

Post Buddy

Developer: Sriram Namilakonda

System Manual

This manual provides the detailed information about the postbuddy: A web application for sharing the text posts

INDEX

Serial Number	Topic	Page Number
1	Introduction to the system	3
2	Technology Stack	4
3	Database overview	5
4	UI overview	7
5	API overview	14
6	Containerization using docker	17
7	Procedure for the system setup	18
8	Flow diagram	19

Introduction to the system

The “**PostBuddy**” web application is a text posts sharing platform similar to Reddit and Hackernews.

Functionality for all the users

- A user can view all the posts and the comments associated with it
- A user can register into the system through sign up facility
- A user, if already registered, can login to the system

Functionality for the registered users

- A user can write a post
- A user can edit the posts which were written by self, previously, but the posts cannot be deleted
- A user can comment on any of the posts, including the self-written posts and the user can delete his/her own comments on any of the posts
- A user can only view the posts or comments written by other users
- A user can logout of the system. After that, the user can only view any posts or comments.

Technology Stack

Database: MongoDB

UI development

- ReactJS
- open-source libraries used: react-router-dom, axios, @material-ui/core, @material-ui/lab, @material-ui/icons
- node version>12 npm version>6. (node package manager)
- IDE used for UI development: Visual studio code

API development

- NodeJs
- Framework: Express
- Mongoose: MongoDB object modelling tool
- IDE used for API development: Visual Studio Code
- Tool to test the API calls: Postman
- Other libraries used: cors, nodemon, dotenv, body-parser

Containerization Tool: Docker

Database

The database is created in the local system using MongoDB. The database export Json files and the database related information file is shared in the database folder attached along with the manual

Database Name: postbuddy

Host: localhost:27017 **Cluster:** Standalone

Edition: MongoDB 4.4.2 Community

Collections:

Collections						
CREATE COLLECTION						
Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
comments	4	112.5 B	450.0 B	1	36.0 KB	
posts	7	155.6 B	1.1 KB	1	36.0 KB	
users	4	152.3 B	609.0 B	1	36.0 KB	

Metadata for the collections:

1. users

Field	Type	Description
_id	Object	This field serves as a unique identifier of a user document. This field is automatically generated by MongoDB
username	String	This field stores the username of the registered user. Typically, every user has a unique username
password	String	This field stores the password of user account in the system. This field along with the username helps users to login to the system
firstname	String	This field stores the first name of the registered user
lastname	String	This field stores the last name of the registered user
email	String	This field stores the mail address of the registered user. Unlike other fields, this field is an optional field

2. posts

Field	Type	Description
_id	Integer	This field serves as a unique identifier of a post document. A unique identifier is generated by the system through randomization
postedby	String	This field stores the username of the user who has written the post
content	String	This field stores the content of the post
commentcount	Integer	This field stores the number of comments received by the post
postdate	String	This field stores the date on which the post has been created in the string format

3. comments

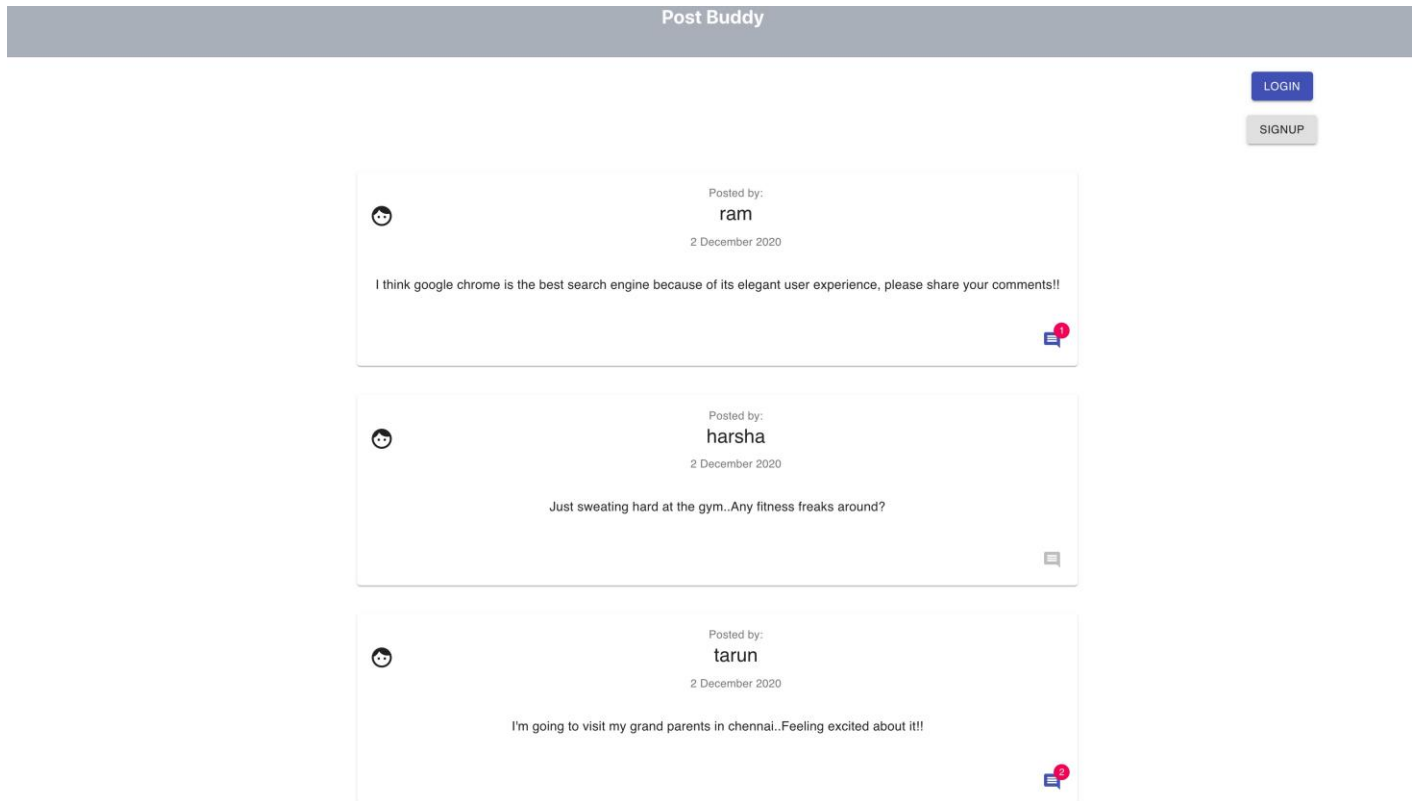
Field	Type	Description
_id	Integer	This field serves as a unique identifier of a comment document. A unique identifier is generated by the system through randomization
postedby	String	This field stores the username of the user who has written the comment
content	String	This field stores the content of the comment
postid	Integer	This field stores the identifier of the post to which this comment is associated to.

UI Overview

Port number: 3000

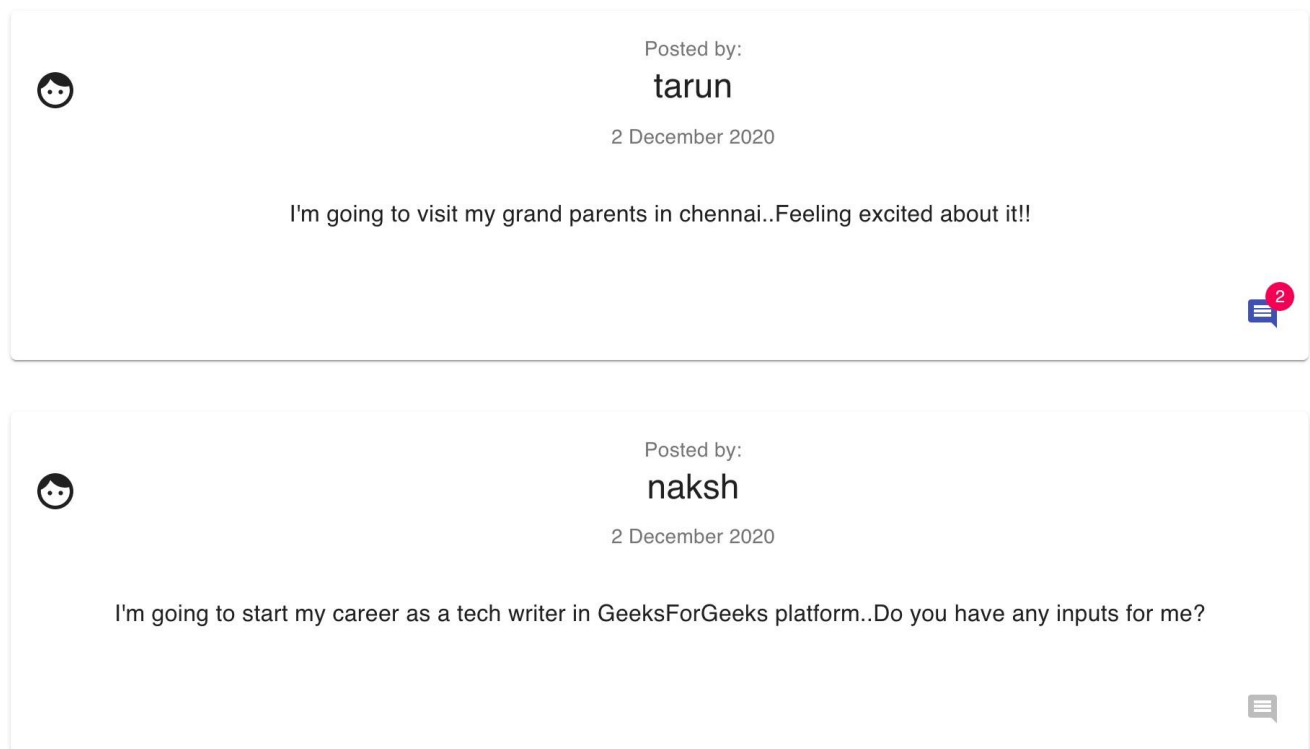
UI endpoint: <http://localhost:3000/>

1. Home Page



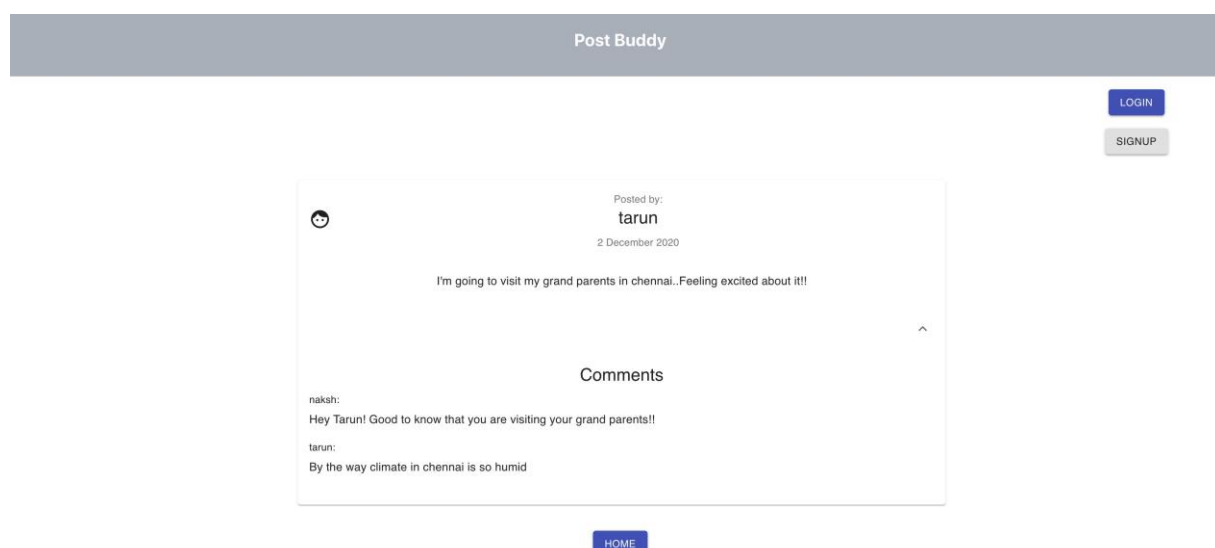
This is the home page of 'Post Buddy' web application that displays all the posts written by the users thus far and it also contains log in or sign-up options where existing user can login to the system and new user can register themselves in the system accordingly

2. A post view in home page



In the home page, a post view is as depicted. The number of comments for the post is displayed bottom right. As the user in this scenario is not logged in, the user can only view the comments for a post if exists. For the posts with no comments, the button will be disabled.

3. A detailed view of the post



Upon clicking the comment icon present at the bottom-right position of a post, the post is displayed explicitly as depicted along with the comments associated to it. The comments section supports the expand and collapse functionality in this page. In this scenario, the user is not logged in to the system. So, the comments and the post can only be viewed by the user.

4. The login page

Post Buddy

Login Form

Username: *

Password: *

SUBMIT

New to Post Buddy?

SIGN UP

HOME

Form Info

* indicates the **mandatory** fields

Upon clicking the login button on the home page. The user will be redirected to the system's login page, Where the user can provide username and password to get into the system. If the user realizes that he/she needs to register, then they can be redirected to sign-up page or if the user wants to go back to home page, it can also be done by clicking on the corresponding buttons

5. The login alert for erroneous credentials

Login Form

Username: *
ram

Password: *

SUBMIT

New to Post Buddy? [SIGN UP](#)

[HOME](#)

Form Info
* indicates the **mandatory** fields

Either username or password is invalid!

If the username and password credentials entered in login form is found to be invalid, then the alert message is displayed as follows.

6. The signup page

Signup Form

Username: *
ram

First Name: *
Sriram

Last Name: *
Namilakonda

Password: *

Re-enter Password: *

Email:

SUBMIT

Already have an account? [LOGIN](#)

[HOME](#)

Form Info
* indicates the **mandatory** fields

The user with the given username already exists!

The signup form helps the users to register into the system. It makes sure that the username of a user to be unique, if it's found out that the entered username already exists, then the alert message is displayed as depicted. Among all the fields specified, only the email field is an optional field.

7. The signup page password mismatch

Post Buddy

Signup Form

Username: *

ram

First Name: *

Sriram

Last Name: *

Namilakonda

Password: *

..

Re-enter Password: *

Email:

SUBMIT

Already have an account?

LOGIN

HOME

Form Info

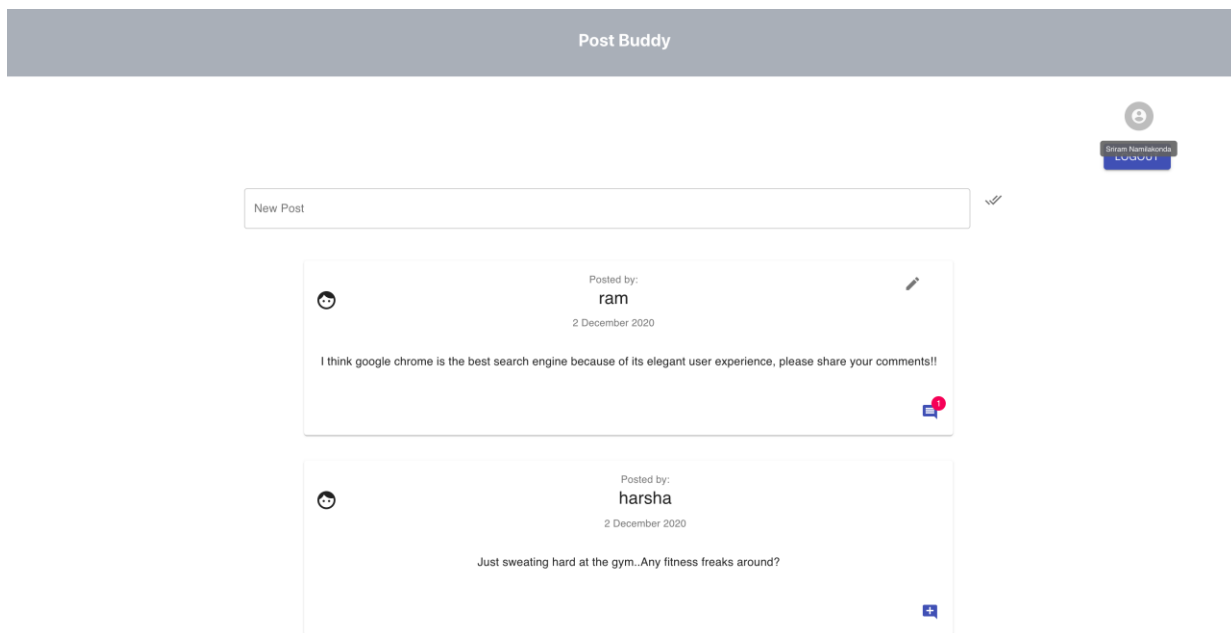
* indicates the **mandatory** fields

The password mismatch has occurred!

In the sign-up page, the user will be asked to enter the password twice for the obvious reasons. If a mismatch is found in the values in the corresponding fields, the alert message will be displayed for password mismatch as depicted.

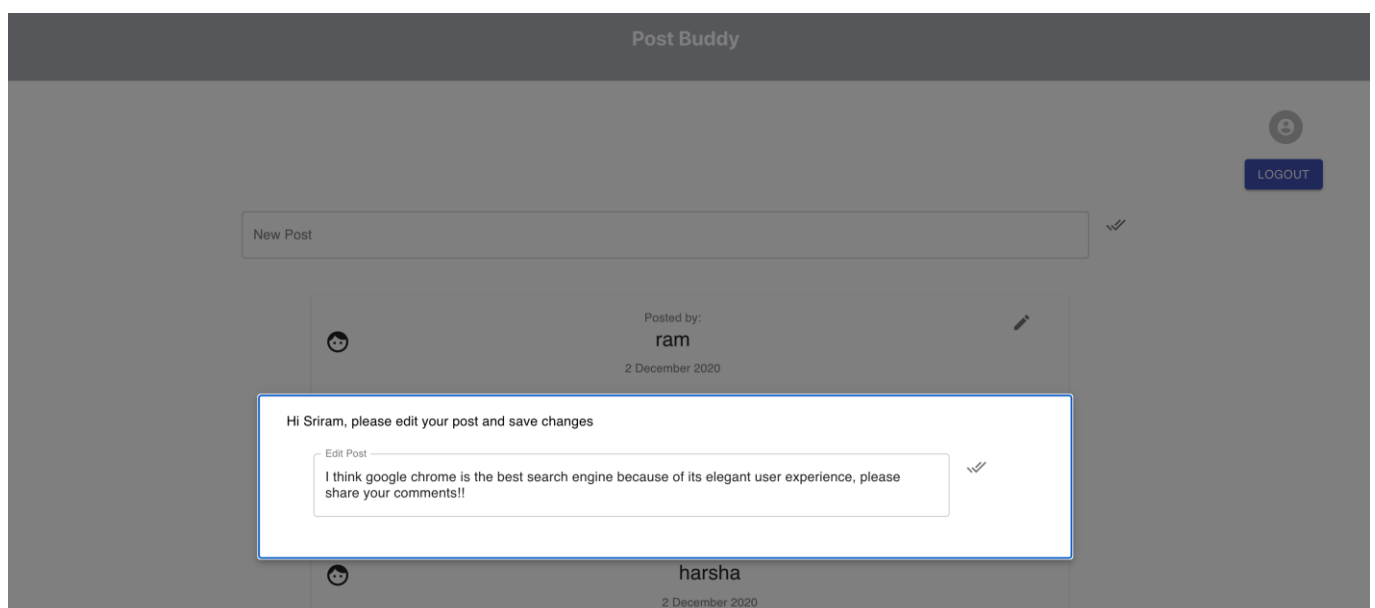
Note: None of the mandatory fields can be left empty in both login and signup page

8. The home page when user logged in



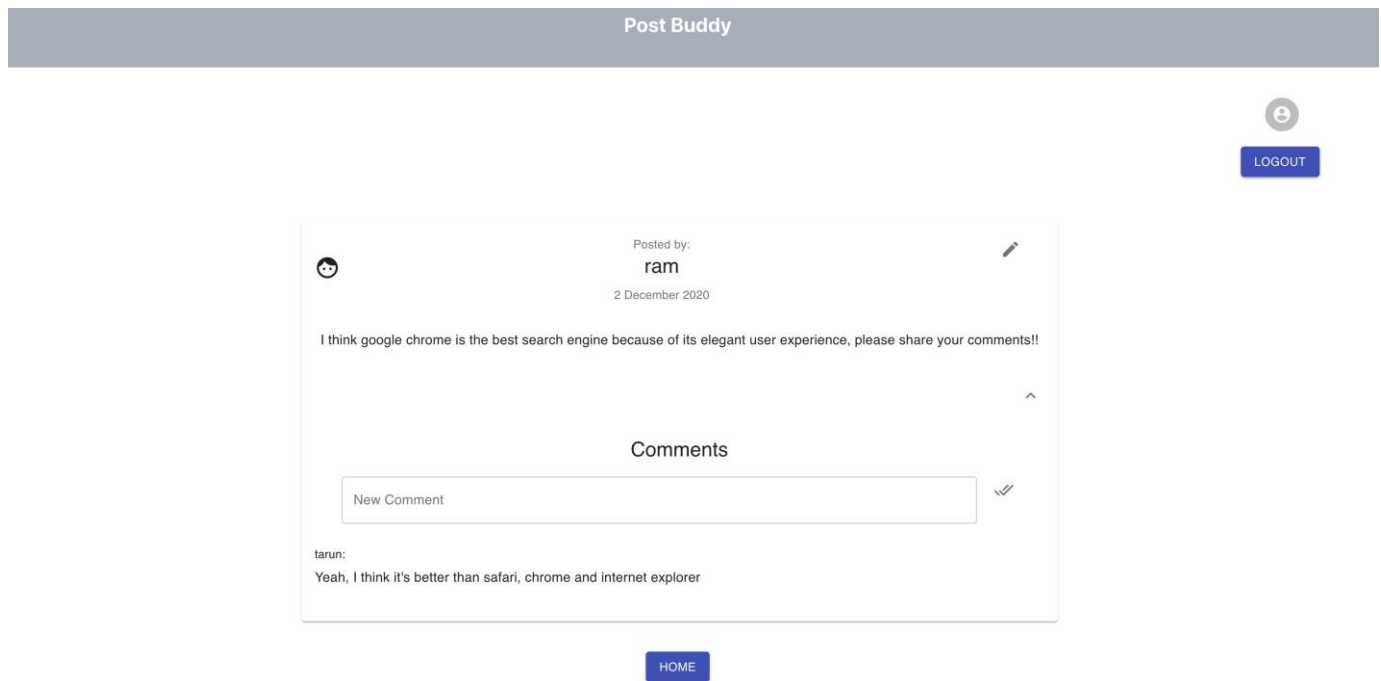
When the user logs into the system, the home page will be displayed with enhanced features such as the posts written by self can be edited and new posts can be written. In place of login and signup options, the logout option and avatar will be displayed. The comments icon won't be disabled for the posts with no comments unlike the scenario where user is not logged in. For such posts, add comment icon will be displayed at bottom right.

9. Post Edit view



If the user chooses to edit the post, a modal pops up with the content of the post, where user can make the desired changes and save it, upon which the corresponding post will be displayed with modified content

10. Detailed post view for logged in user




If the user chooses to leave a comment for the post or view the existing comments, the page with the explicit view of a post will be displayed. If the post was written by self, then the post can be edited from this view as well. A user can comment on his/her own post and view the comments made by other users.

11. Edit and delete comments facility for logged in user

Comments

naksh:

I agree with you mate! I bought the similar refrigerator three years ago..I'm quite satisfied with it



harsha:

Hi Ram! What's the price of the refrigerator?

[HOME](#)

The user can edit or delete the comments made by self and can only view the comments made by others as depicted.

API overview

Rest endpoints and brief description:

Port number:8081

Prefix: <http://localhost:8081/>

1. Users related API calls:

i) **users/ (POST)**

This call is made when the user tries to register into the system. All the relevant fields will be provided from UI in the request body. If the registration is successful, then the corresponding user document is created and the truth value is returned. If the registration is unsuccessful due to duplicate username, then false will be returned

ii) users/login/:username/:password (GET)

This call is made when the user tries to login to the system, the credentials will be retrieved from the request parameters and verified with the user credentials in database. In case of mismatch, false will be returned or else true will be returned

iii) users/:username (GET)

The details of the user with the username provided in the request parameter will be retrieved as a result of this call

2. Posts related API calls:

i) posts/ (POST)

This call is made when the user tries to write a new post. All the details of the post will be provided in request body from UI and the date on which post is created as well as unique identifier of the post will be generated from backend application.

ii) posts/ (GET)

This call retrieves the details of all the posts present in the system. It will be useful while rendering the home page

iii) posts/updateContent (PATCH)

This call updates the content of specific post. The updated content and post id will be provided in request body from UI. This call is made when the user try to edit the post

iv) posts/decreaseCommentCount/:postId (PATCH)

This call decreases the comment count of the post with the post Id specified in the request parameter. It will be useful while deleting the existing comment

v) posts/increaseCommentCount/:postId (PATCH)

This call increases the comment count of the post with the post Id specified in the request parameter. It will be useful while creating the comment

3. Comments related API calls:

i) **comments/ (POST)**

This call is made while creating the comment for a post. The details required will be provided in the request body from UI. The unique identifier for a comment will be generated by the backend application

ii) **comments/ (GET)**

This call is made to retrieve all the comments present in the system

iii) **comments/:commentId (PATCH)**

This call is useful to update the content of the comment with the comment Id specified in the request parameter. It is useful when the user edits his/her own comment

iv) **comments/:commentId (GET)**

This call is useful to obtain the comment details whose comment Id is specified as the request parameter

v) **comments/:commentId (DELETE)**

This call is useful to delete comment with the comment Id mentioned in the request parameter. It will be useful when user deletes his/her own comment

vi) **comments/post/:postId (GET)**

This call retrieves all the comments of a post with the post Id specified in the request parameter. It will be useful when the user is redirected to a post specific view

Containerization using docker

The development of 'PostBuddy' web application is bifurcated into two sub-applications, one for front-end and other for backend. For each of these applications, the docker file is attached so that the entire application can be bundled together as a container image.

1. For front-end application

Command to build the docker image:

```
docker build -t postbuddyfrontend/react-app .
```

Command to run the docker container:

```
docker run -d -it -p 3000:3000/tcp --name react-app postbuddyfrontend/react-app:latest
```

2. For back-end application

Command to build the docker image:

```
docker build -t postbuddybackend/node-web-app .
```

Command to run the docker container:

```
docker run -p 8081:8081 -d postbuddybackend/node-web-app
```

Procedure for system setup

Running the applications instead of docker containers:

- Install the node, npm and mongoDB in the local system with versions as specified in the technology stack
- Create a database named 'postbuddy' in the local MongoDB setup
- Import the collections script files (.json files) attached in the database folder into the 'postbuddy' database
- **Running the backend and frontend applications:**
- Navigate to the application folder and execute command '**npm install**', which downloads all the required node_modules for the application
- Install the additional dependencies and libraries specified in the technology stack for backend and frontend applications accordingly
- To run the application, execute command '**npm run**'
- The front-end application will be running at port 3000 and the back-end application will be running on port 8081, whereas database will be located at default port of mongoDB i.e., 27017

Folders attached

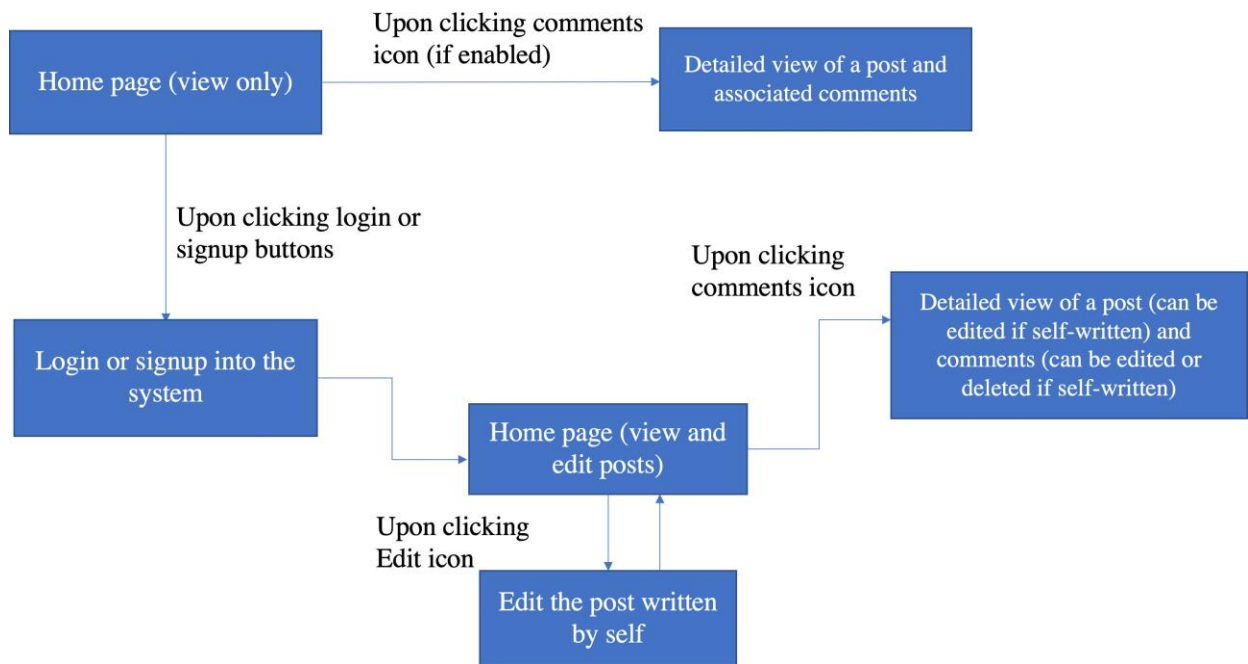
postbuddy: This folder contains the front-end code (ReactJs applictaion)

postbuddy-backend: This folder contains the back-end code (NodeJs)

Database: This folder contains all the collection scripts in Json format in the 'postbuddy' database which can be imported later. The additional information about database setup is also specified in "databaseInfo" folder

README file: which contains all the commands required to setup the environment and run the application in detail

Flow diagram



Post Buddy

Sriram Namilakonda

Email: sriramnamilakonda1@gmail.com

Github Profile: <https://github.com/sriram1100>

Phone: +91 9849737755

