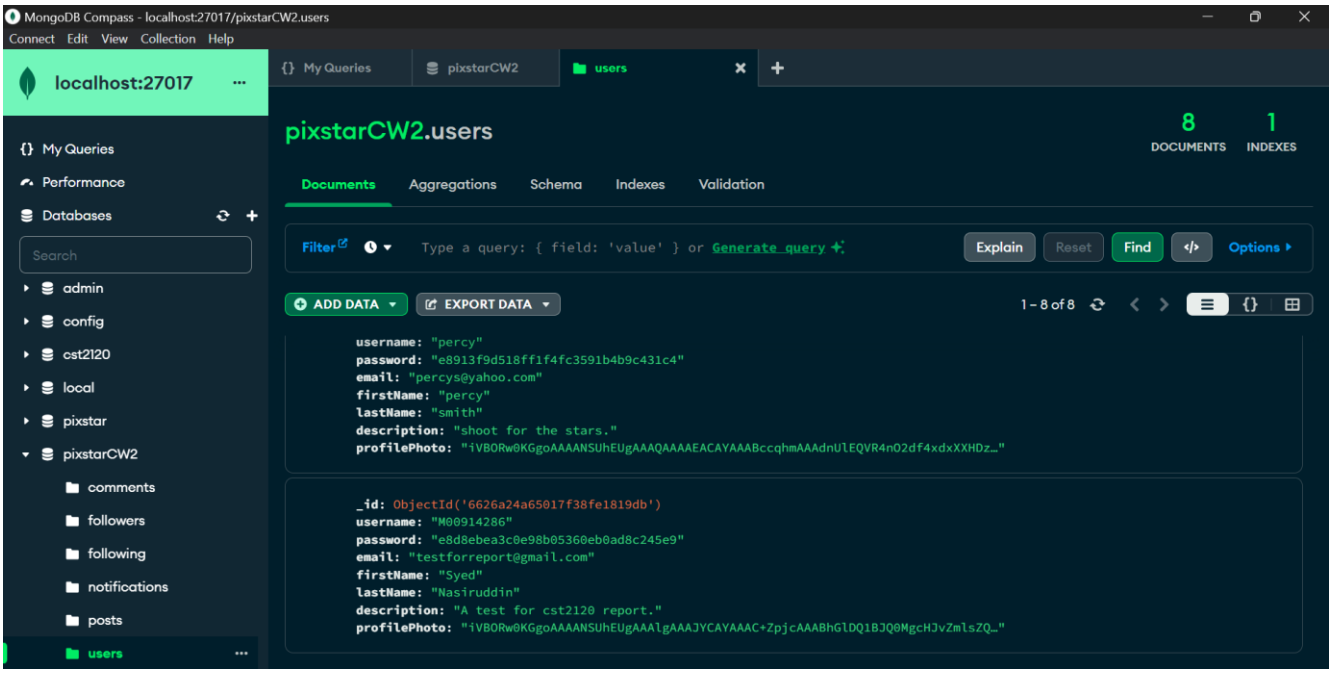
The comment left by the user is updated with the new profile photo.



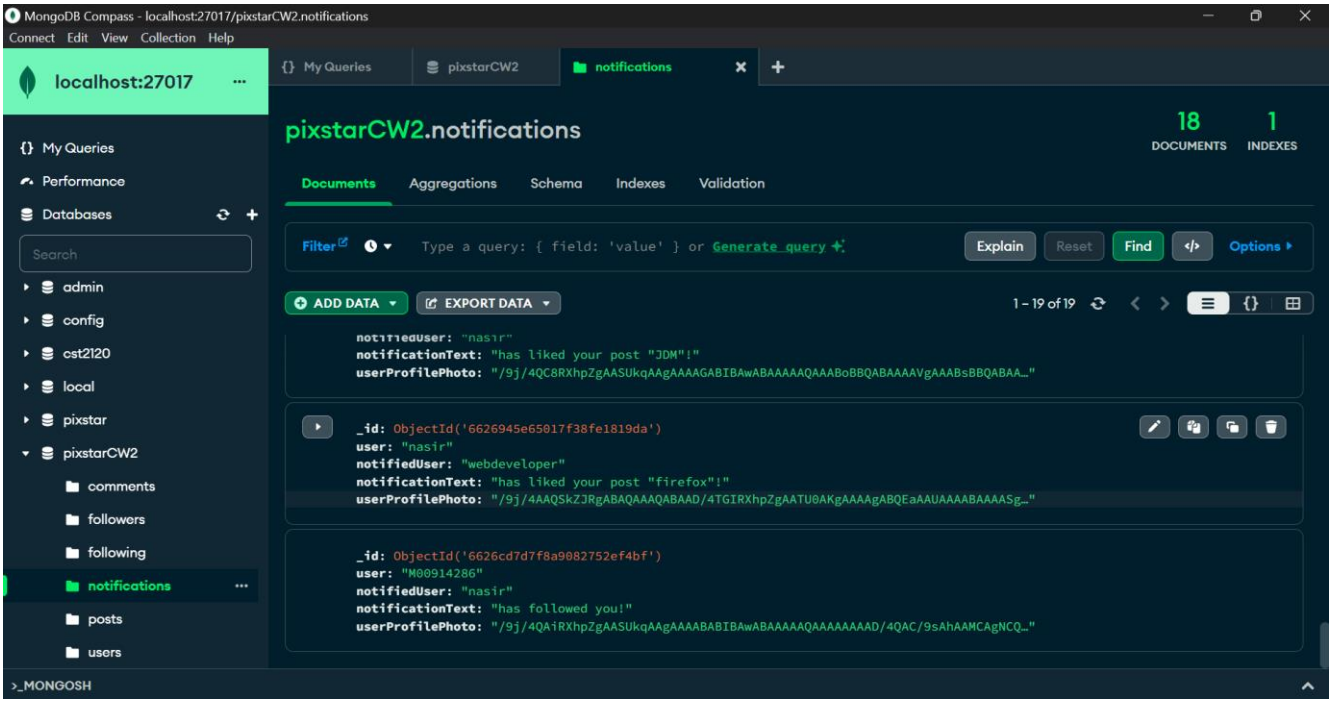
The user's profile photo in notifications has also been updated.

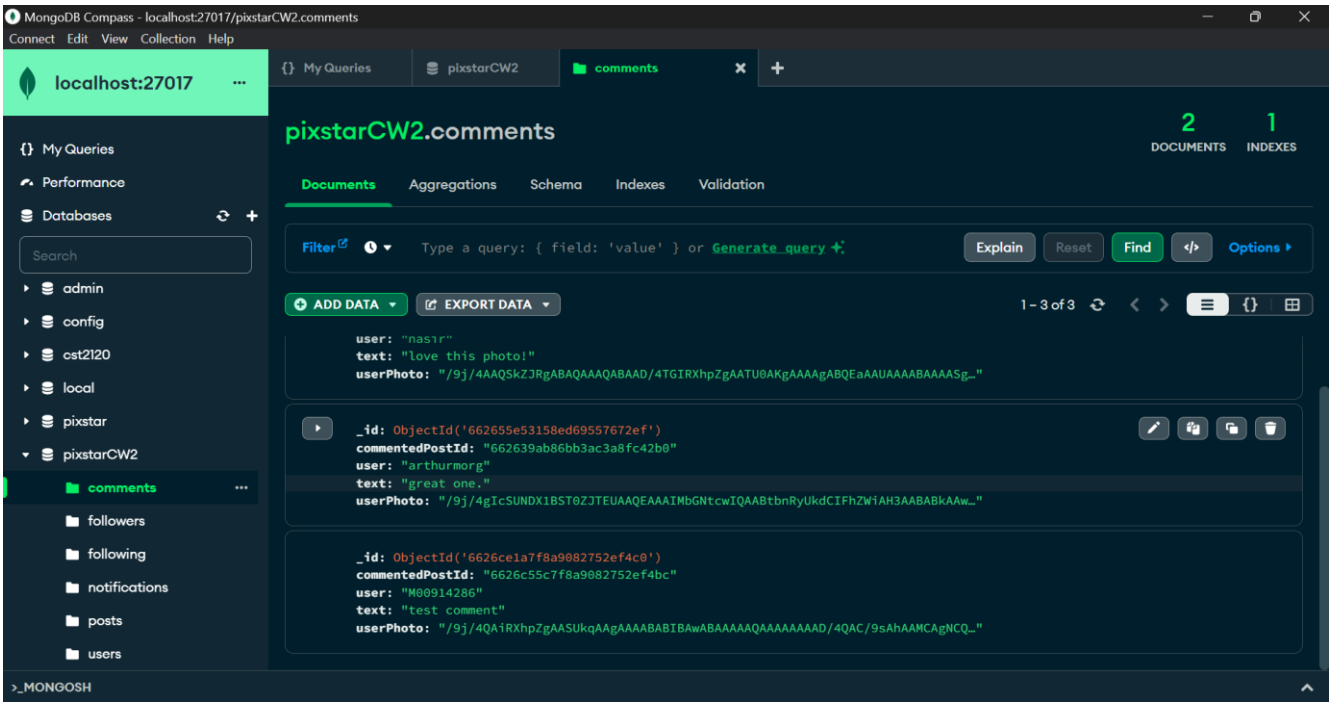


The document in mongoDB in the users collection has been updated.



the documents in the notifications and comments collections have also been updated with the user's new profile photo.

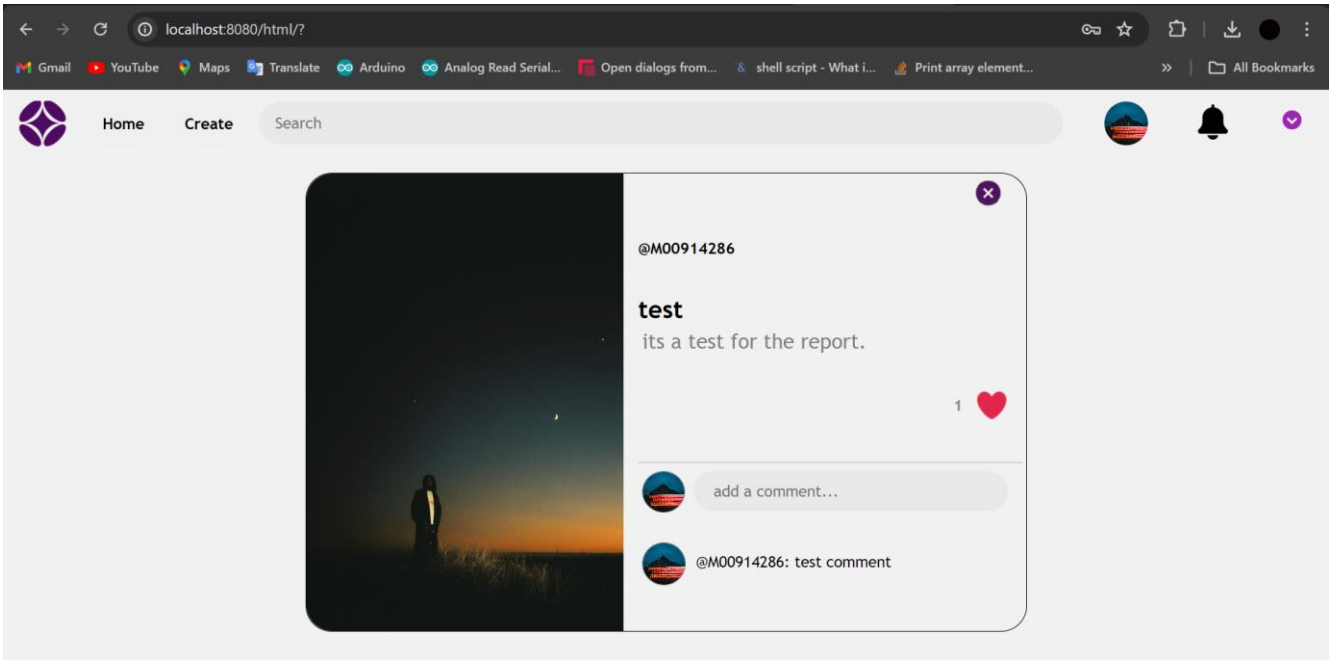

MongoDB Compass - localhost:27017/pixstarCW2.notifications
Connect Edit View Collection Help
localhost:27017
My Queries
Performance
Databases
Search
admin
config
cst2120
local
pixstar
pixstarCW2
comments
followers
following
notifications
posts
users
My Queries
pixstarCW2
notifications
pixstarCW2.notifications
18 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 - 19 of 19
notifiedUser: "nasir"
notificationText: "has liked your post "JDM"!"
userProfilePhoto: "/9j/4QC8RXhpZgAASUkqAAgAAAAGABIBAwABAAAAAQAAABoBBQABAAAAVgAAABsBBQABAA…"
_id: ObjectId('6626945e65017f38fe1819da')
user: "nasir"
notifiedUser: "webdeveloper"
notificationText: "has liked your post "firefox"!"
userProfilePhoto: "/9j/4AAQSkZJRgABAQAAAQABAAD/4TGIRXhpZgAATU0AKgAAAAgABQEaAAUAAAABAAAASg…"
_id: ObjectId('6626cd7d7f8a9082752ef4bf')
user: "M00914286"
notifiedUser: "nasir"
notificationText: "has followed you!"
userProfilePhoto: "/9j/4QAiRXhpZgAASUkqAAgAAAABABIBAwABAAAAAQAAAAAAAAD/4QAC/9sAhAAMCAgNCQ…"
>_MONGOSH

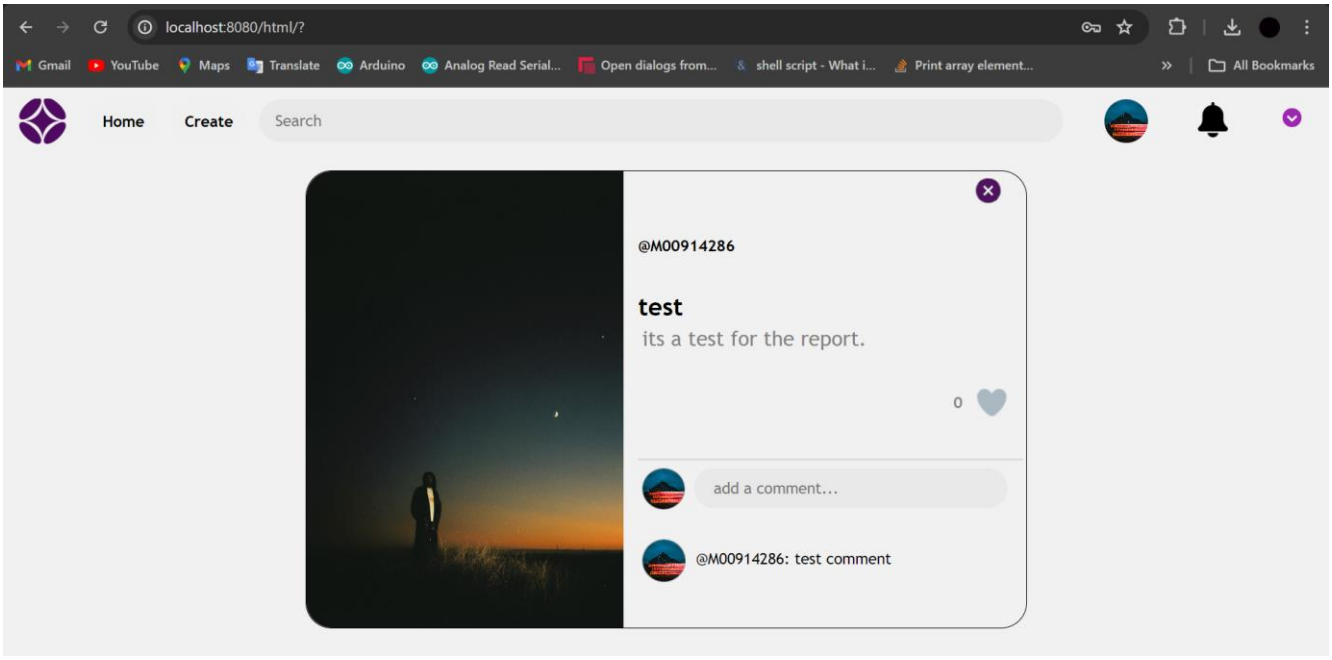

MongoDB Compass - localhost:27017/pixstarCW2.comments
Connect Edit View Collection Help
localhost:27017
My Queries
Performance
Databases
Search
admin
config
cst2120
local
pixstar
pixstarCW2
comments
followers
following
notifications
posts
users
My Queries
pixstarCW2
comments
pixstarCW2.comments
2 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 - 3 of 3
user: "nasir"
text: "love this photo!"
userPhoto: "/9j/4AAQSkZJRgABAQAAAQABAAD/4TGIRXhpZgAATU0AKgAAAAgABQEaAAUAAAABAAAASg…"
_id: ObjectId('662655e53158ed69557672ef')
commentedPostId: "662639ab86bb3ac3a8fc42b0"
user: "arthurmorg"
text: "great one."
userPhoto: "/9j/4gIcSUNDX1BST0ZJTEUAAQEAAAIMbGNtcwIQAABtbnRyUkdCIFhZWiAH3AABABkAAw…"
_id: ObjectId('6626ce1a7f8a9082752ef4c0')
commentedPostId: "6626c55c7f8a9082752ef4bc"
user: "M00914286"
text: "test comment"
userPhoto: "/9j/4QAiRXhpZgAASUkqAAgAAAABABIBAwABAAAAAQAAAAAAAAD/4QAC/9sAhAAMCAgNCQ…"
>_MONGOSH

Likes And Comments

Likes

For the likes, whenever the user is viewing a post, they can click on the heart to like and unlike the post. The heart is filled when the user has liked the post, and empty when they haven't, the likes are adjusted accordingly.

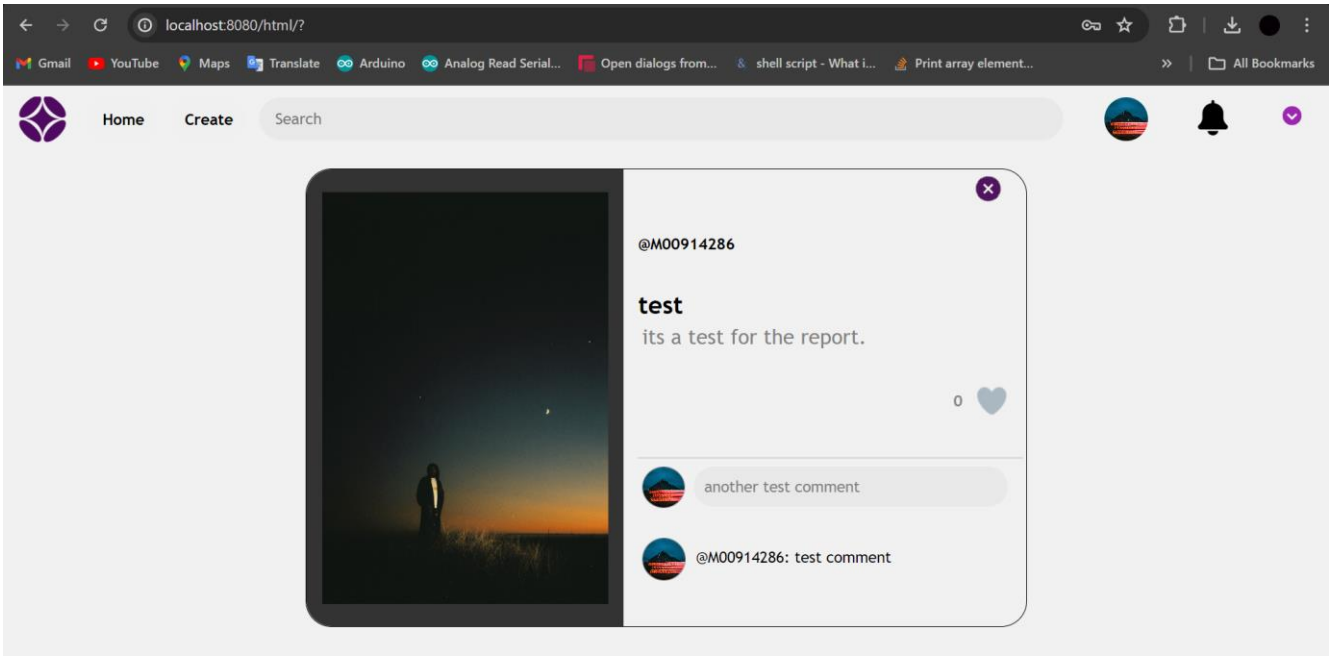




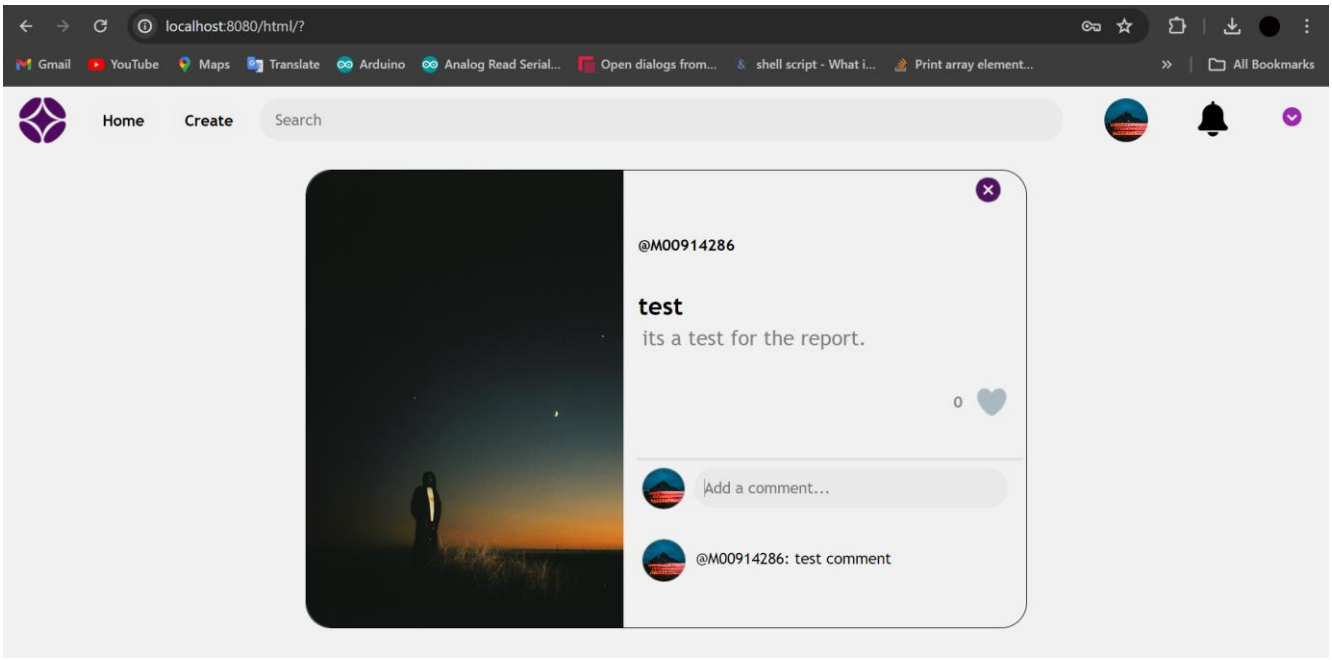
The state of the like button is determined everytime a user view a post by checking the likedArray in the mongoDB database.

Comments

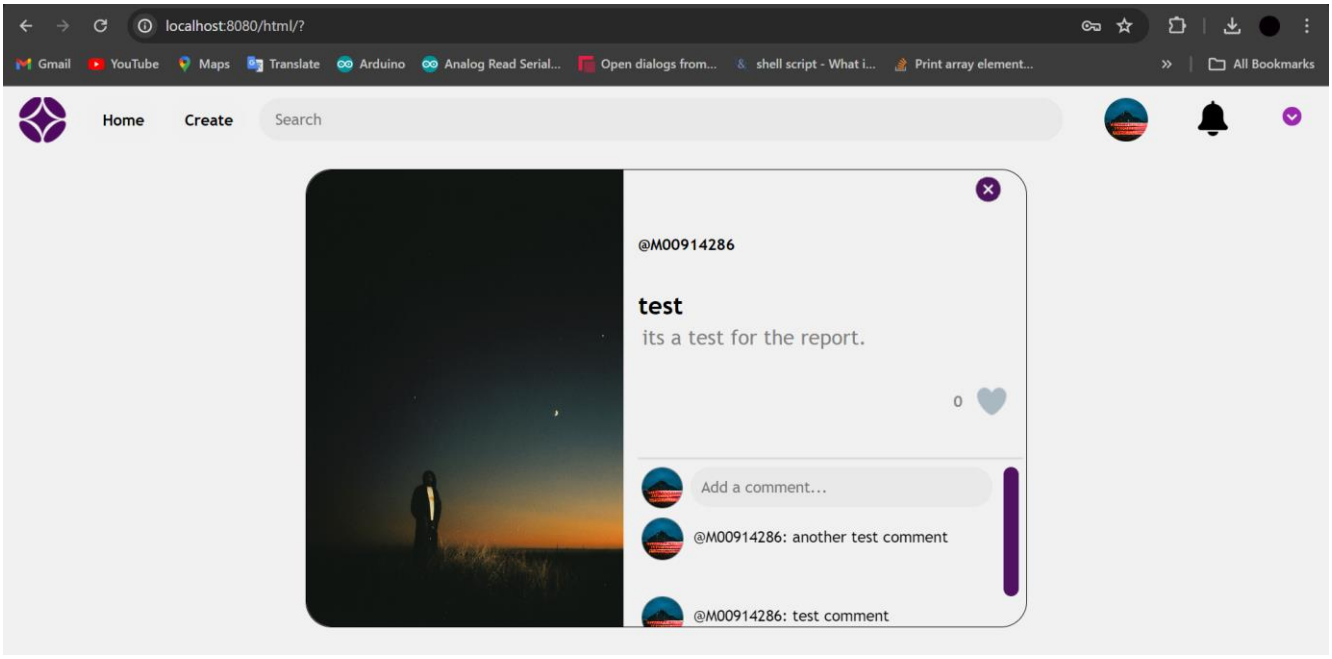
The user can comment on posts by typing their comment and hitting enter, to view their comments they can exit the post and open it again.



After hitting enter, the field is cleared and the user can write another comment, the user is free to comment how many ever time they like.



The comment appears after the user leaves and re-opens the post.

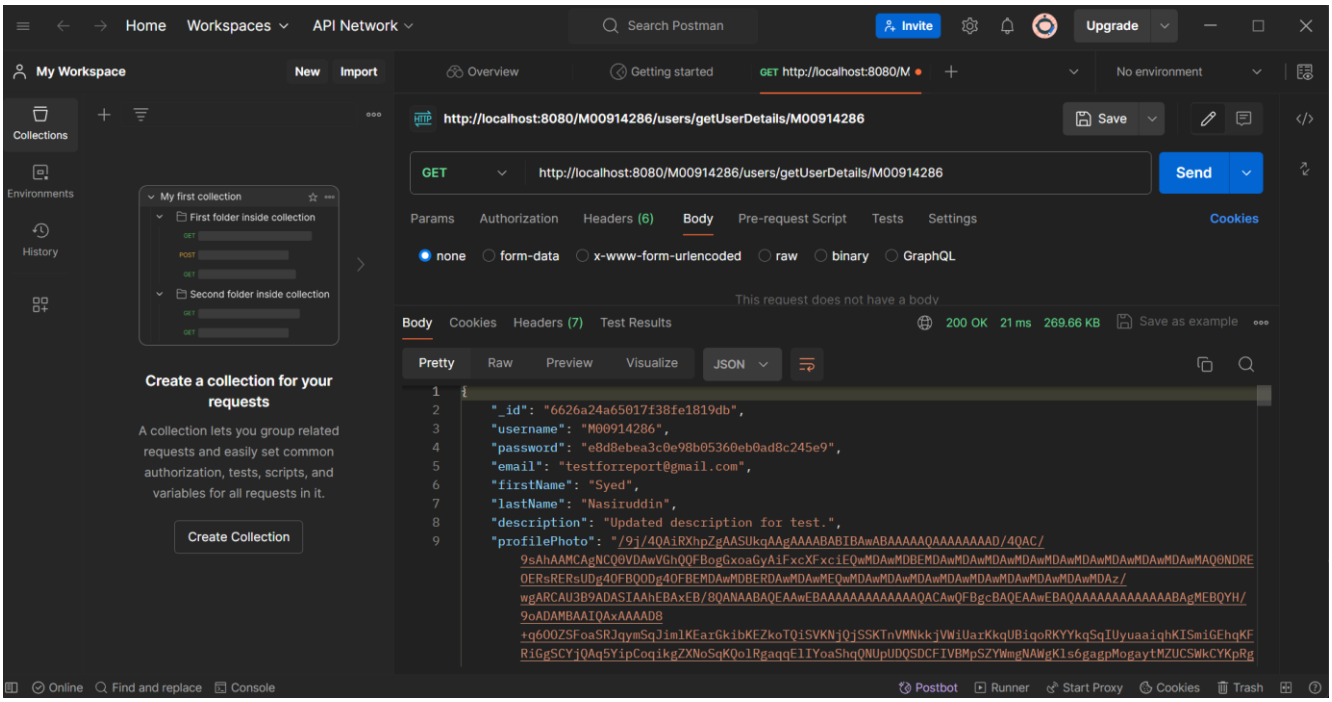


Webservices

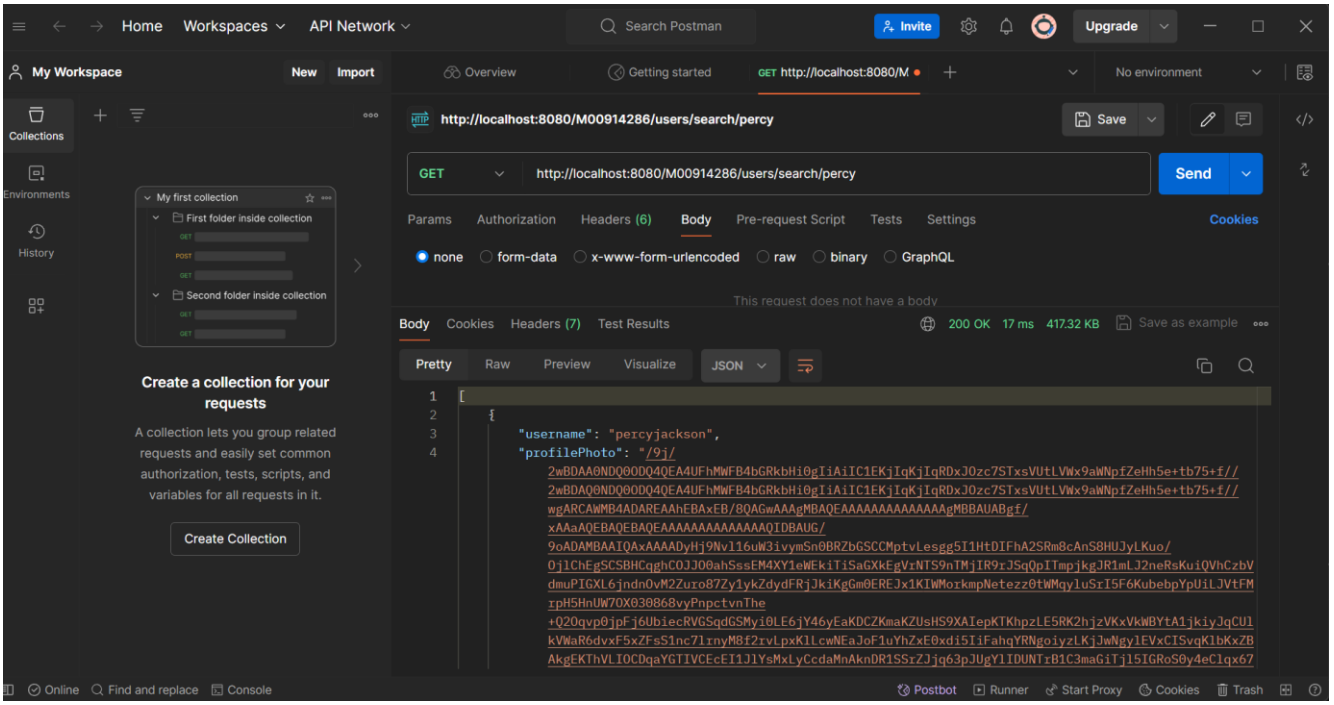
Users Collection

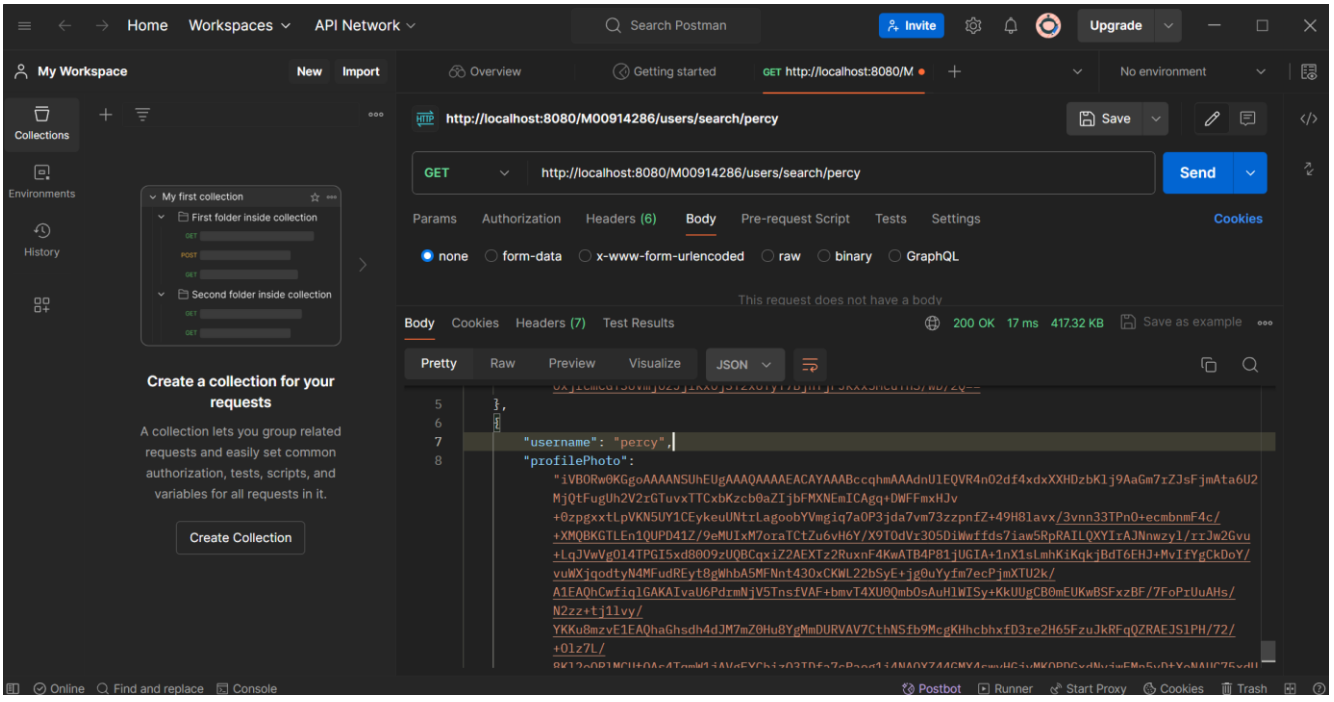
The following webservices are related to the users collection:

1. **getUserDetails**
   This webservices gets the provided user's details from the users collection.



2. **searchUsers**
   This webservice searches for users with the provided username, even users with a part of the provided username in their username are returned, the users username a profile photo is returned.

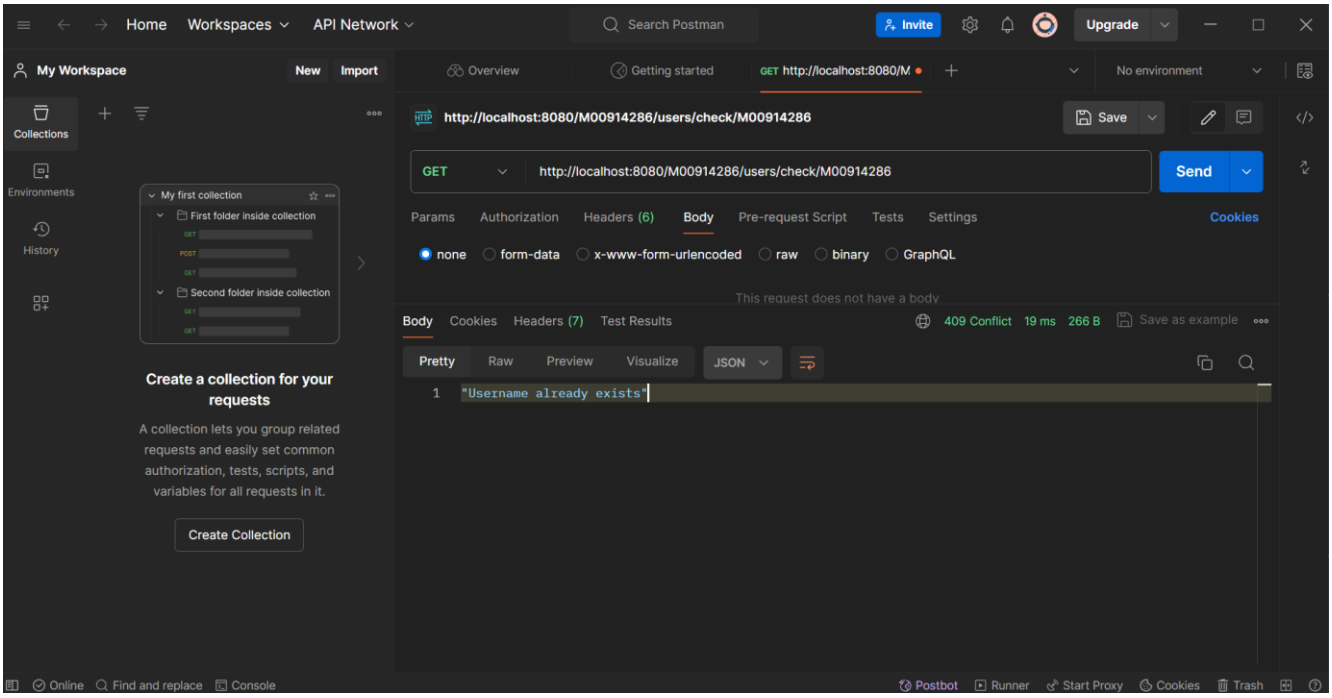




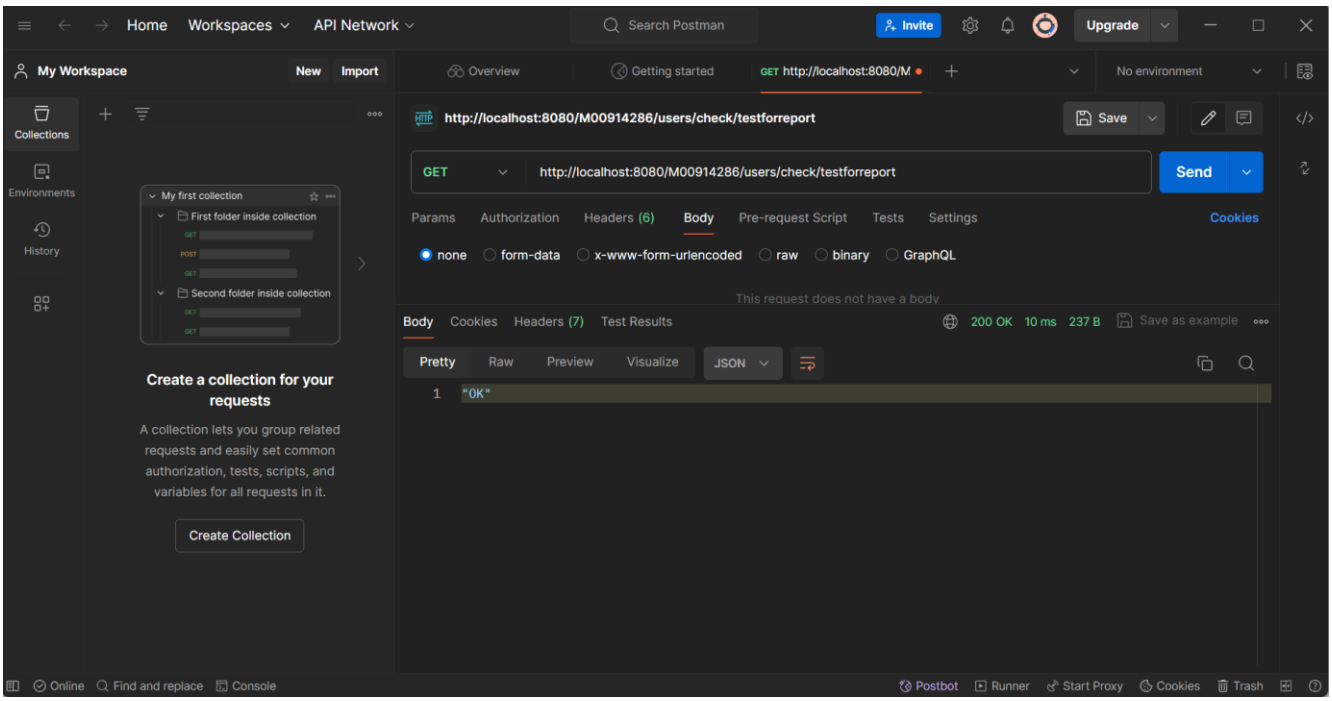
In the above screenshots, percy is provided and both “percy” and “percyjackson” are returned, this webservice is used for the users search functionality provided by the search bar.

3. checkExistingUser

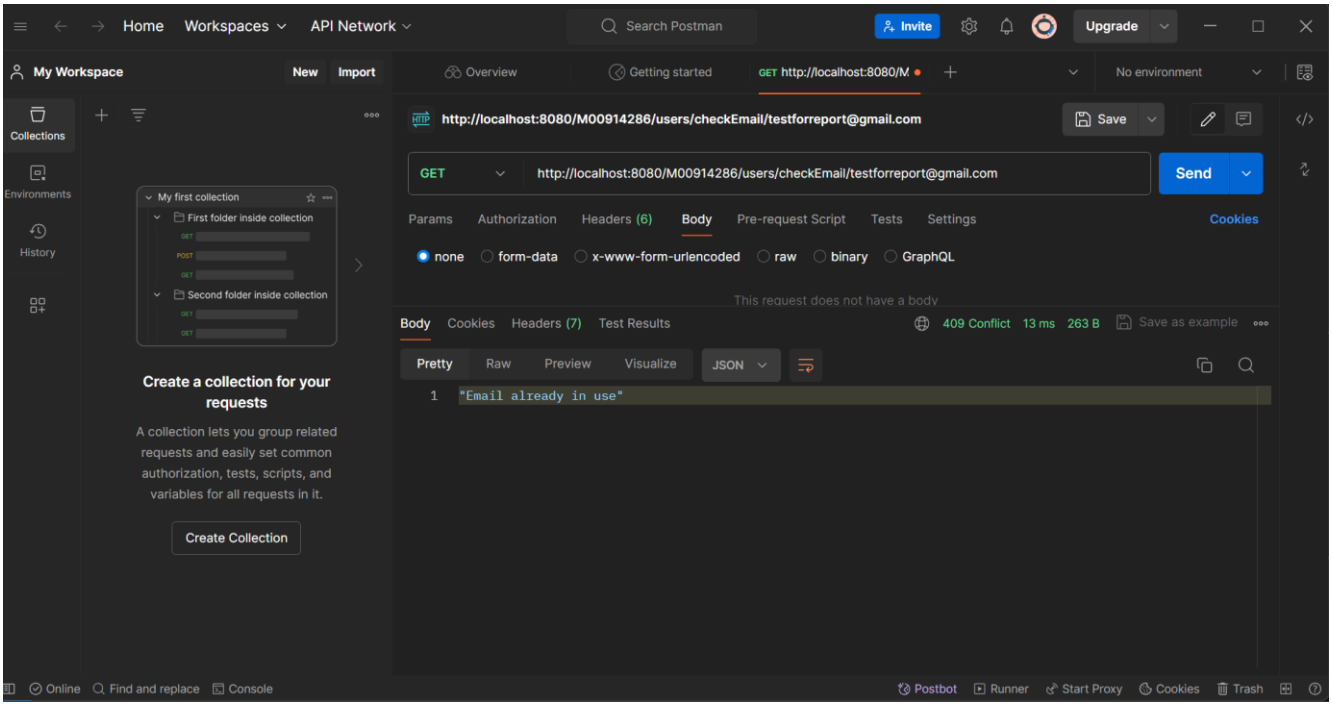
checks if the provided username in the parameters is already taken by another user in the collection, if yes it sends the below reponse.



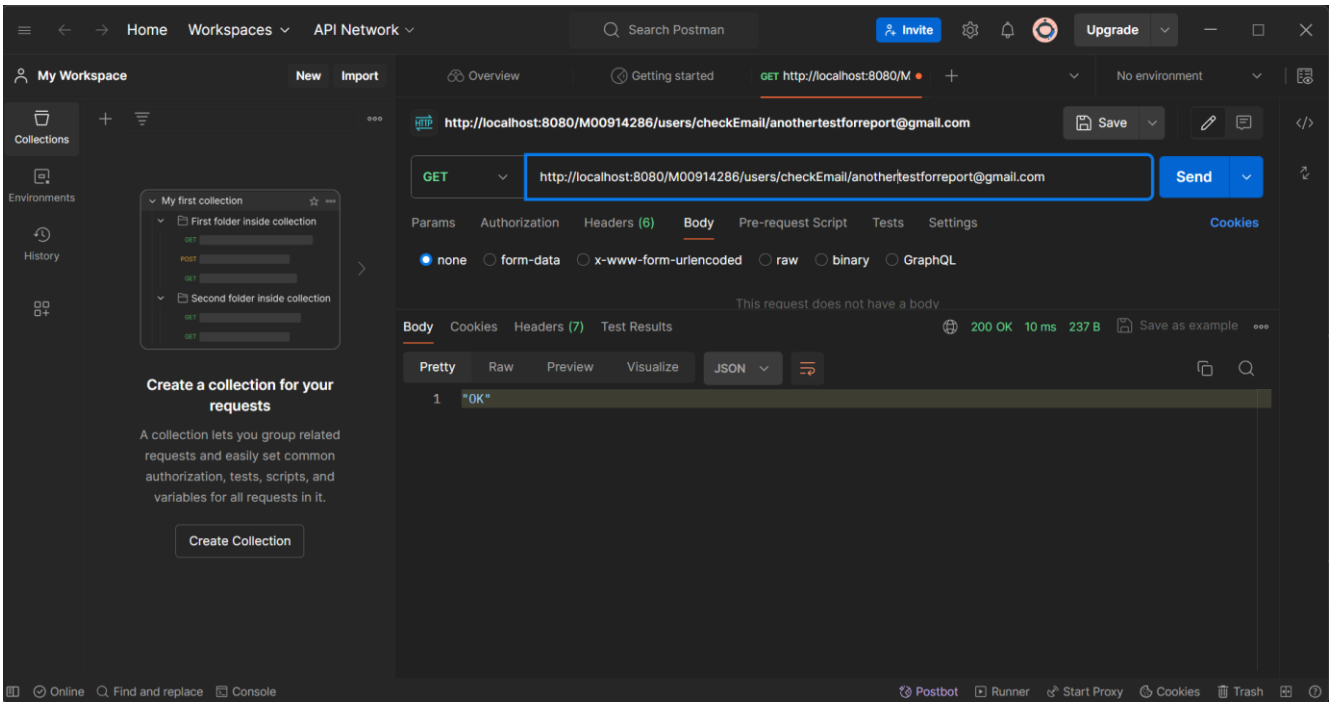
if not, then it sends the following response,



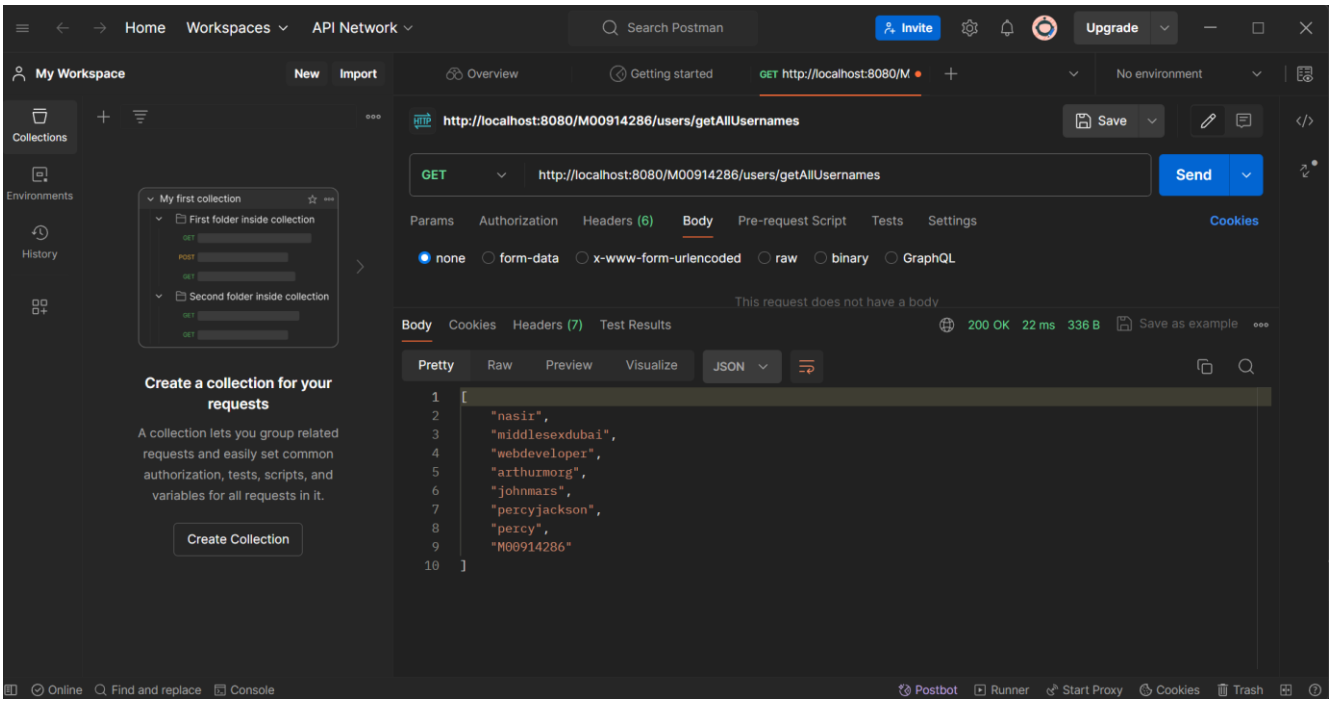
4. **checkExistingEmail**
   checks if the provided email in the parameters is already taken by another user in the collection, if yes it sends the below reponse.



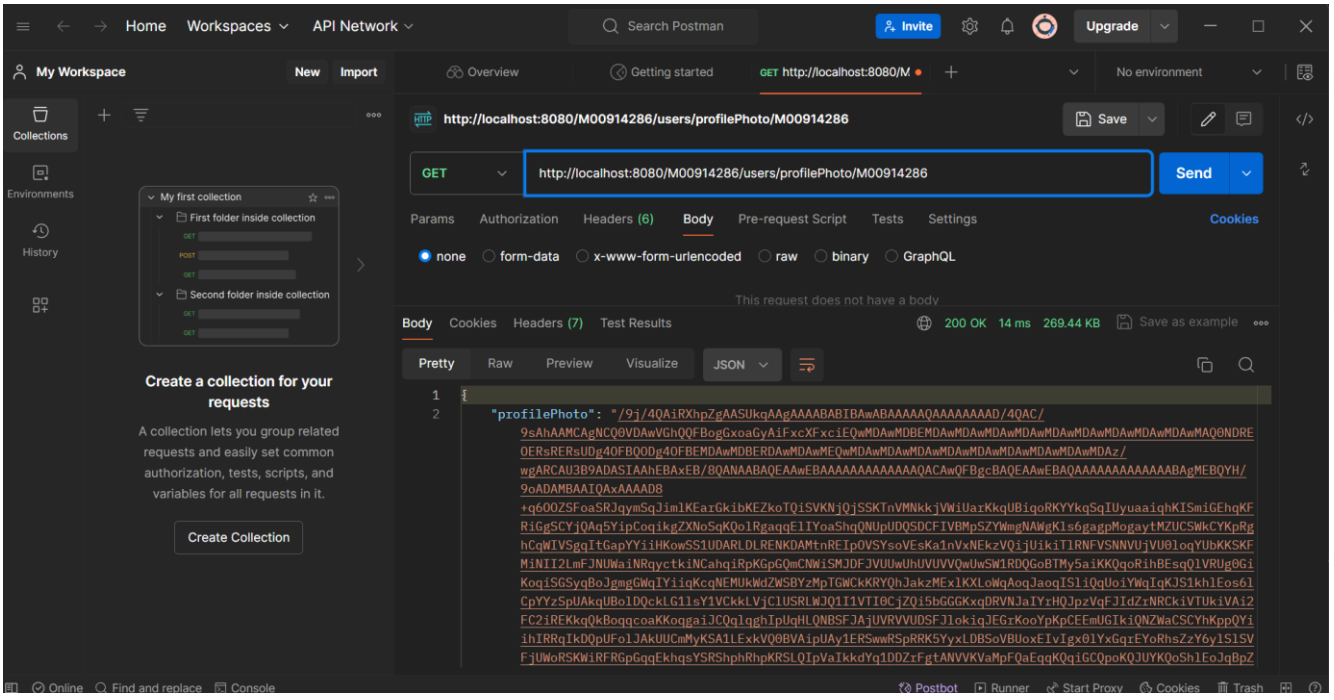
If not, then it sends the following response,



5. **getAllUsernames**
   gets all the usernames of the users stored in the collection.

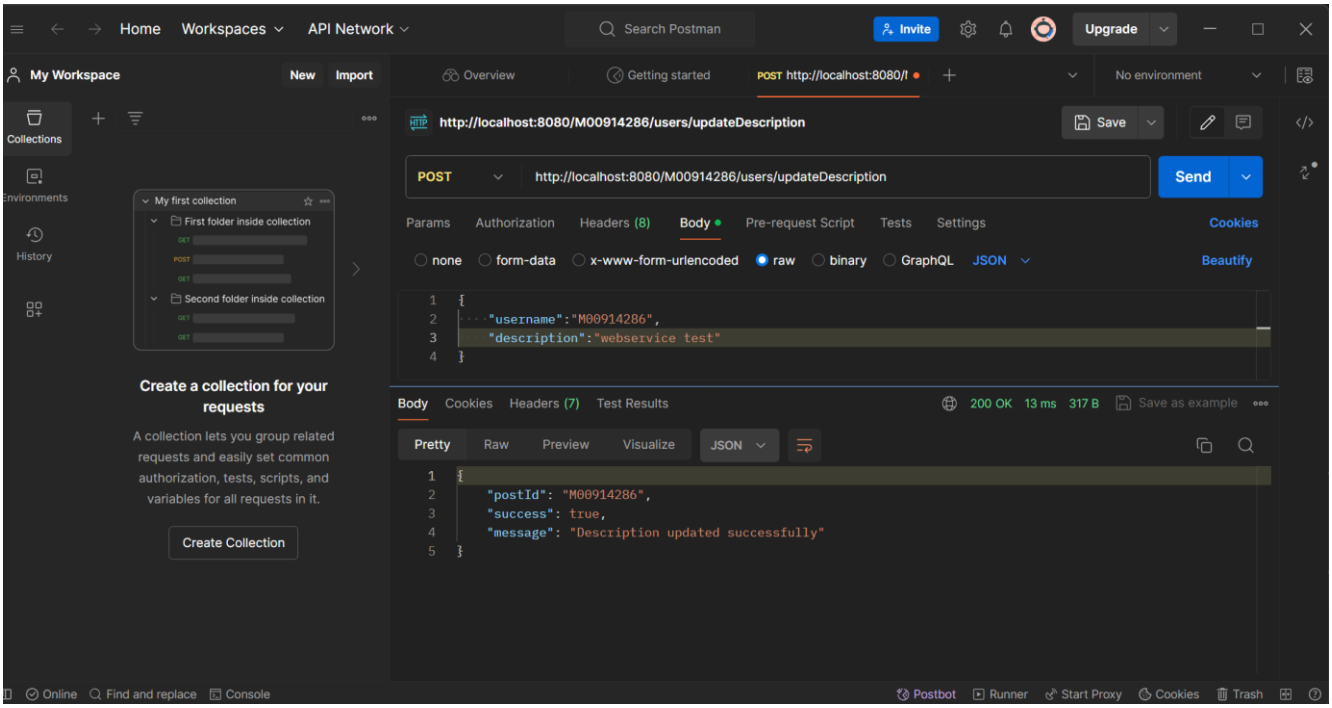


6. **getUserProfilePhoto**
   gets the provided user's profile photo.

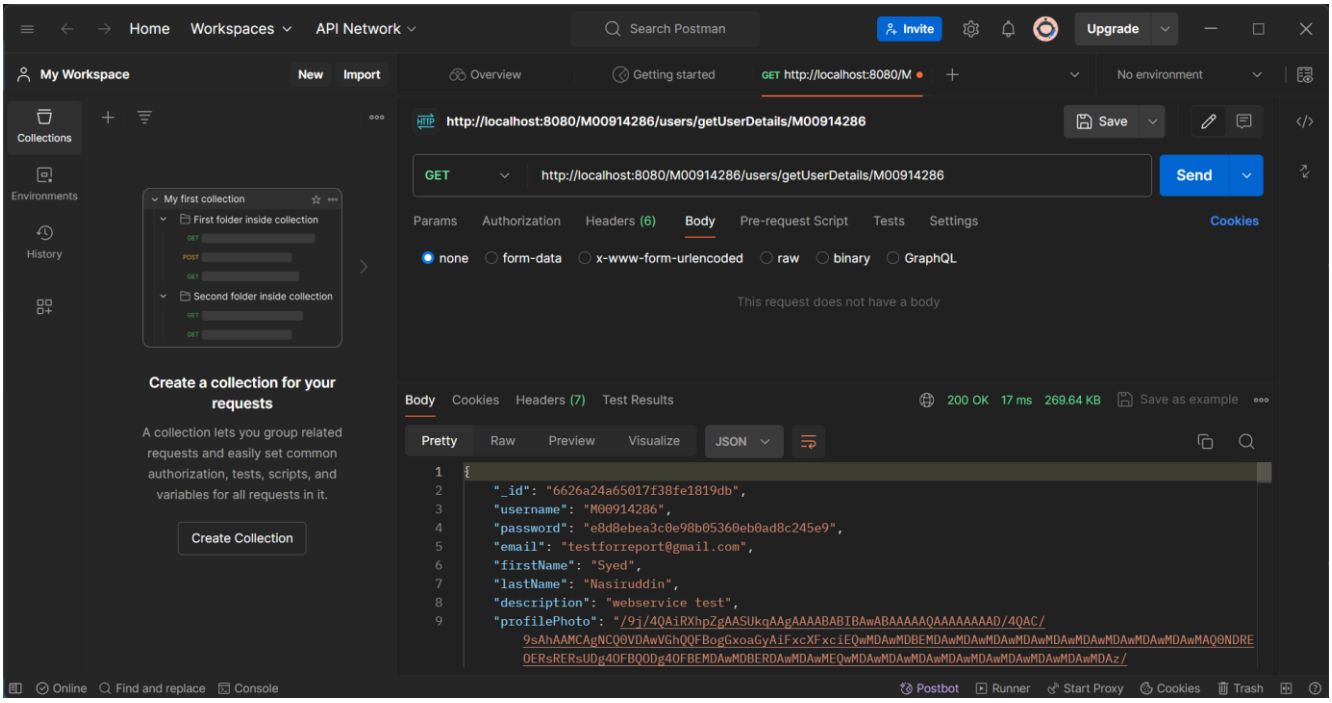


7. updateUserDescription

updates the provided user's description.

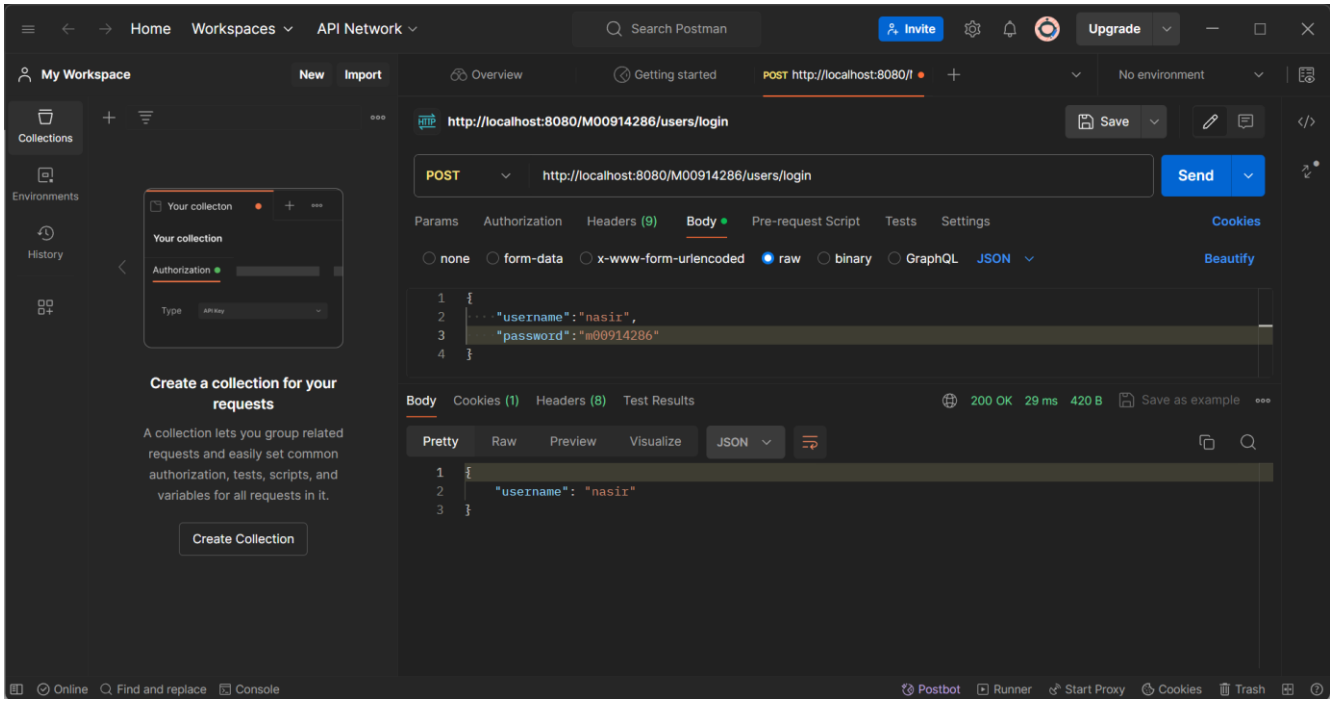


the update can be seen below,



8. checkUser

This webservice is used for login, it takes a username and password, checks if the provided password matched the decrypted stored password, if so it returns the stored username.



Excluded Webservices

For the following webservices I have not used postman:

- **signupPost**
- **updateProfilePhoto**
- **updateUserProfilePhoto**

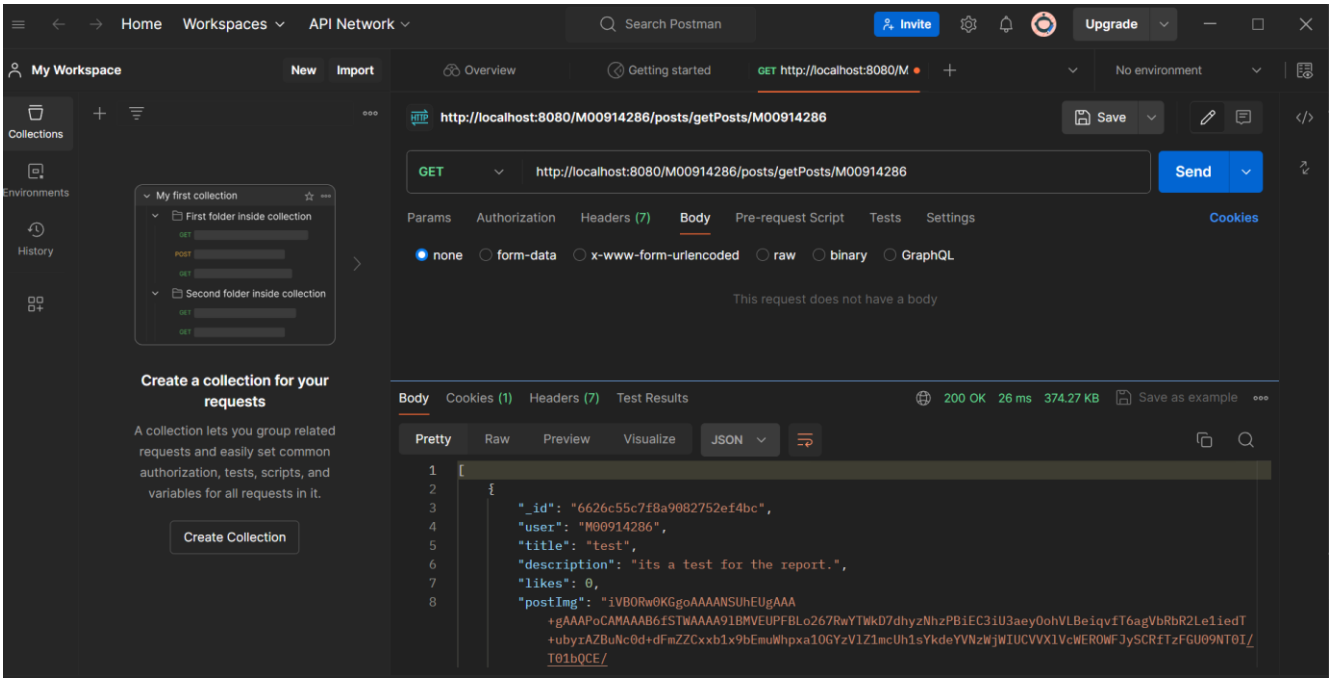
The first to work alongside each other, they are passed as request 1 and request 2 to the postToDatabase function, the are used for the creation of a user, signupPost first adds all the text data to the document and then the document is updated with profile photo provided during the sign-up process.

The last of the three, is used when the user is editing their profile, both the profile edit and sign up process will be covered in the video.

Posts Collection

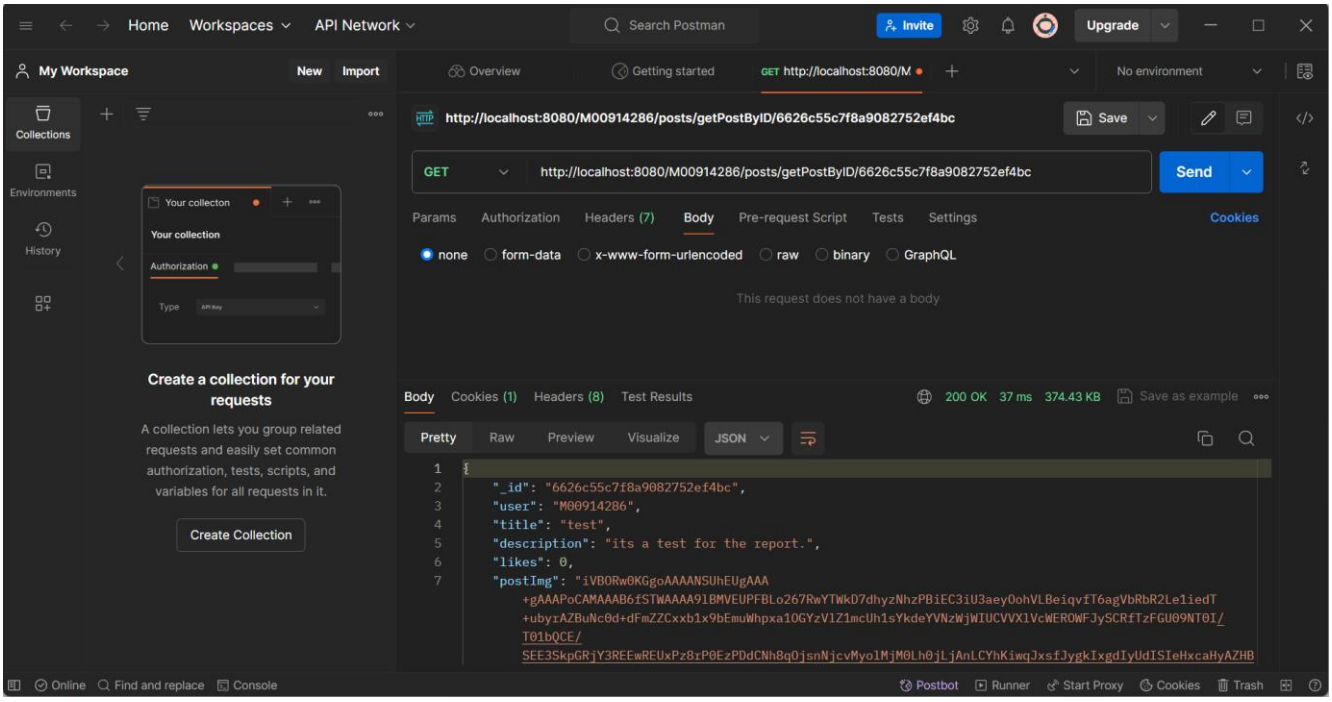
The below webservices are related to the posts collection:

1. **getUserPosts**
   gets the provided users posts that they made.



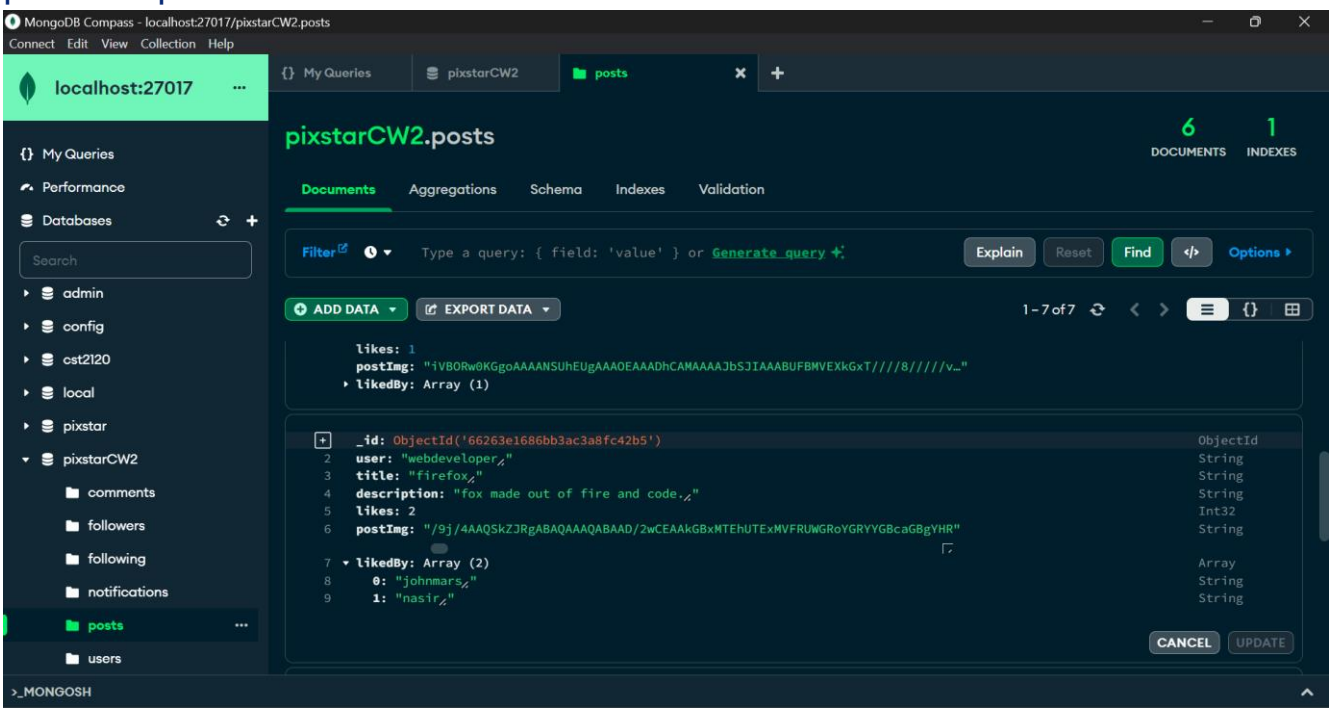
2. getPostDetailsbyID

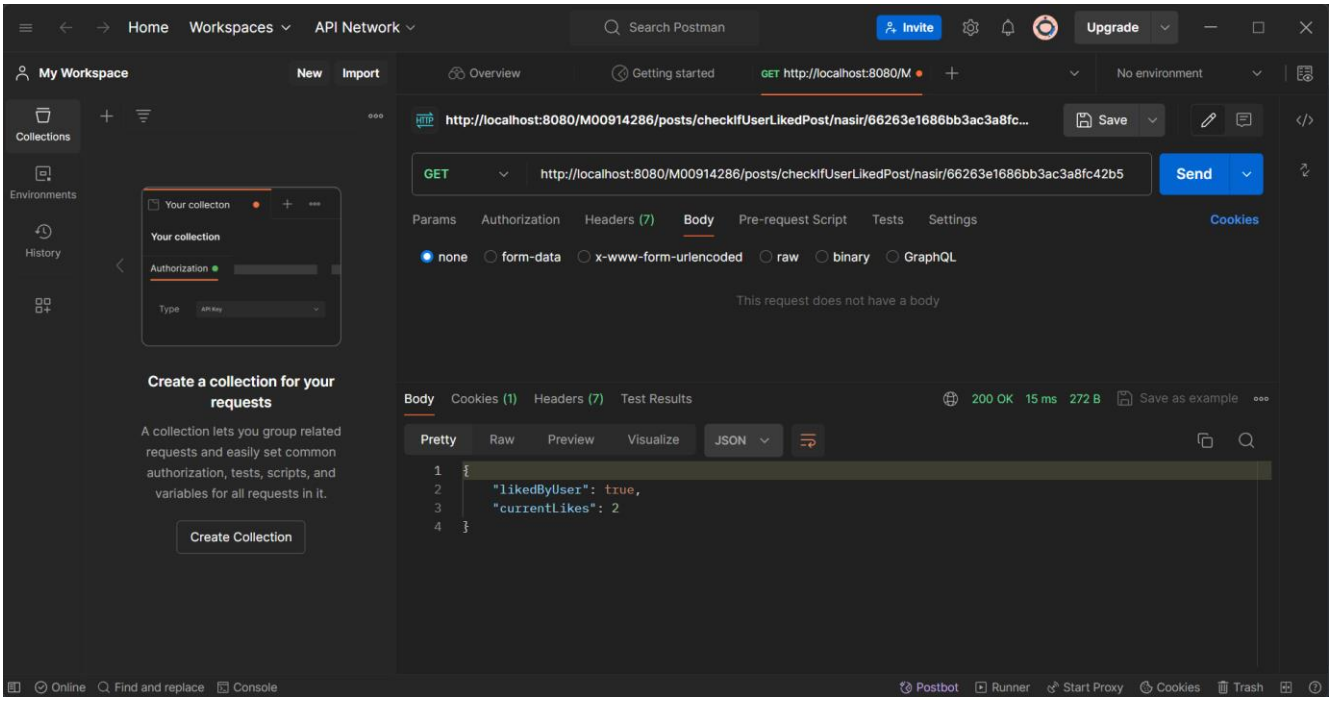
gets a post's details by the value of it its _id attribute.



3. checkIfUserLikedPost

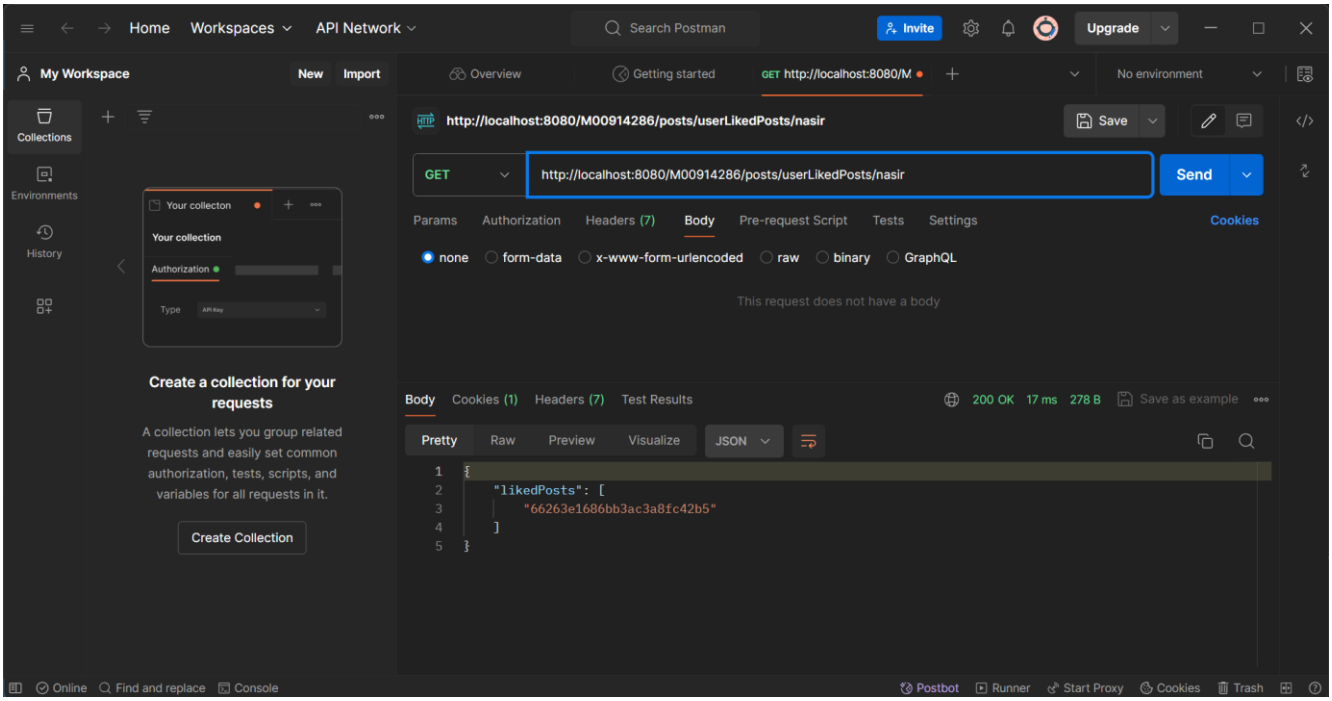
checks if the provided user is in the likedBy array of the post that has the provided post Id.





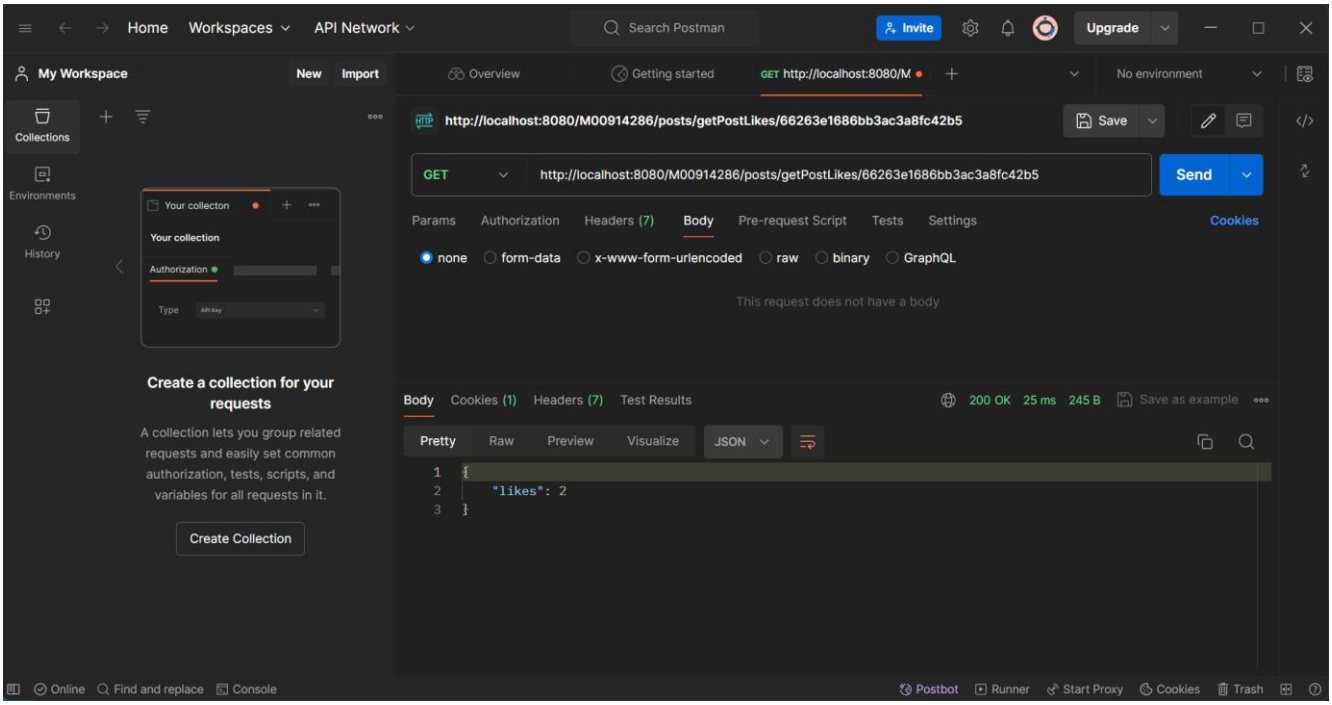
4. getUserLikedPosts

gets the post Ids of the posts that the provided user has liked.



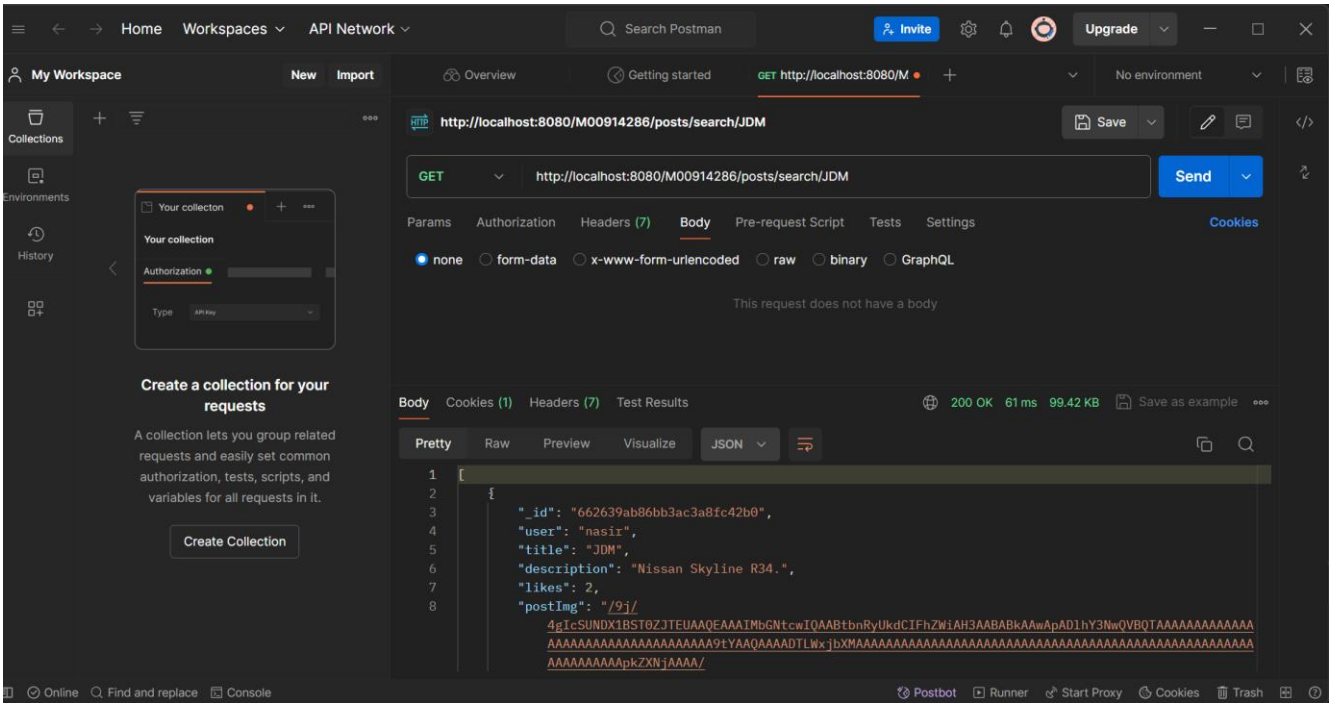
5. getPostLikes

gets the like count of the post that has the provided post Id.



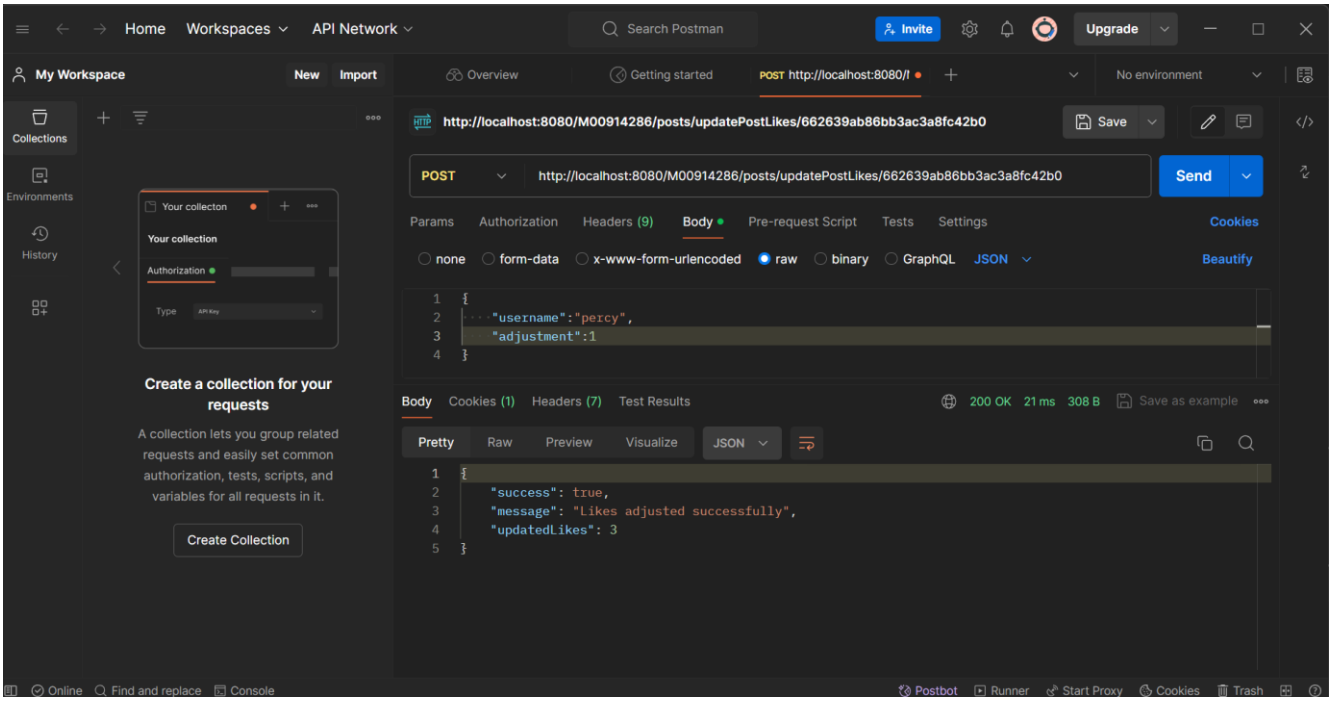
6. searchPostsByTitle

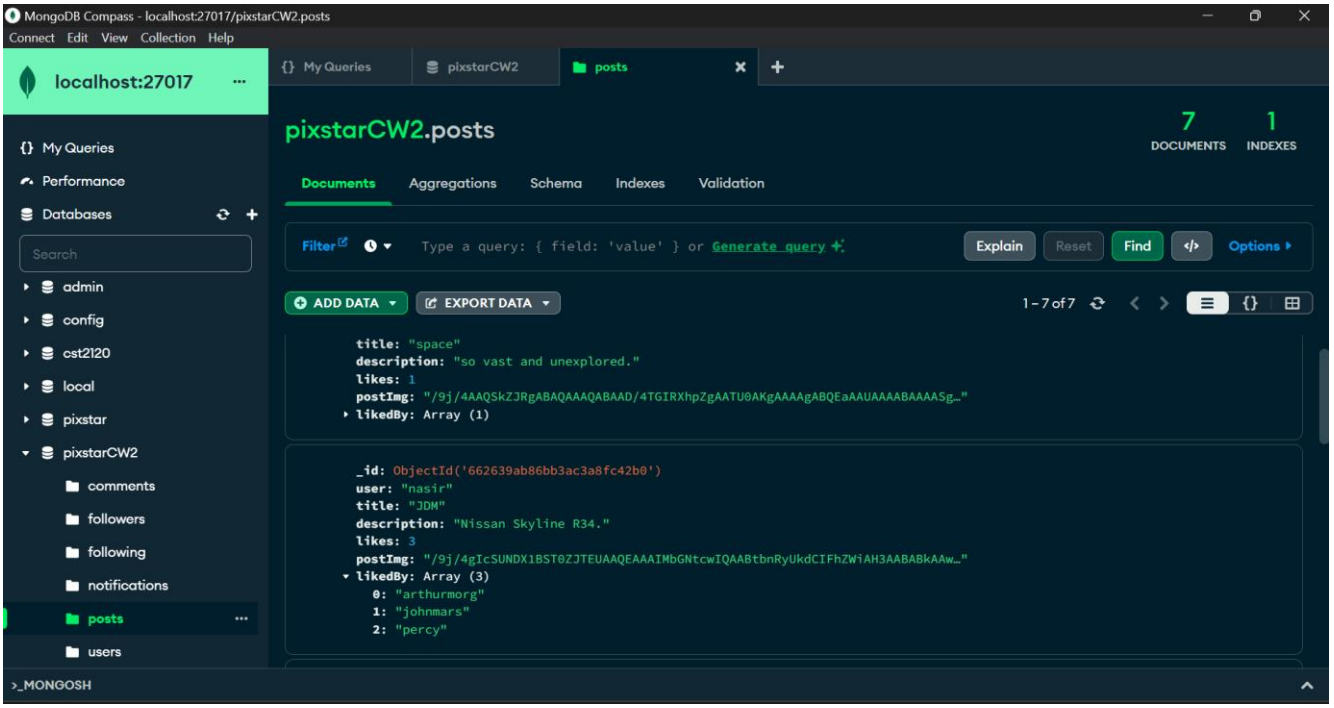
gets posts that have the provided word/phrase in their title.



7. updatePostLikes

updates the post's current liked by adding the adjustment provided, also takes the user and adds them to the likedBy array of the post.





Excluded Webservices

The following webservices have not been documented using postman:

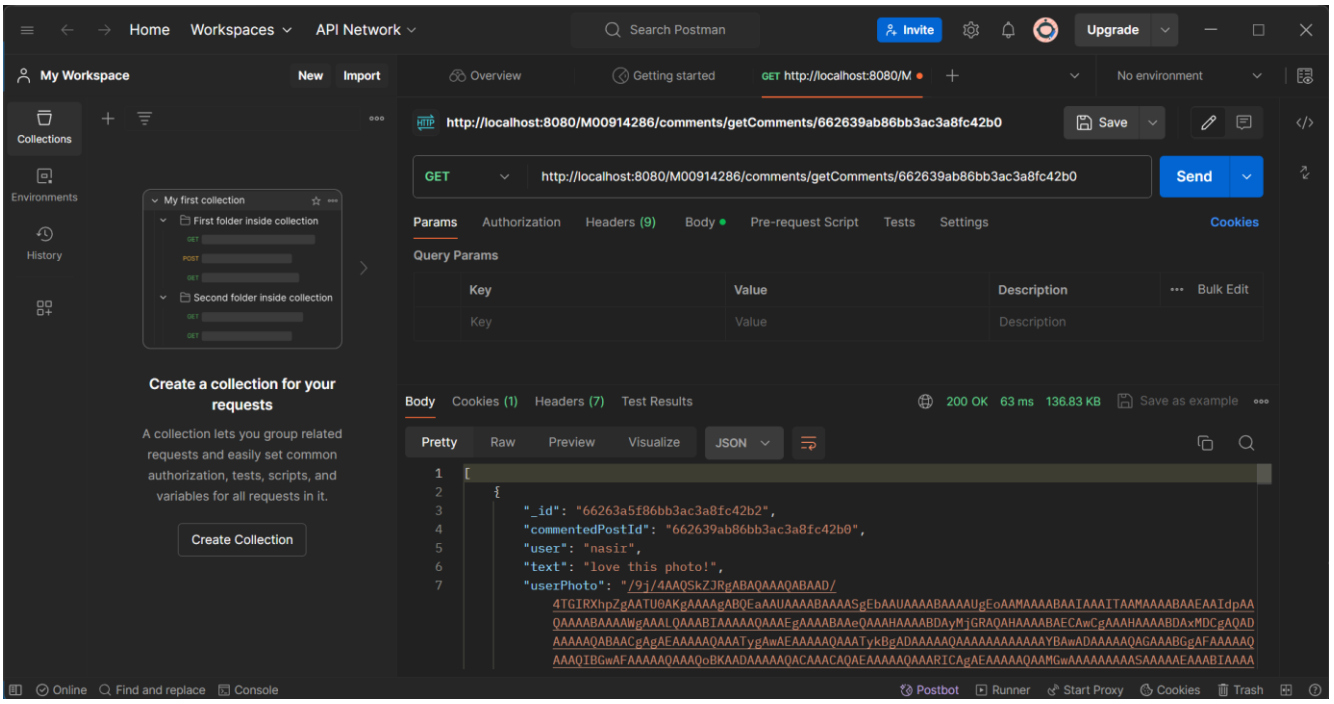
- **makePost**
- **updatePostImage**

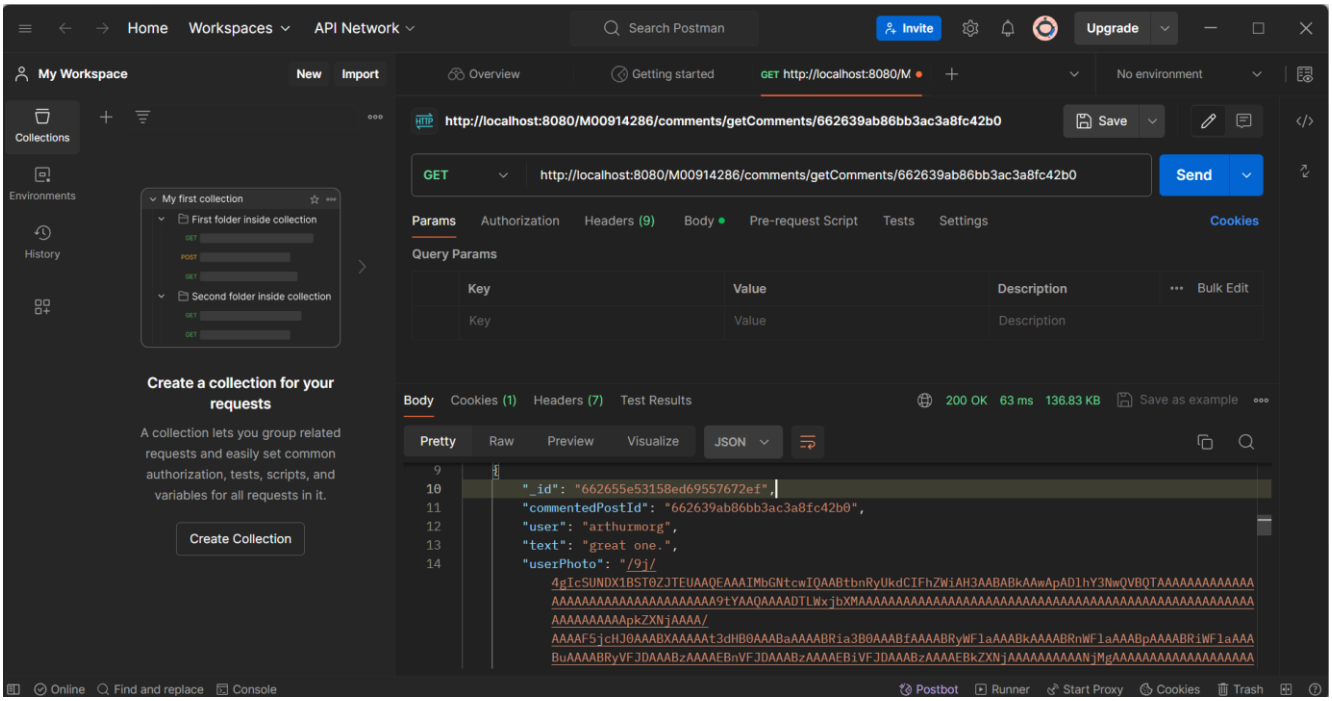
Both of these webservices are involved in the post creation process and are passed as request 1 and request 2 to the postToDatabase function, the post creation process will be covered in the video, a successful creation of the post will indicate the correct working of these webservices.

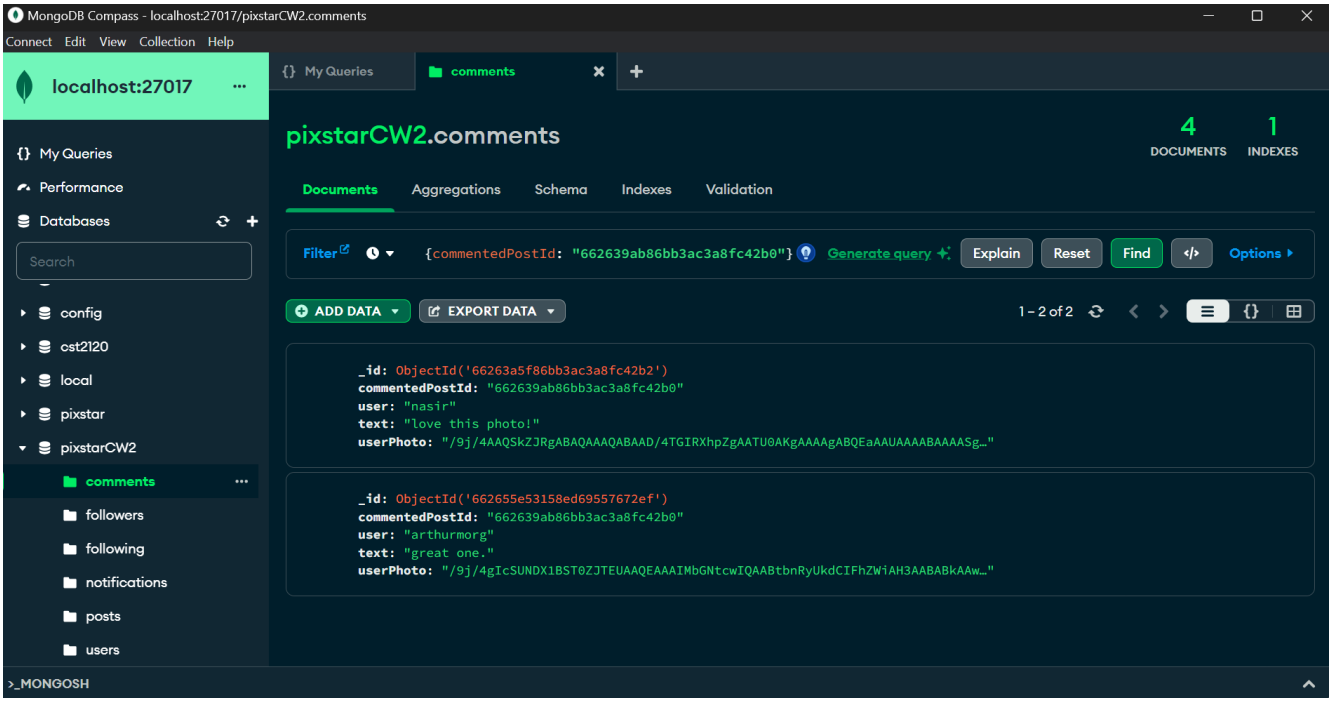
Comments Collection

These webservices are related to the comments collection:

1. getComments
   gets all the comments made for the post that has the provided post Id.




Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
GET http://localhost:8080/M
No environment
Collections
Environments
History
http://localhost:8080/M00914286/comments/getComments/662639ab86bb3ac3a8fc42b0
Save
GET
http://localhost:8080/M00914286/comments/getComments/662639ab86bb3ac3a8fc42b0
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Body
Cookies (1)
Headers (7)
Test Results
200 OK
63 ms
136.83 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
"_id": "662655e53158ed69557672ef",
"commentedPostId": "662639ab86bb3ac3a8fc42b0",
"user": "arthurmorg",
"text": "great one.",
"userPhoto": "/9j/
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash


MongoDB Compass - localhost:27017/pixstarCW2.comments
Connect
Edit
View
Collection
Help
localhost:27017
My Queries
Performance
Databases
Search
config
cst2120
local
pixstar
pixstarCW2
comments
followers
following
notifications
posts
users
My Queries
comments
pixstarCW2.comments
4
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
{commentedPostId: "662639ab86bb3ac3a8fc42b0"}
Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 - 2 of 2
_id: ObjectId('66263a5f86bb3ac3a8fc42b2')
commentedPostId: "662639ab86bb3ac3a8fc42b0"
user: "nasir"
text: "love this photo!"
userPhoto: "/9j/4AAQSkZJRgABAQAAAQABAAD/4TGIRXhpZgAATU0AKgAAAAgABQEaAAUAAAABAAAASg…"
_id: ObjectId('662655e53158ed69557672ef')
commentedPostId: "662639ab86bb3ac3a8fc42b0"
user: "arthurmorg"
text: "great one."
userPhoto: "/9j/4gIcSUNDX1BST0ZJTEUAAQEAAAIMbGNtcwIQAABtbnRyUkdCIFhZWiAH3AABABkAAw…"
>_MONGOSH

Excluded Webservices

The following webservices have not been documented using postman:

- makeComment
- updateCommentPhoto
- updateUserProfilePhotoInComments

The first two are involved in the comment creation process, and are passed as request 1 and request 2 to the postToDatabase function, a successful creation of the comment (with both text and profile photo in base64 format) will indicated the proper working of these services.

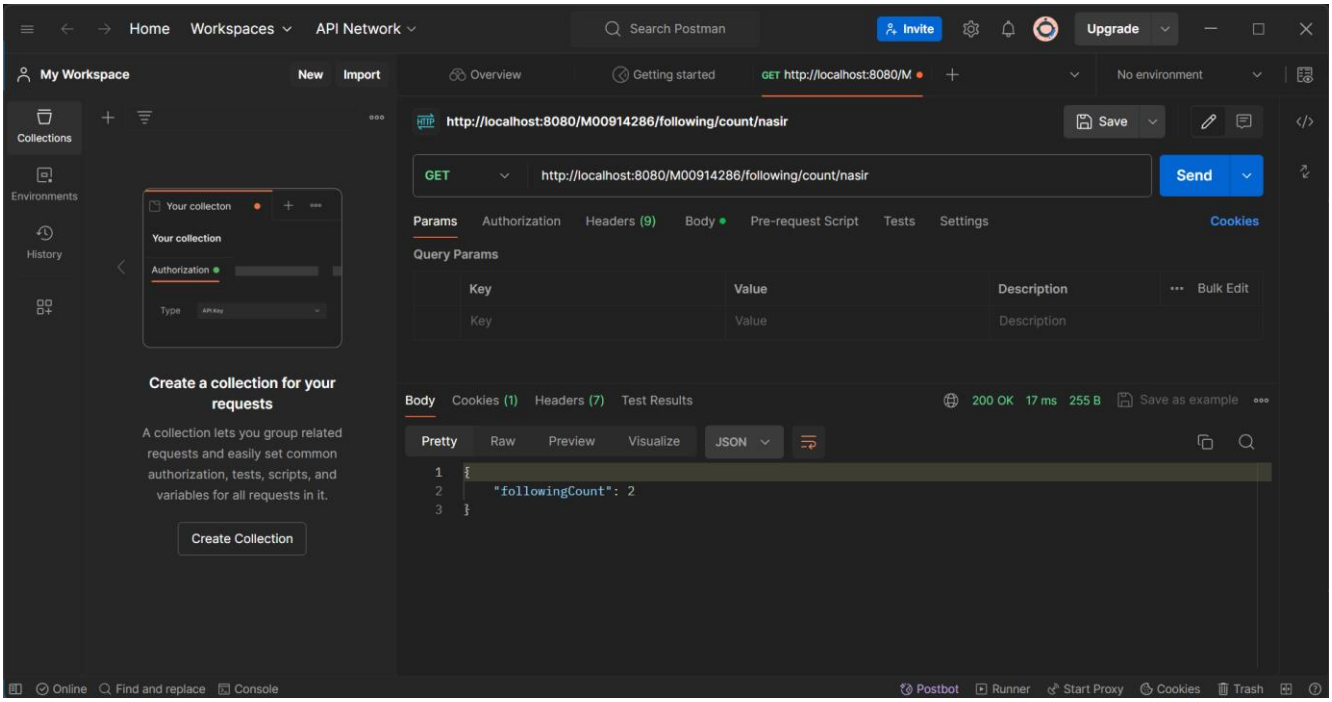
The last of the three is used when the user changes their profile photo by editing it in their account tab, and a change in the profile photo of the comments left by the user on posts would indicate the correct working of this webservice.

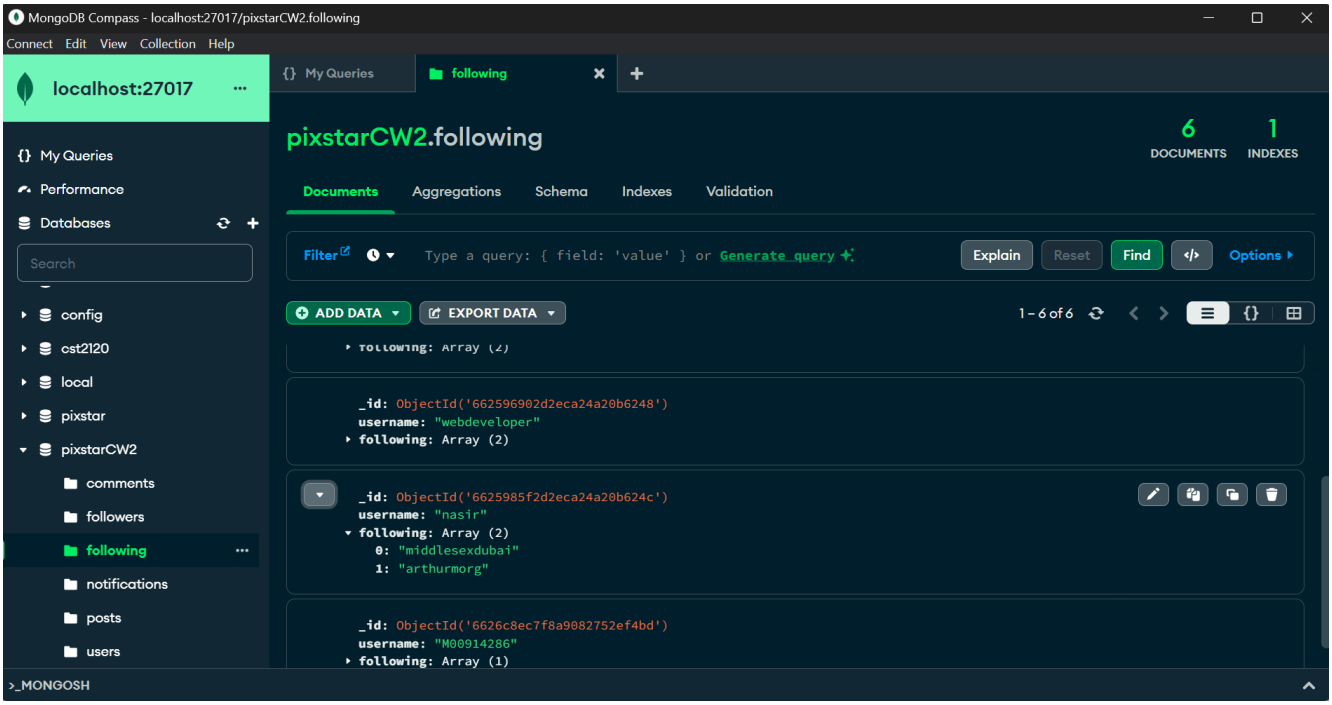
This will be covered in the video.

Following Collection

The below webservices are related to the “following” collection:

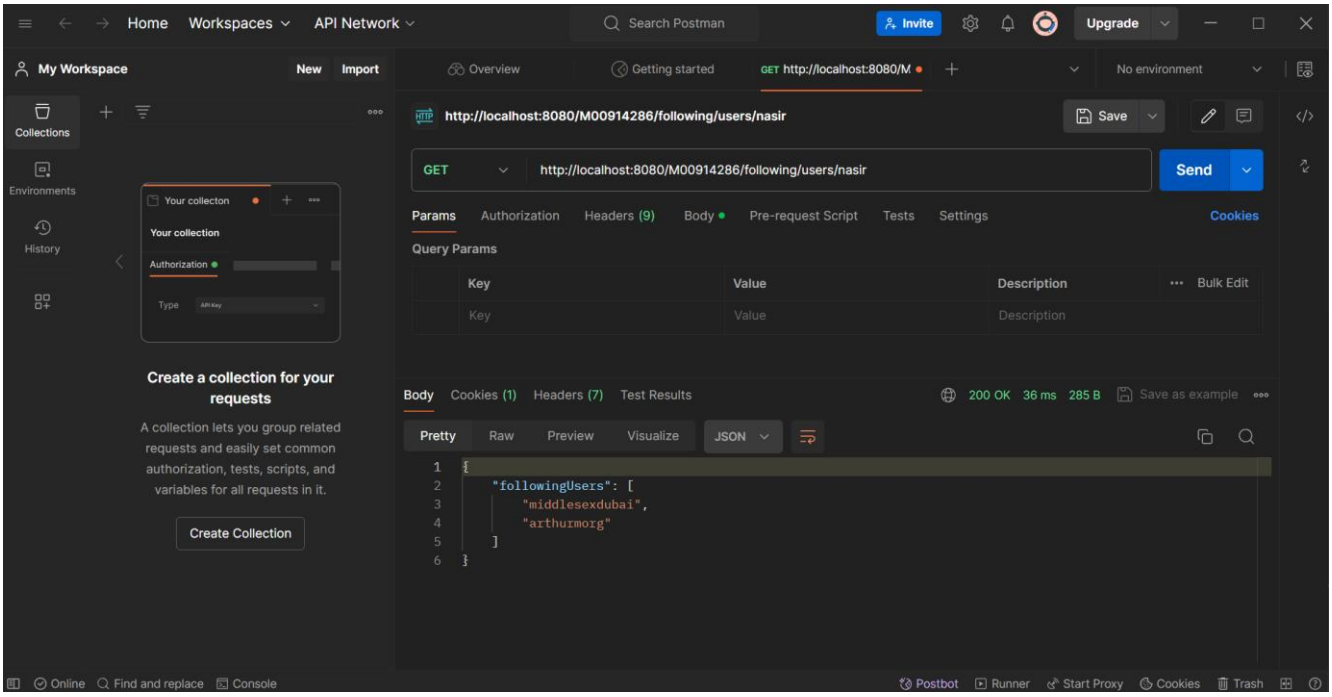
1. **getFollowingCount**
   gets the provided user’s following count (number of people following them).

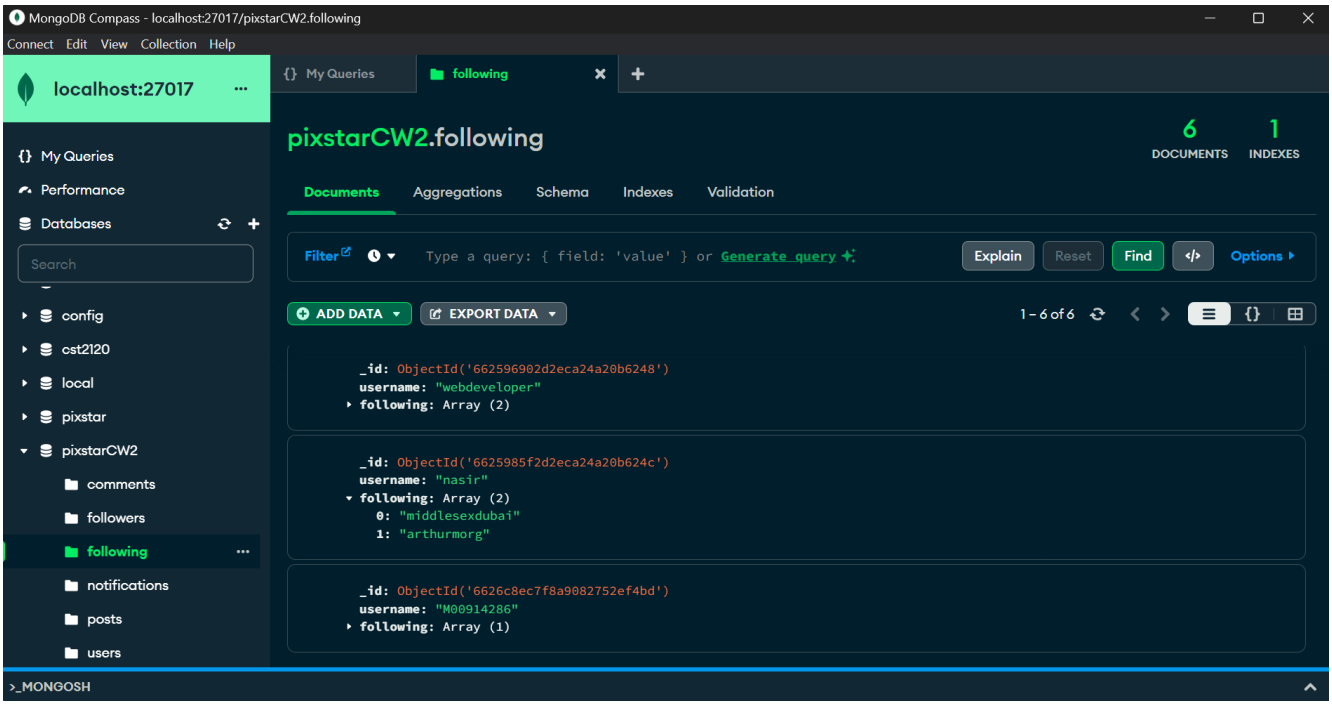




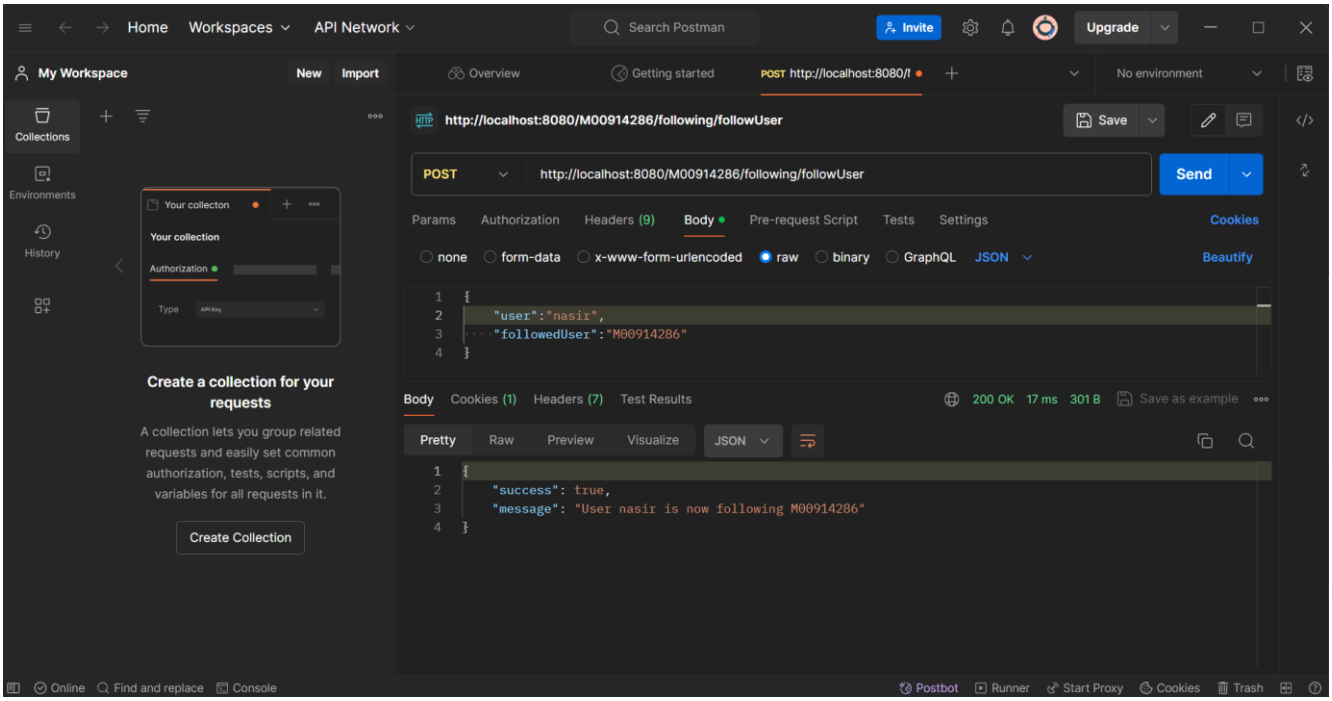
2. getUsersFollowing

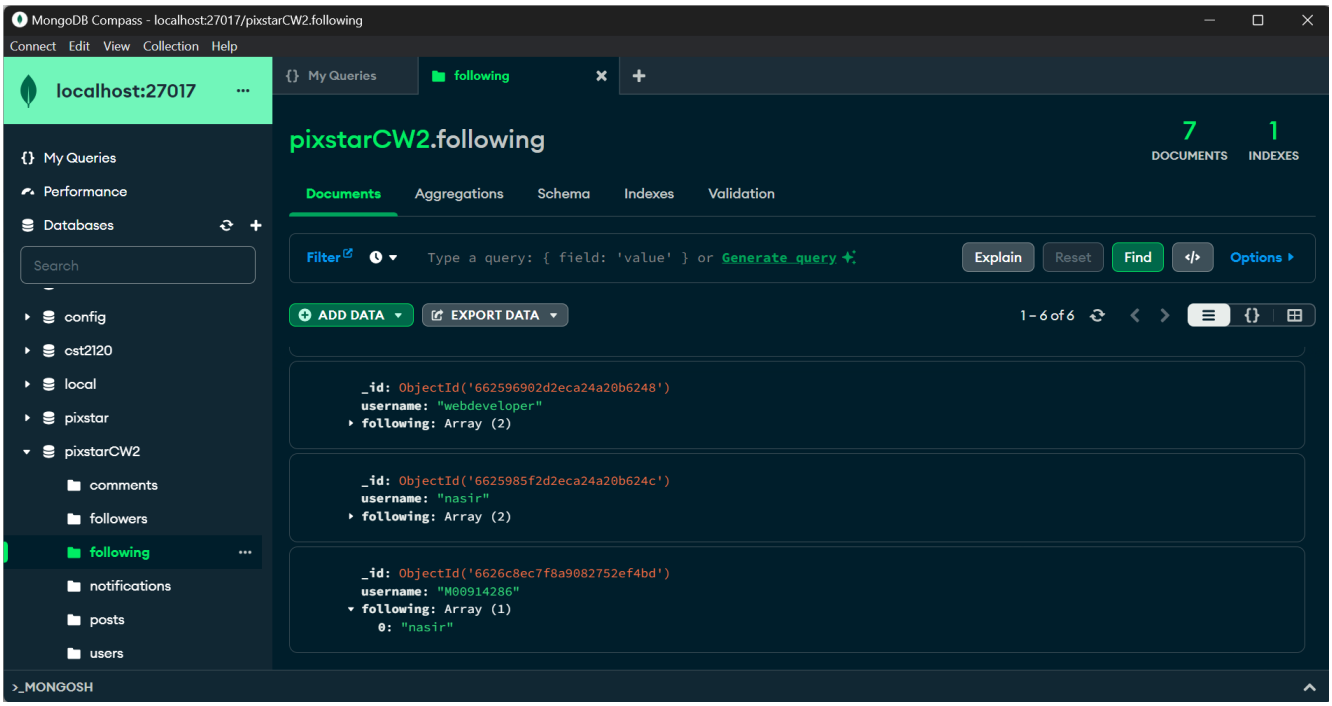
gets the provided users following.





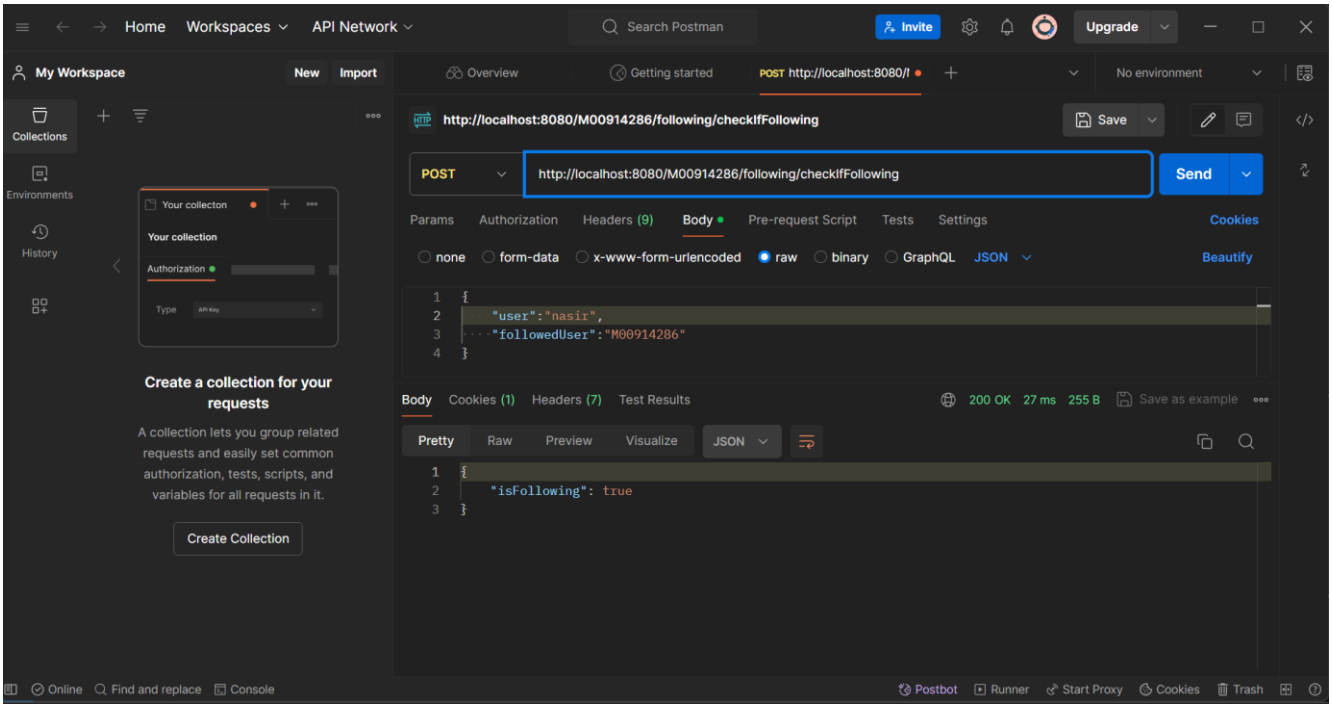
3. **followUser**
   adds a user to another user's following list.

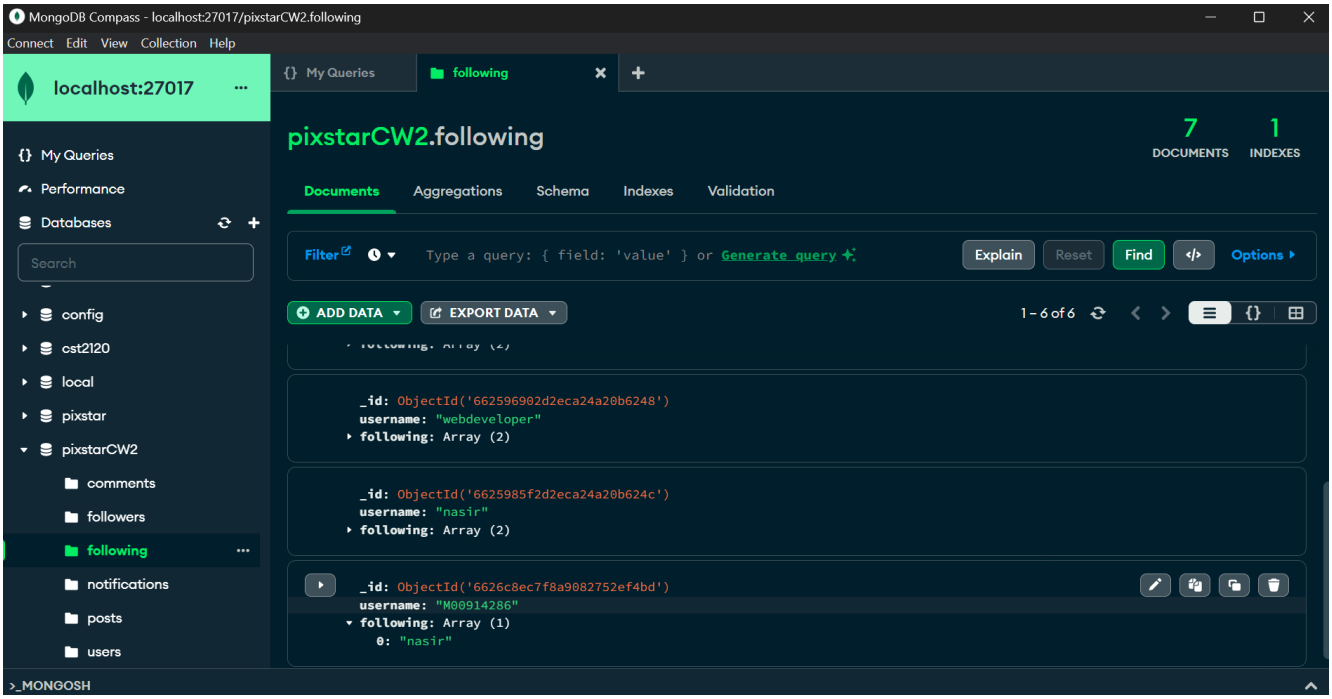




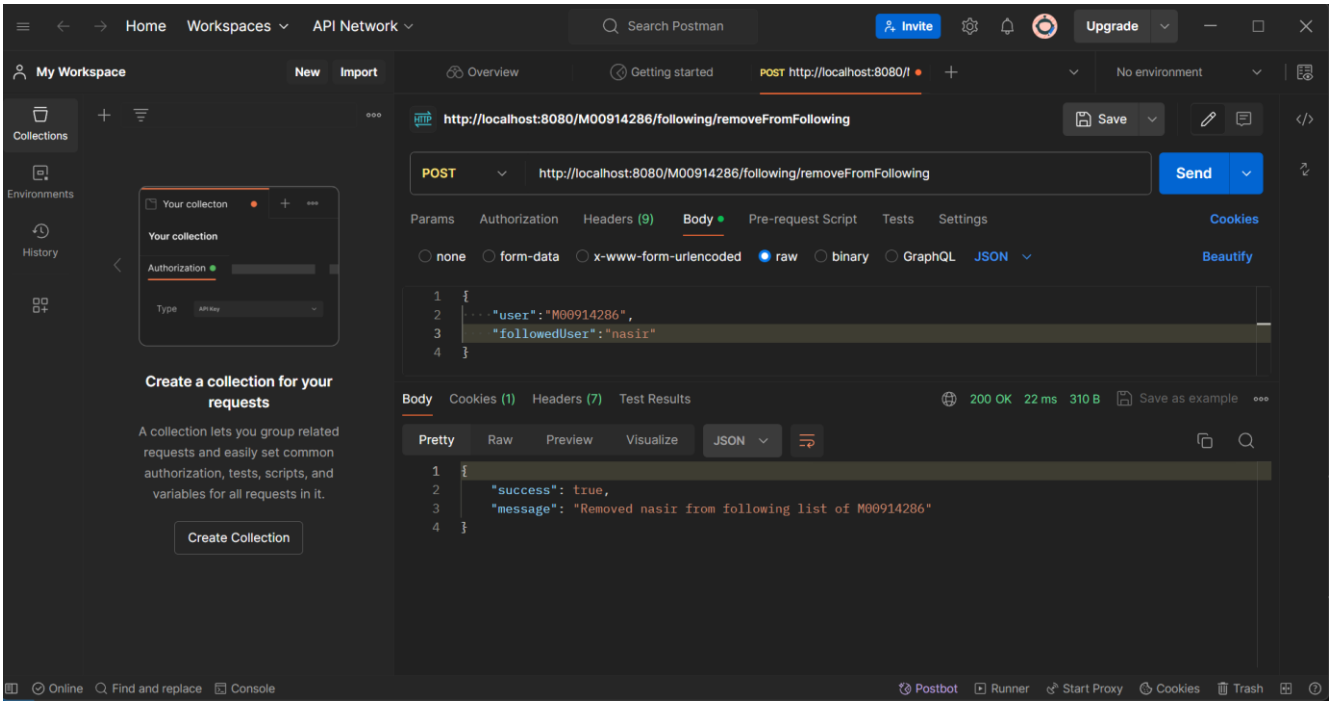
4. checkIfFollowing

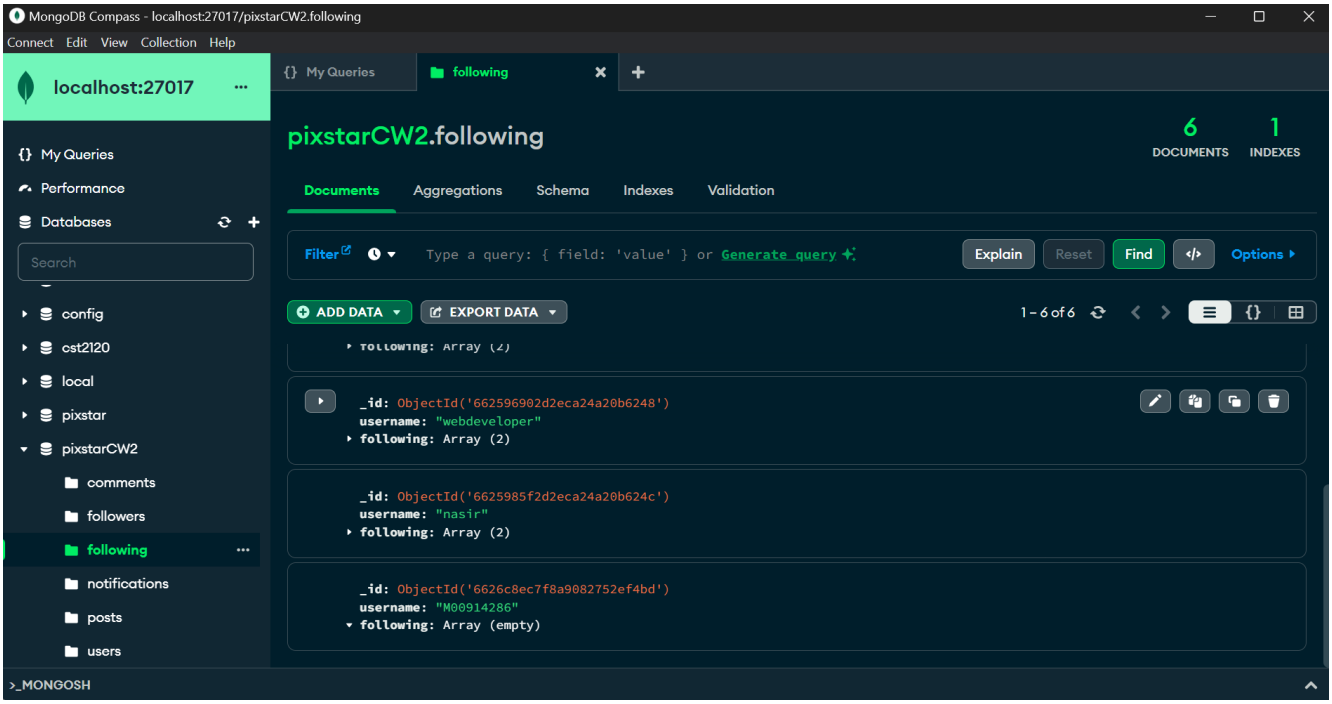
checks if one user is following another, be checking if their in the users following array.





5. **removeFromFollowing**
   removes one user from the following of another.

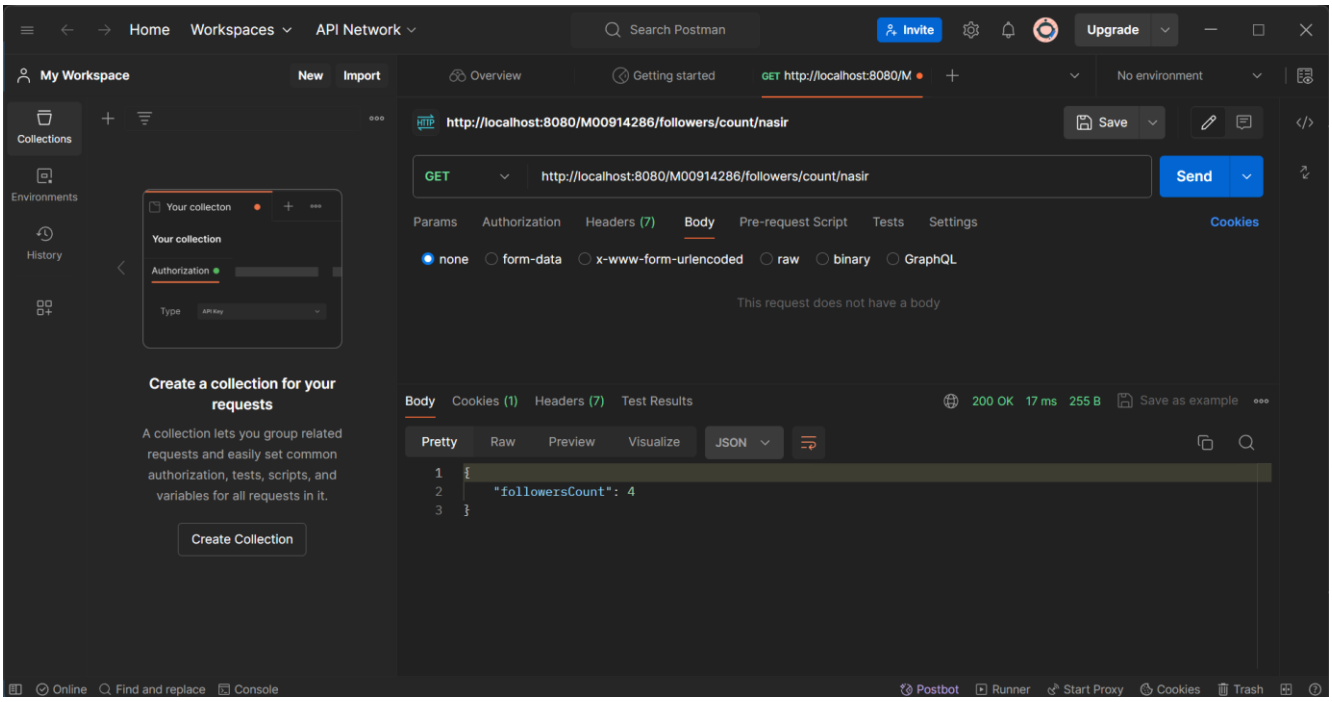

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
POST http://localhost:8080/
No environment
Collections
Environments
History
http://localhost:8080/M00914286/following/removeFromFollowing
Save
POST
http://localhost:8080/M00914286/following/removeFromFollowing
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"user":"M00914286",
"followedUser":"nasir"
}
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Body
Cookies (1)
Headers (7)
Test Results
200 OK
22 ms
310 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"message": "Removed nasir from following list of M00914286"
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash

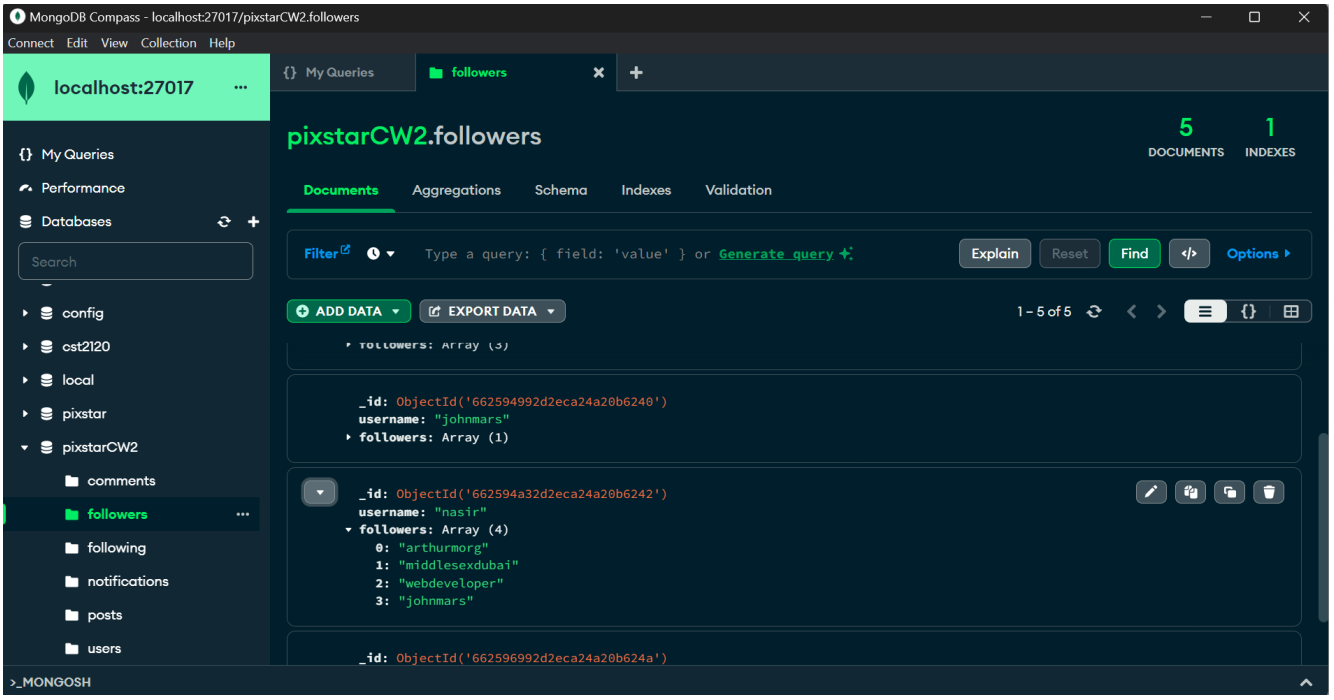

MongoDB Compass - localhost:27017/pixstarCW2.following
Connect
Edit
View
Collection
Help
localhost:27017
My Queries
following
My Queries
Performance
Databases
Search
config
cst2120
local
pixstar
pixstarCW2
comments
followers
following
notifications
posts
users
pixstarCW2.following
6
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 - 6 of 6
_id: ObjectId('662596902d2eca24a20b6248')
username: "webdeveloper"
following: Array (2)
_id: ObjectId('6625985f2d2eca24a20b624c')
username: "nasir"
following: Array (2)
_id: ObjectId('6626c8ec7f8a9082752ef4bd')
username: "M00914286"
following: Array (empty)
>_MONGOSH

Followers Collection

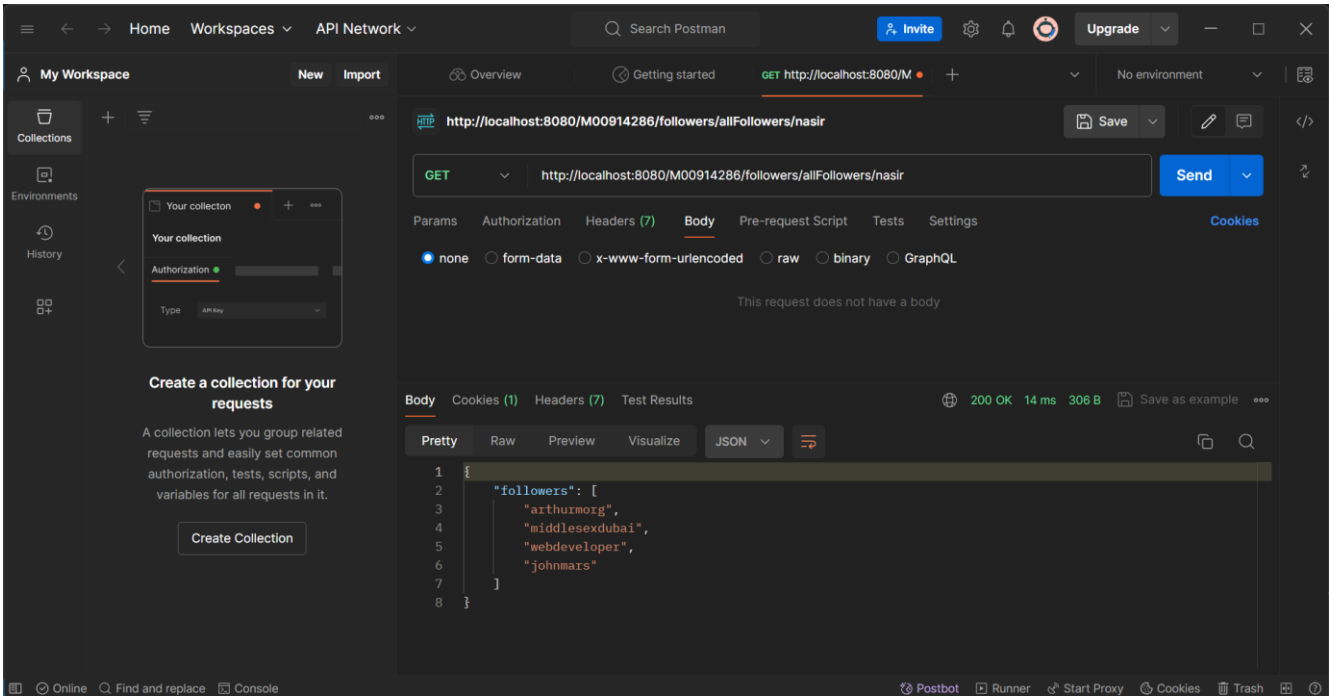
The below webservices are related to the “followers” collection:

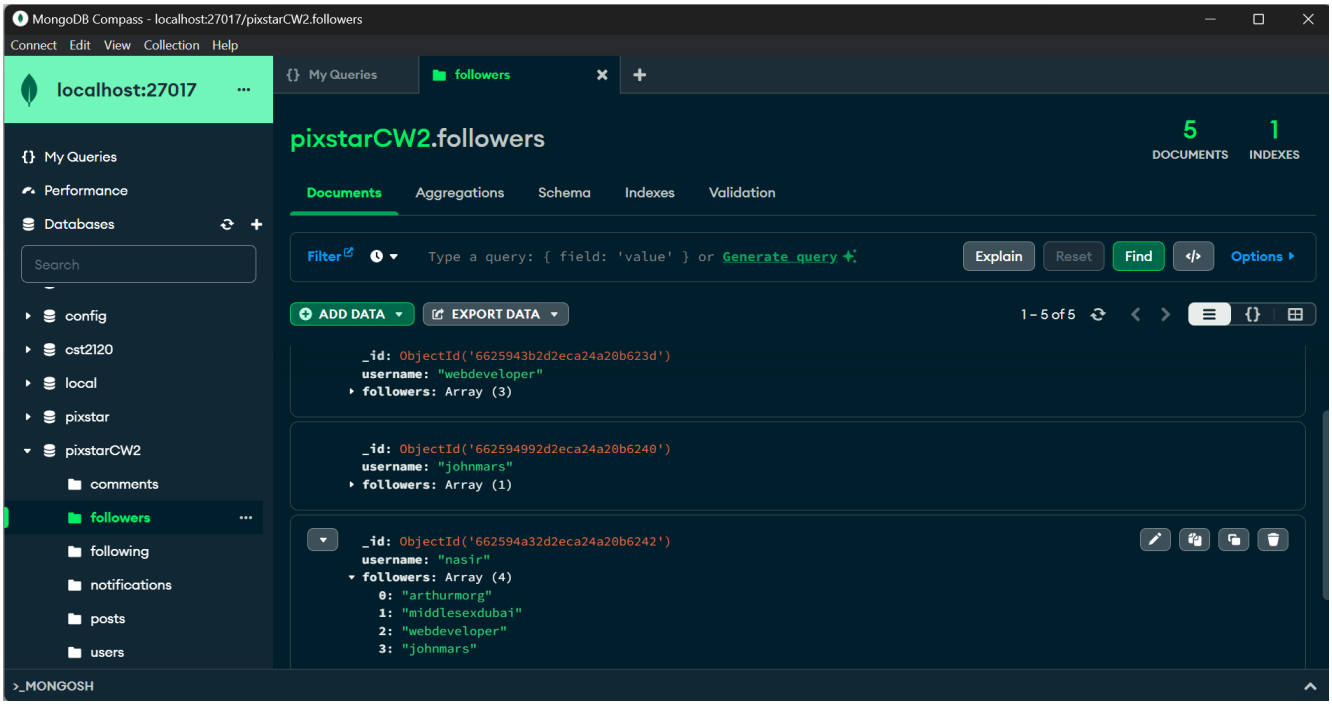
1. **getFollowersCount**
   gets the count of followers for the provided user.



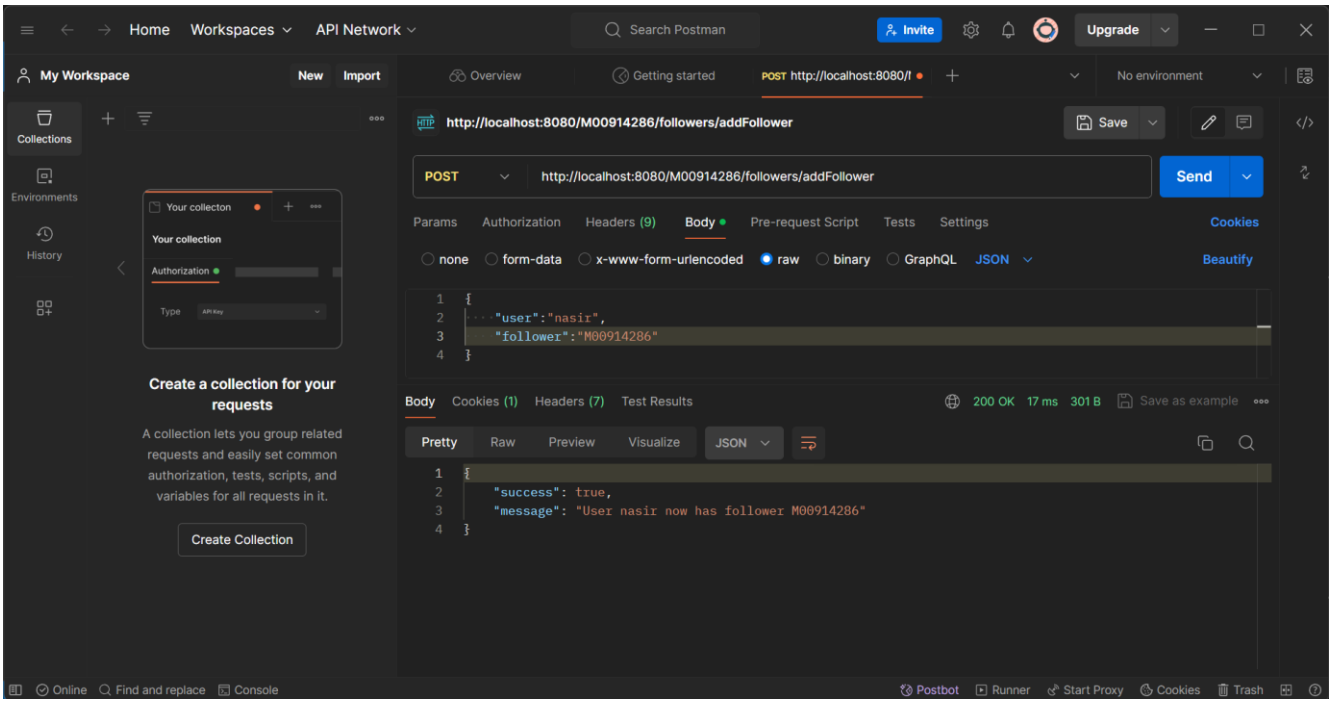


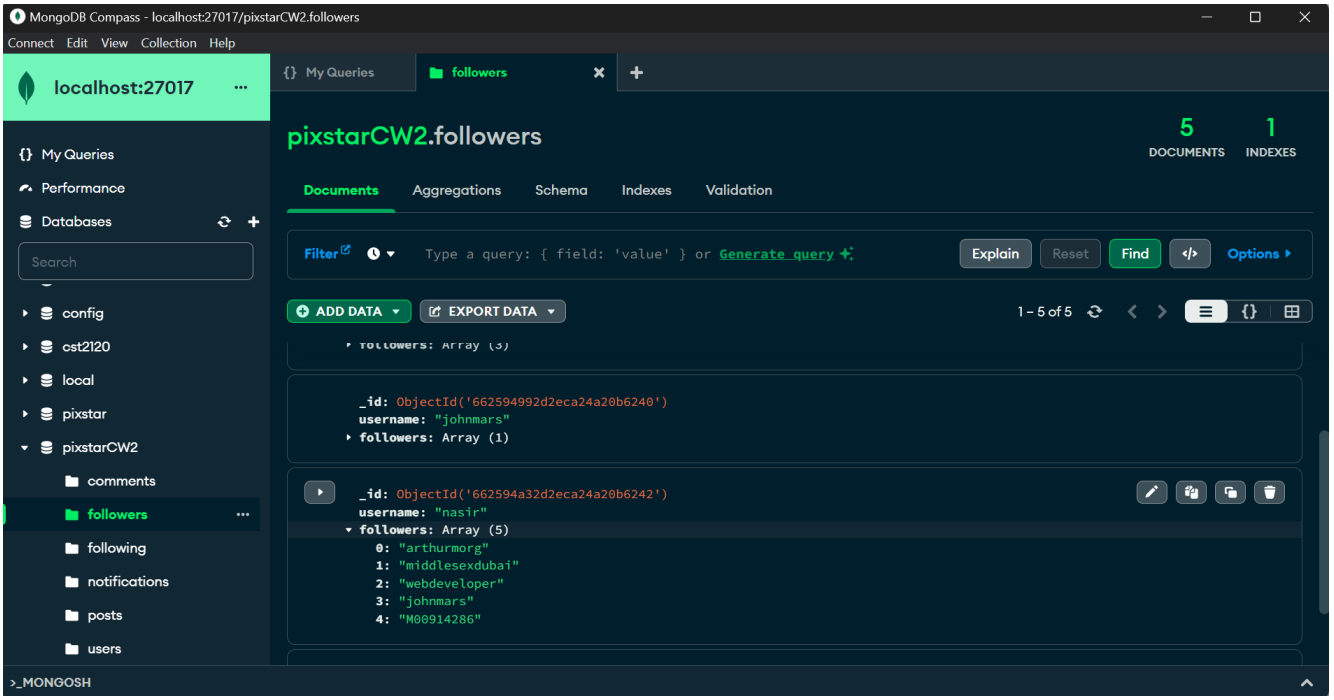
2. **getUserFollowers**
   gets the provided user's followers (username and profile photo) .





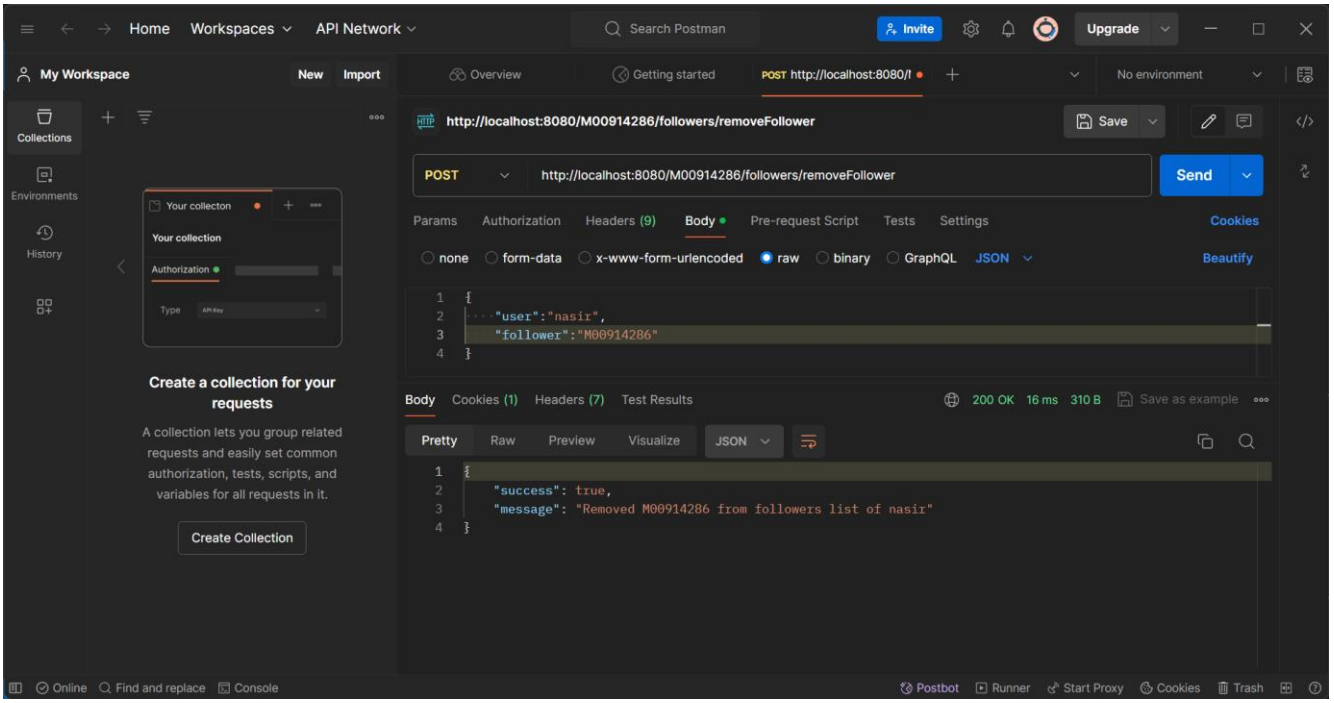
3. **addFollower**
   adds a user to the follower array of another user.

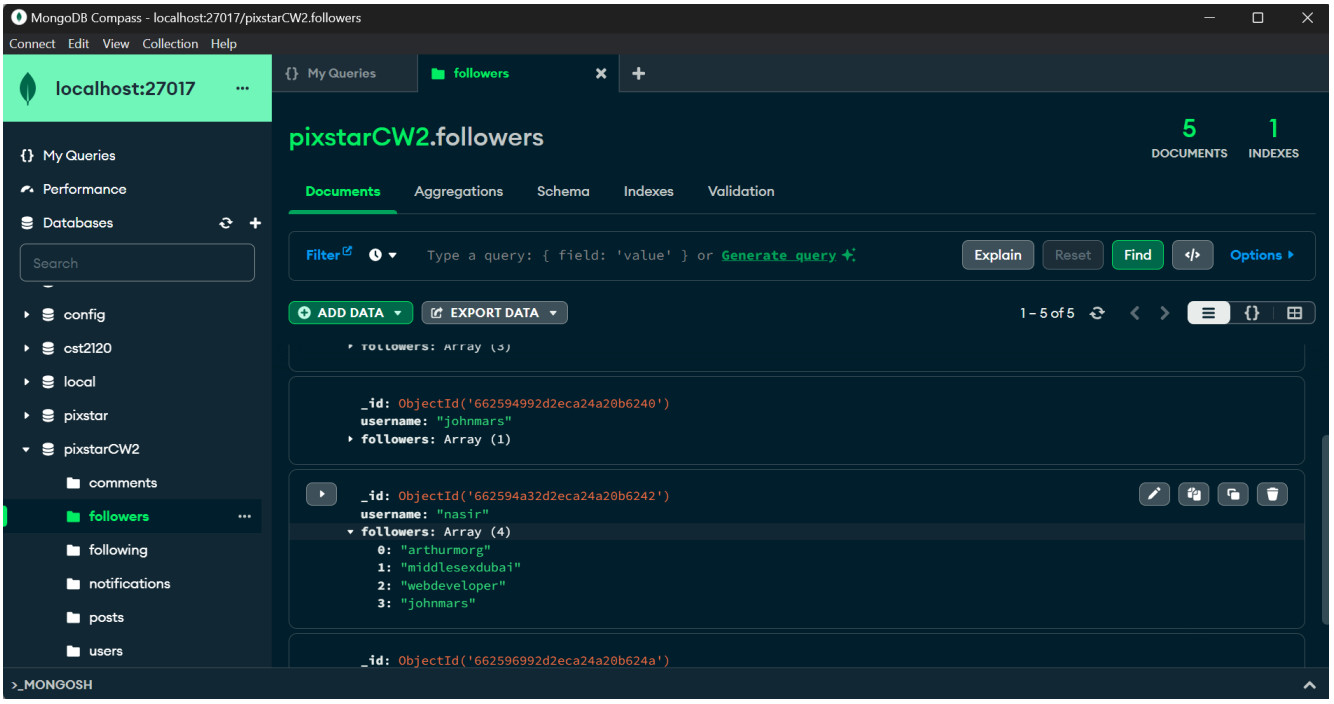




4. removeFollower

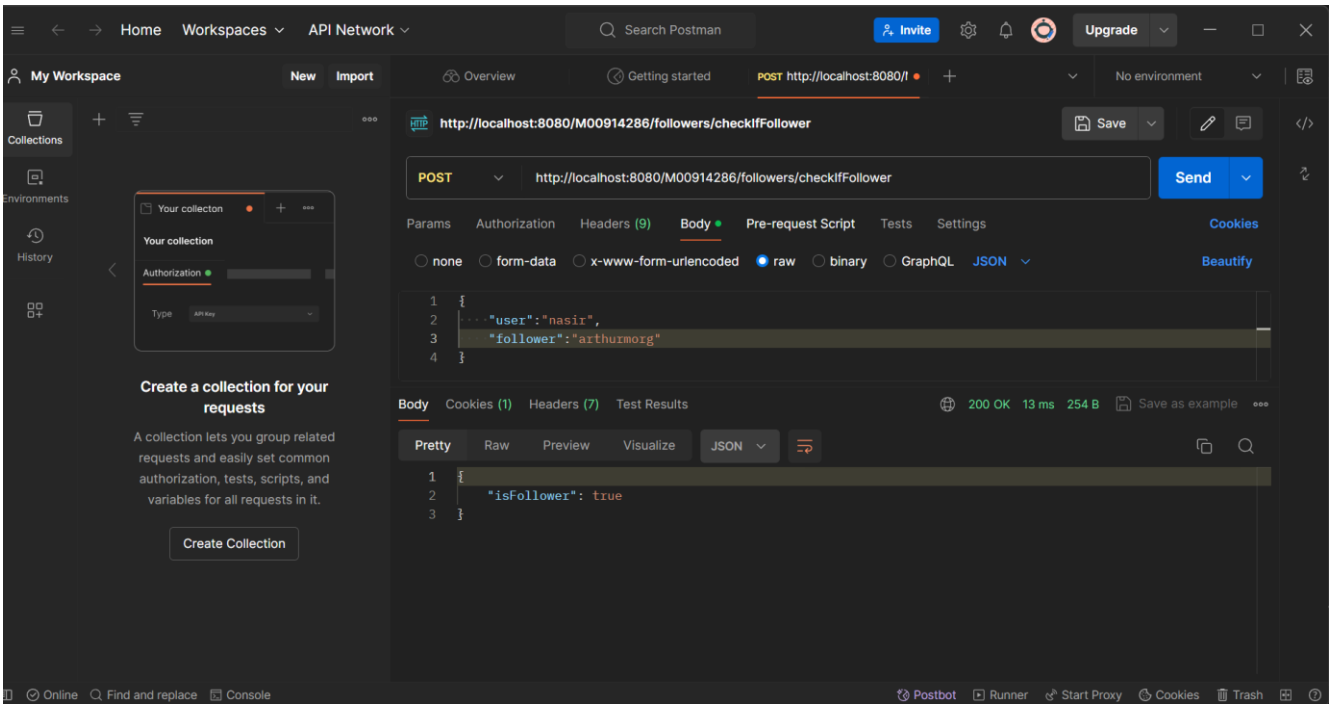
removes the provided follower from the user.

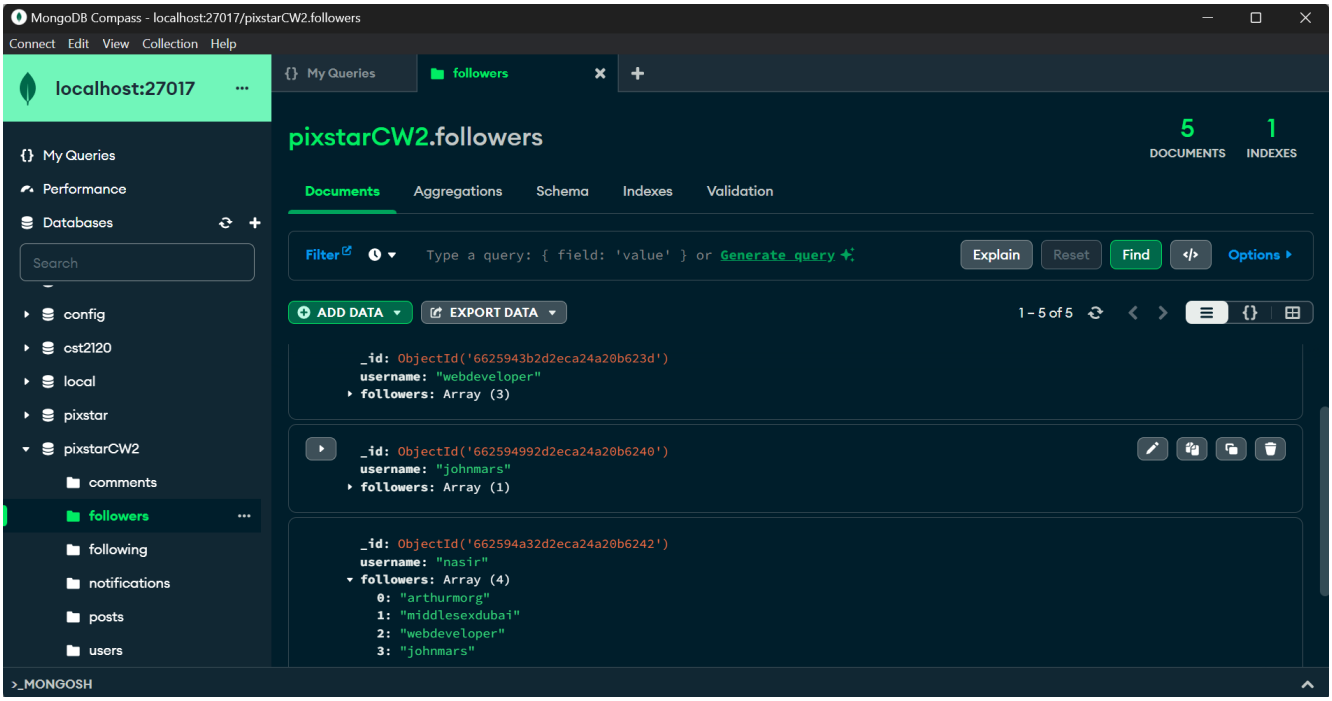




5. checkIfFollower

checks if the provided follower is in the follower array of the user.

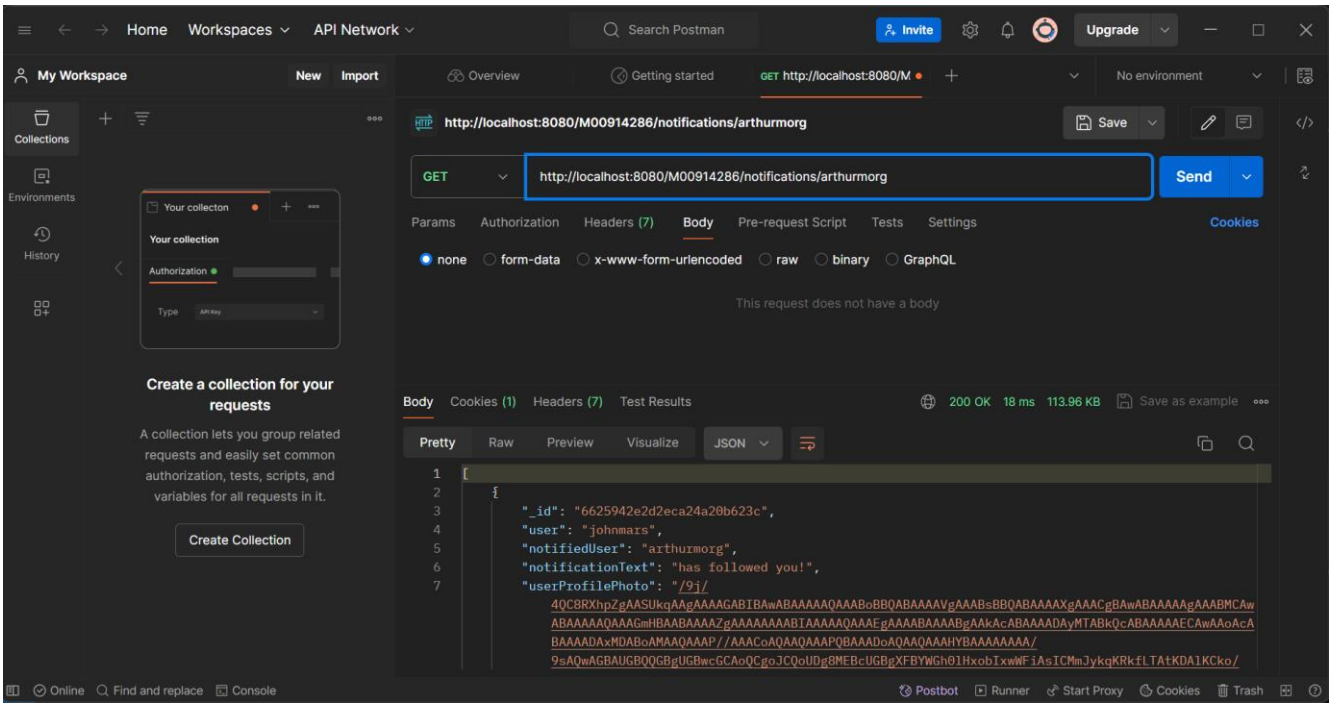


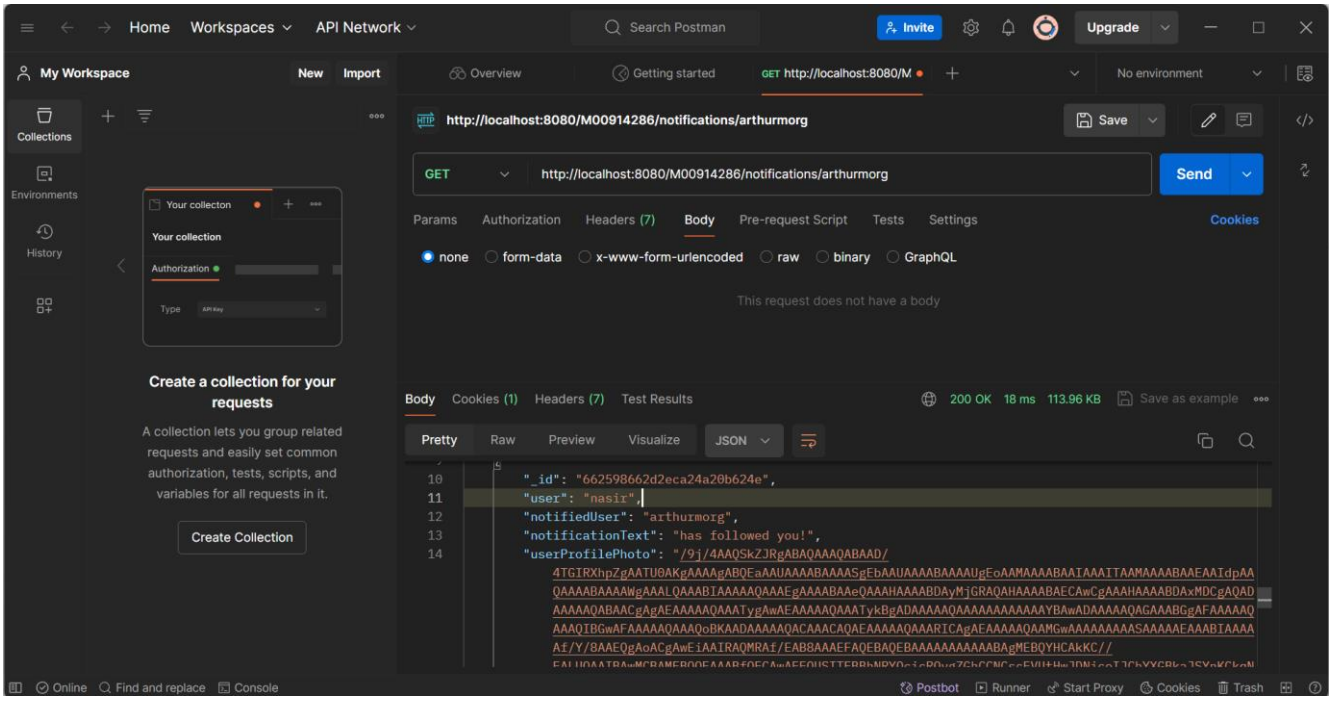

MongoDB Compass - localhost:27017/pixstarCW2.followers
Connect Edit View Collection Help
localhost:27017
My Queries
Performance
Databases
Search
config
cst2120
local
pixstar
pixstarCW2
comments
followers
following
notifications
posts
users
My Queries
followers
pixstarCW2.followers
5 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 – 5 of 5
_id: ObjectId('6625943b2d2eca24a20b623d')
username: "webdeveloper"
followers: Array (3)
_id: ObjectId('662594992d2eca24a20b6240')
username: "johnmars"
followers: Array (1)
_id: ObjectId('662594a32d2eca24a20b6242')
username: "nasir"
followers: Array (4)
0: "arthurmorg"
1: "middlesexdubai"
2: "webdeveloper"
3: "johnmars"
>_MONGOSH

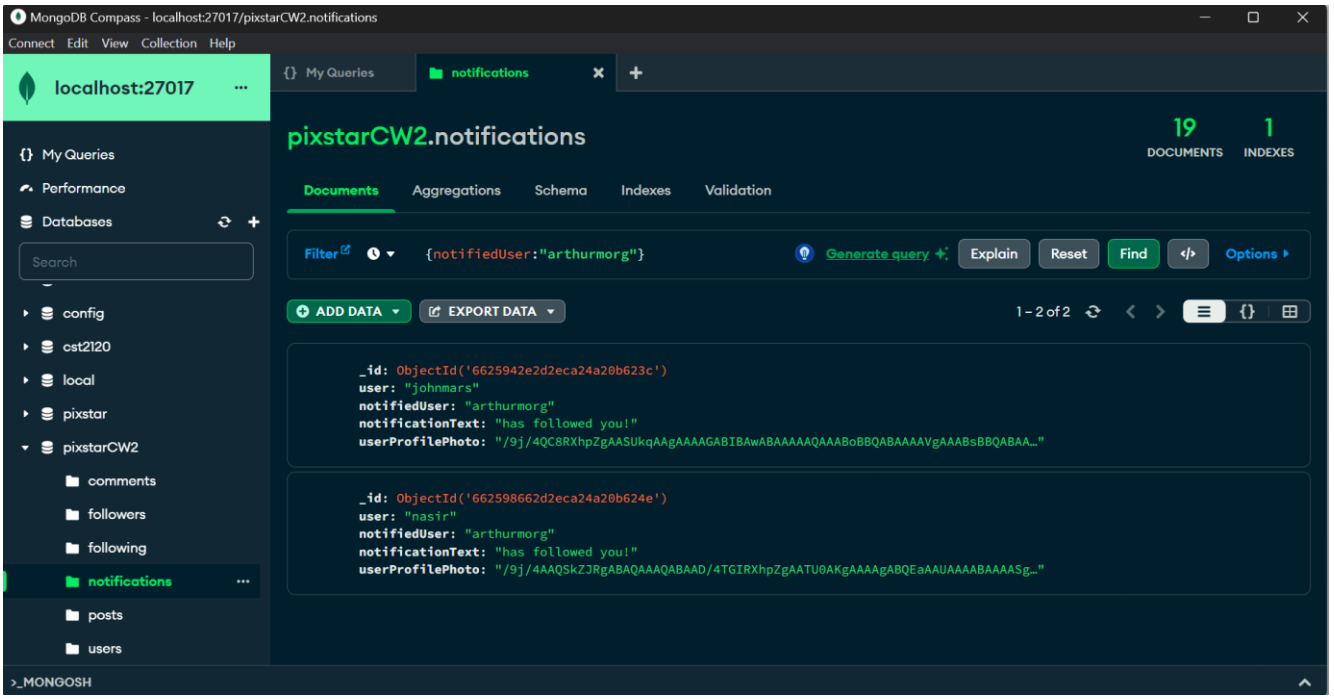
Notifications Collection

The following webservices are related to the notifications collection:

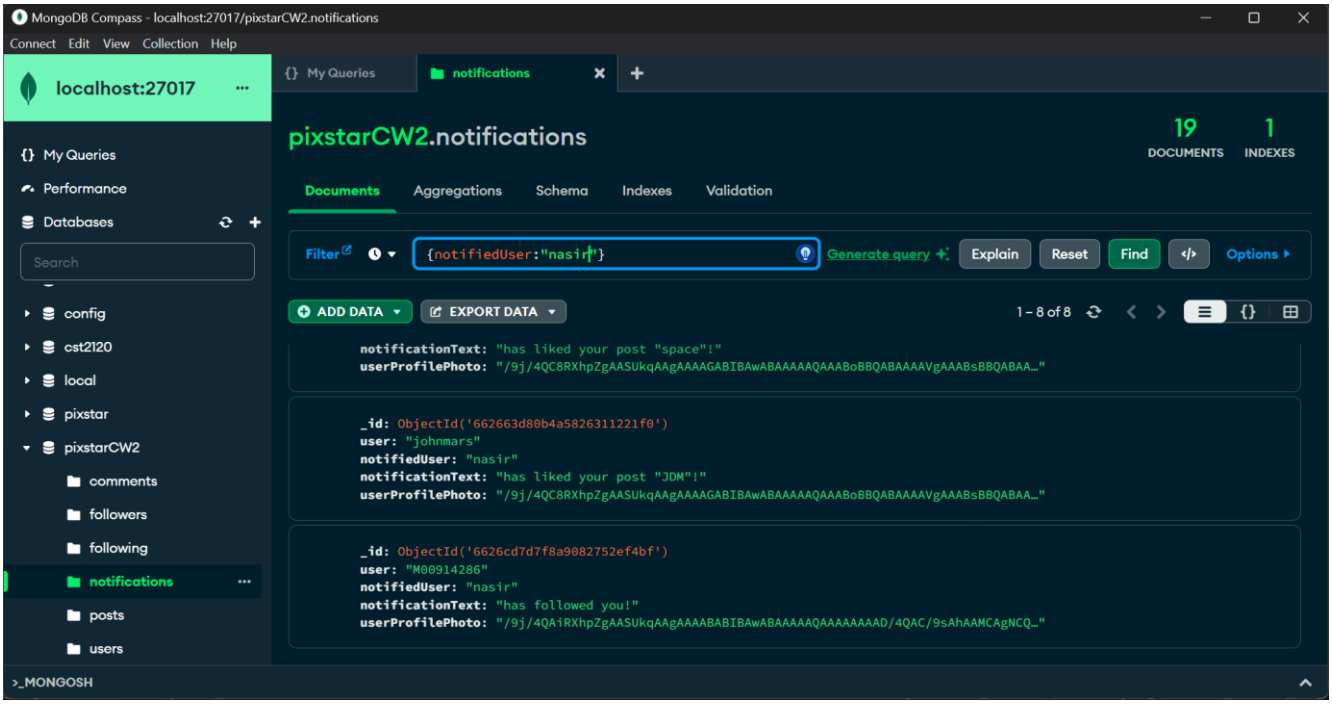
1. **getNotifications**
   gets all the notifications for the provided user.

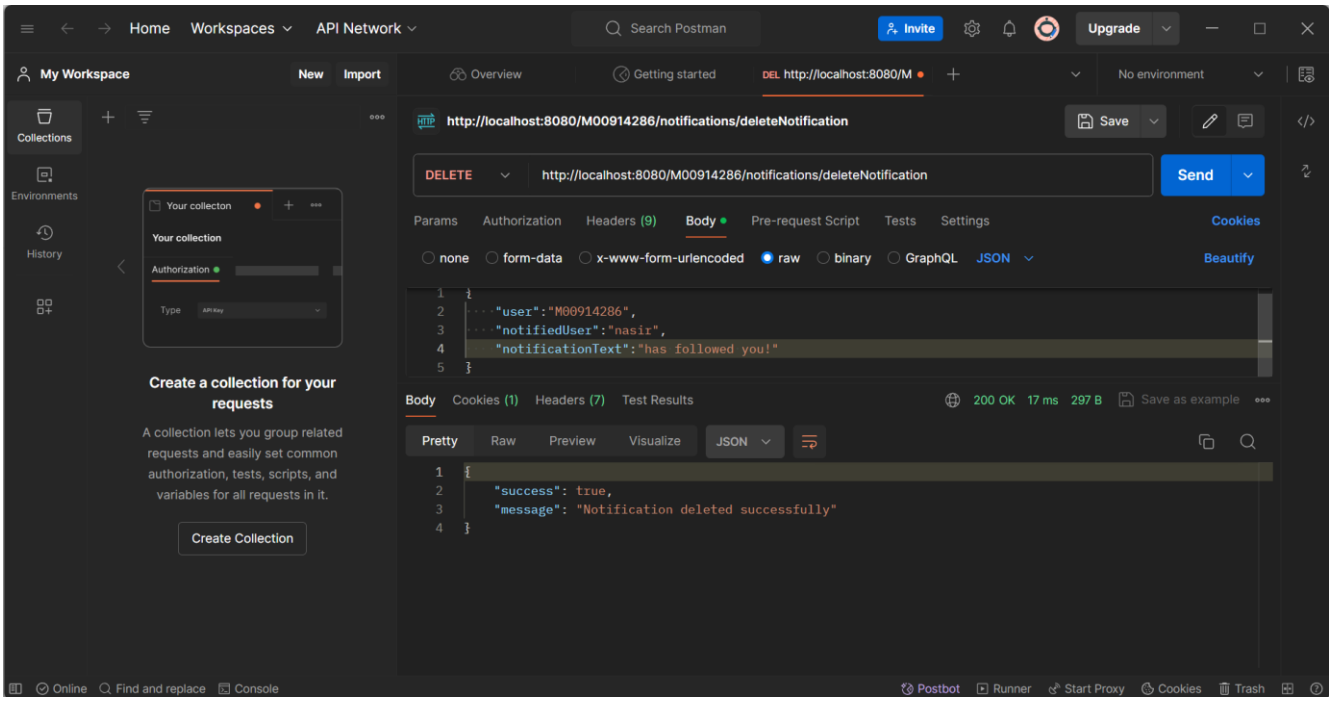


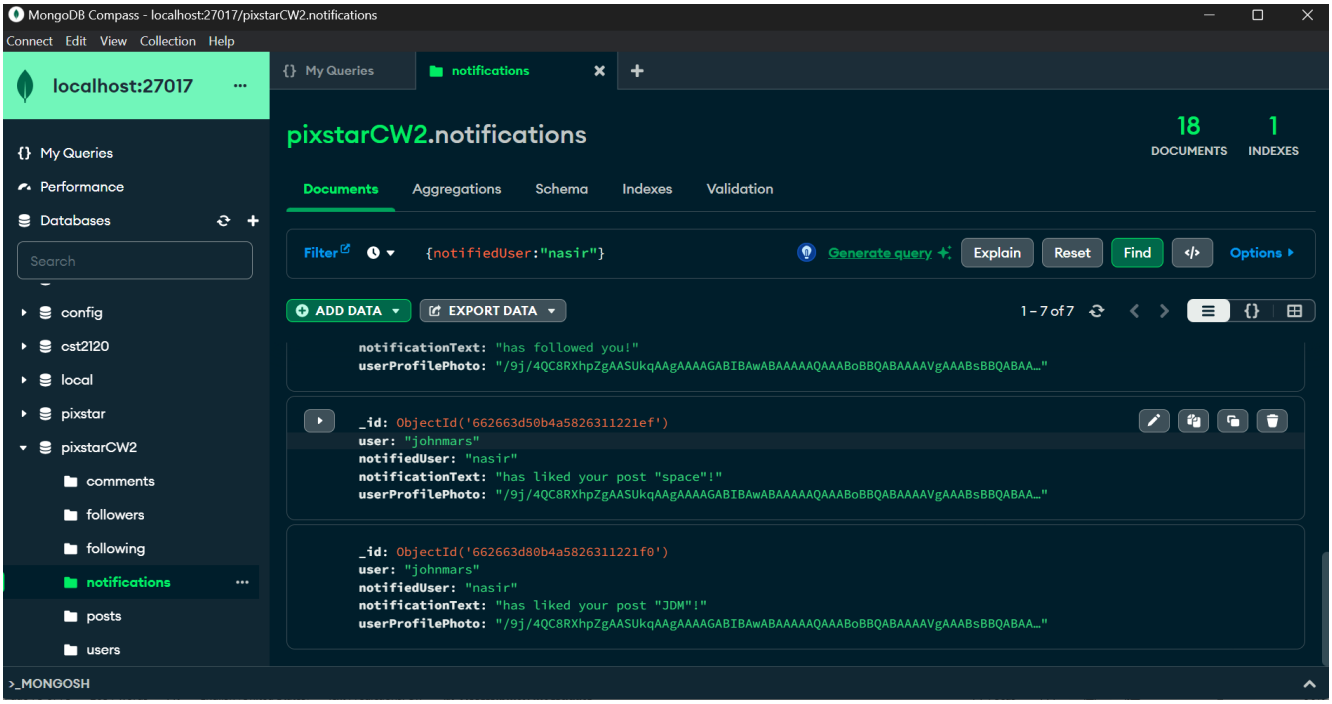




2. **deleteNotification**
   deletes a notification that has the specified user, notifiedUser, and notificationText.


MongoDB Compass - localhost:27017/pixstarCW2.notifications
Connect Edit View Collection Help
localhost:27017
My Queries
Performance
Databases
Search
config
cst2120
local
pixstar
pixstarCW2
comments
followers
following
notifications
posts
users
My Queries
notifications
pixstarCW2.notifications
19 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
{notifiedUser:"nasir"}
Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 – 8 of 8
notificationText: "has liked your post "space"!"
userProfilePhoto: "/9j/4QC8RXhpZgAASUkqAAgAAAAGABIBAwABAAAAAQAAABoBBQABAAAAVgAAABsBBQABAA…"
_id: ObjectId('662663d80b4a5826311221f0')
user: "johnmars"
notifiedUser: "nasir"
notificationText: "has liked your post "JDM"!"
userProfilePhoto: "/9j/4QC8RXhpZgAASUkqAAgAAAAGABIBAwABAAAAAQAAABoBBQABAAAAVgAAABsBBQABAA…"
_id: ObjectId('6626cd7d7f8a9082752ef4bf')
user: "M00914286"
notifiedUser: "nasir"
notificationText: "has followed you!"
userProfilePhoto: "/9j/4QAiRXhpZgAASUkqAAgAAAABABIBAwABAAAAAQAAAAAAAAD/4QAC/9sAhAAMCAgNCQ…"
>_MONGOSH


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
DEL http://localhost:8080/M
No environment
Collections
Environments
History
http://localhost:8080/M00914286/notifications/deleteNotification
Save
DELETE
http://localhost:8080/M00914286/notifications/deleteNotification
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"user":"M00914286",
"notifiedUser":"nasir",
"notificationText":"has followed you!"
}
Body
Cookies (1)
Headers (7)
Test Results
200 OK
17 ms
297 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"message": "Notification deleted successfully"
}
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash



Excluded Webservices

The following webservices have not been documented using postman:

- createNotification
- updateNotificationProfilePhoto
- newNotificationUserProfilePhoto

The first two are involved in the notification creation process, and a successful creation of the notification (with both text and profile photo in base64 format) will indicated the proper working of these services.

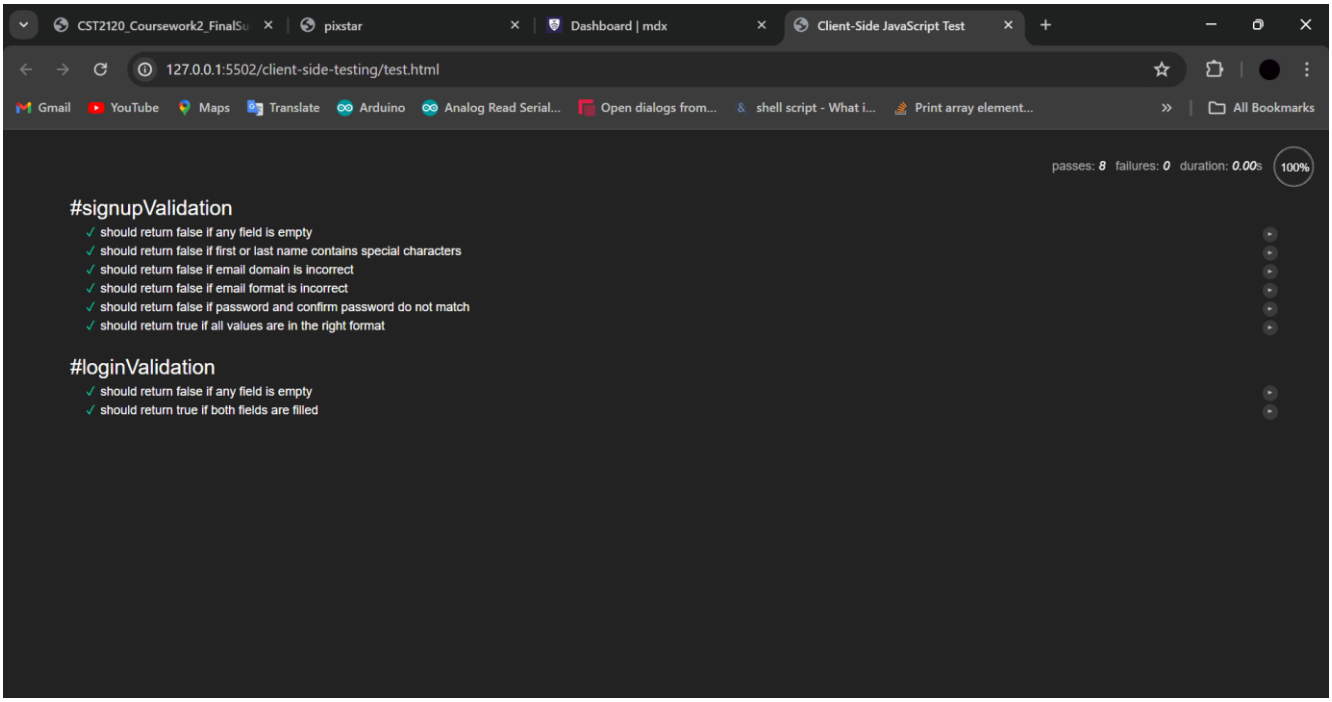
The last of the three is used when the user changes their profile photo by editing it in their account tab, and a change in the profile photo of the notifications in which the user is present would indicate the correct working of this webservice.

This will be covered in the video.

Testing

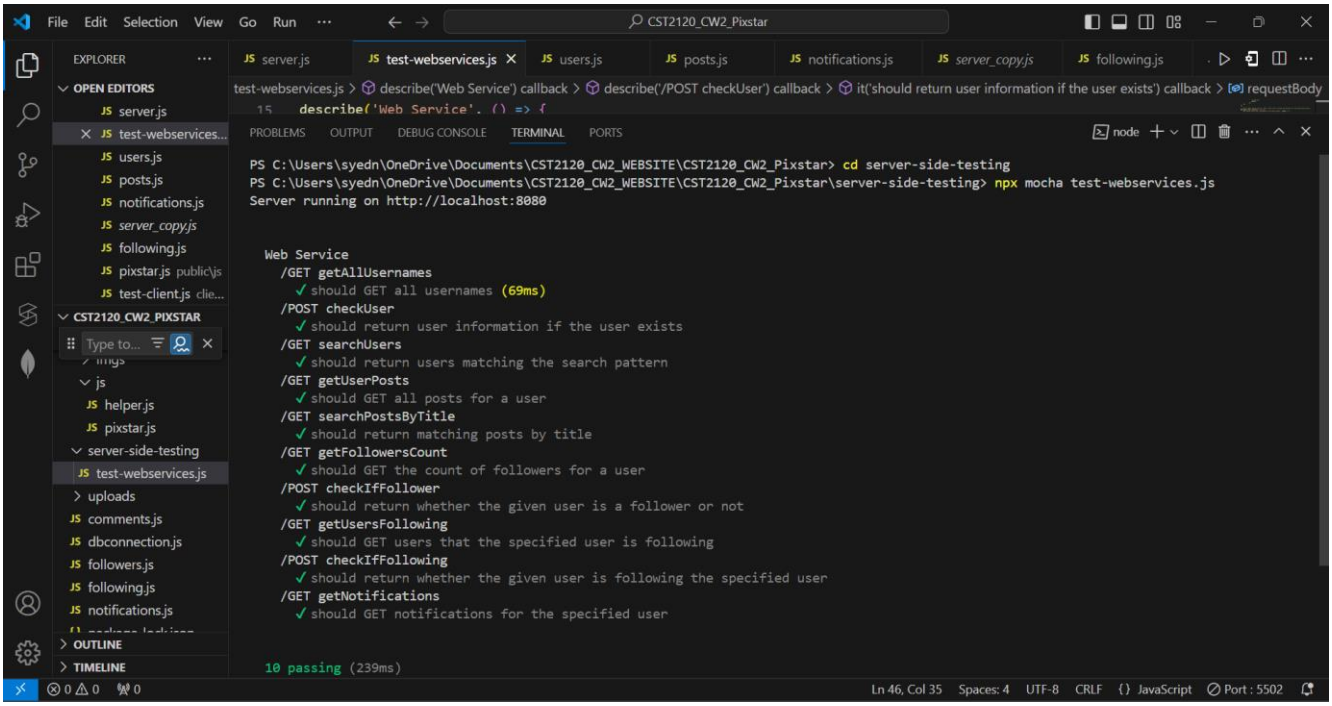
Client-Side Testing

For client-side testing, there are two files test-client.js that contains the tests and test.html that displays their results, I have tested the validation for the sign up and login processes. By opening the test.html file the results are shown as below,



Server-Side Testing

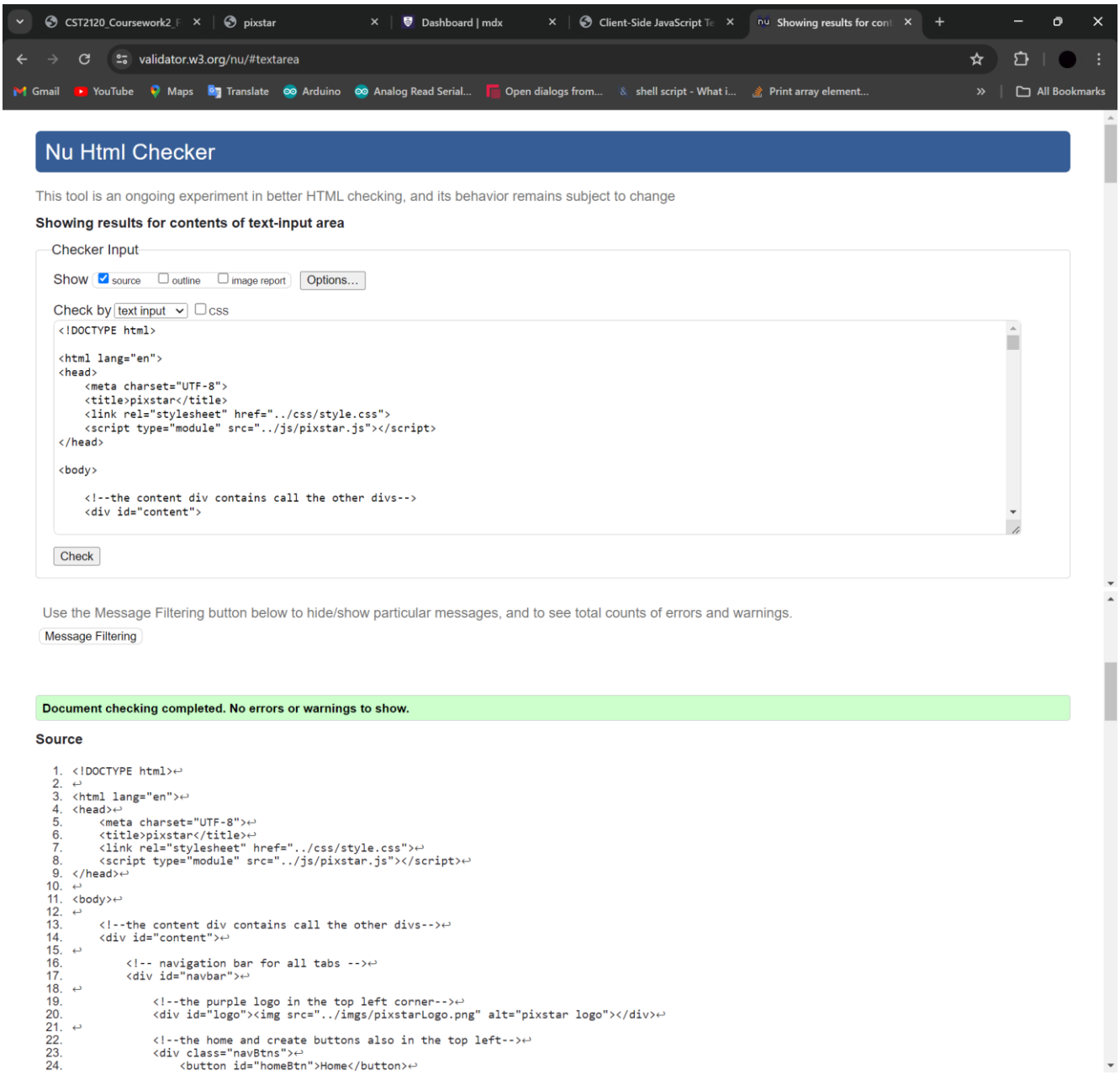
For server-side testing, I have tested ten of my webservices, containing webservices for all the collections. The tests were run in the terminal by first “cd server-side-testing”, and then using the command “npx mocha test-webservices.js”



HTML and CSS Validation

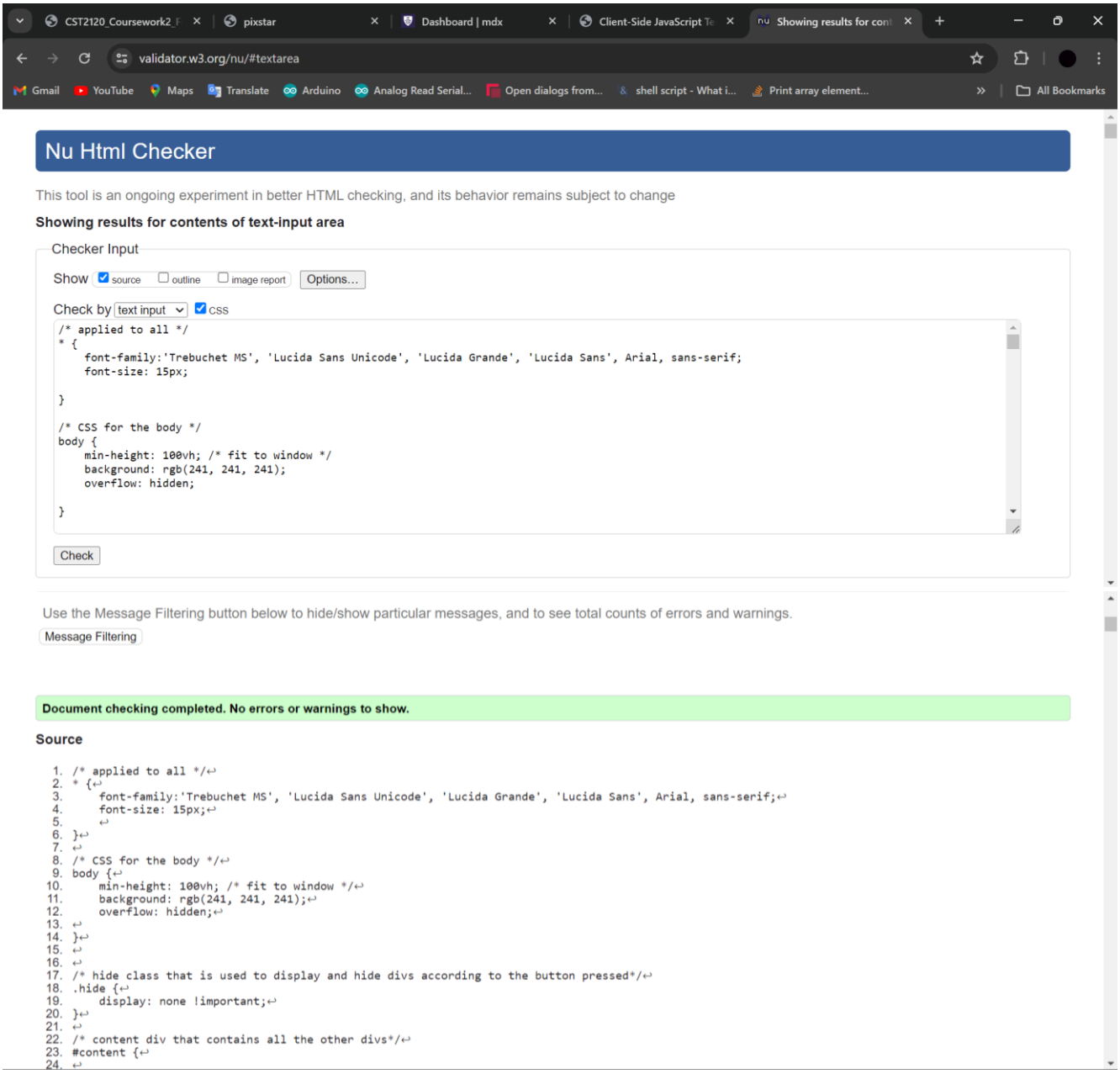
HTML

Attached below is the screenshot of the validation of the index.html page.



CSS

Attached below is the screenshots of the validation for the style.css file.



Security, Legal, and Privacy Issues

Resolved Issues

1. **Password Storage**
   Initially, the passwords were being stored in plaintext format, which poses a huge security risk, this was resolved by storing the password in the following format.

   First, the generateHash function is called in the users.js file, and the password is passed as the argument and the hashed password is returned, password is hashed using SHA-256 this will be used as the key for both the encryption and decryption of the password.

   Then the encrypt function is called passing both (password, hashedPassword), the function takes sixteen characters from the hashedPassword to be used as the initialization vector (IV) which is s a random value used in conjunction with the key for encryption.

   A cipher object is created with the AES-256 encryption algorithm and is initialized with the hashed key and IV.

   The plaintext password is then updated and encrypted.

   The decrypt function works in a similar manned, but converts the encrypted password back to plain text.

2. **Session Security**
   A secret for the session is generated randomly using the crypto library, a random secret for session management enhances security by making it difficult for attackers to predict or manipulate session data.

   The cookie has a max age of twelve hours, and is terminated after the time has expired, this time limit supports both usability and security.

   Session is also terminated as soon as the user logs out.

Unresolved Issues

There are a few issues that are unresolved, they are listed below:

1. **Static Initialization Vector (IV)**
   Using a static IV derived from the hashed password introduces a potential vulnerability. An IV should be unique for each encryption operation to prevent patterns in the ciphertext and mitigate certain cryptographic attacks.

   Generating a random IV for each encryption operation and transmitting it along with the ciphertext would be a much more secure approach.

2. **No Block System**
   Users currently do not have the option to block other users, and so this would mean the users details could potentially be accessed by malicious individuals, that may harass them by commenting on their posts.

3. **No Private Profiles**
   Although the user can remove followers, the following system does not have requests that need to be acknowledged, and so any user can follow another user by just a click of a button and view their posts, every users posts are also displayed in the explore tab.