A
PROJECT REPORT ON

Blog Management System

By

Ruchit Parekh (CE006) (20CEU0S035)

Gautam Rathod (CE009) (20CEUTG136)

Aadit Baldha (CE016) (20CEUON123)

B.Tech CE Semester-VI
Subject: System Design Practice

Guided by:
Prof. Brijesh S. Bhatt
Dept. of Comp. Engg.

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

CERTIFICATE

This is to certify that the practical / term work carried out in the subject of **System Design Practice** and recorded in this journal is the bonafide work of

Ruchit Parekh (CE006) (20CEU0S035)

Gautam Rathod (CE009) (20CEUTG136)

Aadit Baldha (CE016) (20CEUON123)

of B.Tech semester VI in the branch of Computer Engineering during the academic year 2022-2023.

Prof. Brijesh S. Bhatt
Professor,
Computer Engg, Department
Faculty of Technology
Dharmsinh Desai University

Dr. C. K. Bhensdadia
Head,
Computer Engg, Department
Faculty of Technology
Dharmsinh Desai University

Table of Contents

Introduction

Brief Overview

- A Blog Management System is a software platform designed to manage and organize the content, and functionality of a blog. It allows bloggers to publish their content, and acts as a means to connect with friends and family. Blogs and social media platforms offer a great way for individuals and businesses to network with others in their industry or niche. By connecting with others, individuals and businesses can share knowledge, collaborate on projects, and potentially open up new business opportunities.

- The core feature of Blog Management System allows users to create, publish and delete their blog. This feature allows bloggers to easily add new posts, and delete posts.

- Additionally likes and comments allows bloggers to add comments on blogs, and like certain posts. This feature helps bloggers to engage with their readers, and respond to their comments.

- Friendship management tools allow users/bloggers to connect with people and increase their involvement.

- Search tools help bloggers and readers to find blogs of their interested topics. This feature typically includes tools for keyword research and finds matching keywords in the blogs.

- Analytics tools provide admins with insights into the blog's performance, including engagement, based on the likes on the post. This feature also includes tracking the sentiment of a particular post based on the positive or negative comments made on that post.

Plan

- The first step would be to gather requirements. Based on requirements, we would design the system architecture, database schema, and user interface. The technology used is MERN stack (MongoDB, ExpressJs, ReactJs, NodeJs).

- Testing would be conducted to ensure code quality and functionality. Finally, the application would be monitored by developers , which will ensure fixing bugs, and enhancing features based on requirements of the project.

Approach

The approach to developing an online Blog Management should involve following best practices in software development, such as modular programming, naming conventions, and code commenting. It's important to use version control to ensure code quality and catch errors early in the development process. Overall, our approach would be to prioritize code quality, functionality while also ensuring that the project is maintainable.

Analysis

Problem Statement

The problem that a blog management project aims to solve is the lack of user interaction to connect with each other. In traditional blog management systems the blogs act as plain articles where the readers do not have any kind of interaction with the blogs and cannot connect with other people. Moreover, the traditional blog management system does not include any blog analysis where the user's reception plays a vital role in deciding the blog's sentiment.

Scope

The scope of an Online Blog Management project includes the development of a web-based platform that allows users to post a blog with/ without an image. Other users can interact with the blog by liking or by commenting on the post. Users can search for blogs by searching particular keywords. Users can add friends and see their posts. Users can also delete their post.

Admin can view the sentiment analysis on any particular blog based on the total positive, negative or neutral comments. The project should also include features such as user authentication, and session management.

Software Requirements Specification (SRS):-

Types of Users:-

1. Bloggers
2. Admin

Functional Requirements :-

R.1 : Authenticate User

R.2: Manage Blog

R.3: Impressions of blog based on like and comments

R.4: Manage Friends

R.5: View Blog sentiment Analysis

R.1: Authenticate User

R.1.1 : Register User

Description:- User will be able to login only after registration.For registration user have to enter User details like username , password , profile image , email id etc .

Input:- User Details
Output:- User Register Successfully

R.1.2 : Login

Description:- After Registering Successfully User will be able to login. Correct Username and password should be entered so,user will be successfully authenticated otherwise it will not be able to authenticate.

Input:- Username and Password
Output:- User will be able to login and navigate to home page.

R.1.3 : Logout

Description :- User can Logout from the system after clicking on logout Button.

Input:- Click on Logout button
Output:- Logout from the system.

R.2 : Manage Blog

R.2.1 : Add Blog

Description :- If User wants to add a blog ,then the user has to give the blog title Blog Description and image for that blog.After that blog will be posted Successfully.

Input:- Enter Blog Details
Output:- Blog created successfully

R.2.2 : Delete Blog

Description :- If User want to delete a blog ,then the user can just click on the delete button , after that blog will be deleted.

Input :- Click on delete Blog.
Output:- Blog deleted Successfully.

R.2.3 : View All Blogs

Description :- User all his/her blog posted as well as blog posted by other people.

R.2.4 : Search Blog

Description :- User can search any blog on the system. For that user have to type a blog name in search blog if it matches with any of the blog name in system it will render that blog otherwise not.

Input:- Enter Blog name to be searched
Output:- searched Blog will be displayed.

R.3 : Impressions of Blog based on likes and comments

R.3.1 : Like a Blog

Description :- Users will be able to like a blog.For that user will have to click on the like button of the blog.

Input:- Click on like Button
Output:- Blog like successfully

R.3.2 : Comment on blog

Description :- Users will be able to comment on any blog. The username and the comment of the user will be displayed in comment section of that post.

Input:- Enter Comment
Output:- comment will be added on that blog.

R.4 Manage Friends

R.4.1 : Add friend

Description:- User will be able to add friends , by clicking on add button.

Input:- Click on Add friend Button.
Output:- Friend added successfully.

R.4.2 : View Friend Profile

Description :- if user wants to see his/her friend profile then user can view by just clicking on its profile photo.

Input:- Click on User Profile Photo
Output:- User's Friend profile will be displayed.

R.4.3. Remove Friend

Description :- User can remove friend from his/her friend list ,on clicking remove friend button.

Input :- Click on Remove Button
Output :- Friend is remove from friend list.

R.5 : View Blog Sentiment Analysis

Description :- Admin will be able to add sentiment analysis of a particular Blog. He/She will analyze blog comments and based on that analysis will be generated.

Input:- Blog
Output :- sentiment analysis of that blog

Non Functional Requirements :- These are the requirements that are not functional in nature. Especially these are the constraints the system must work within.

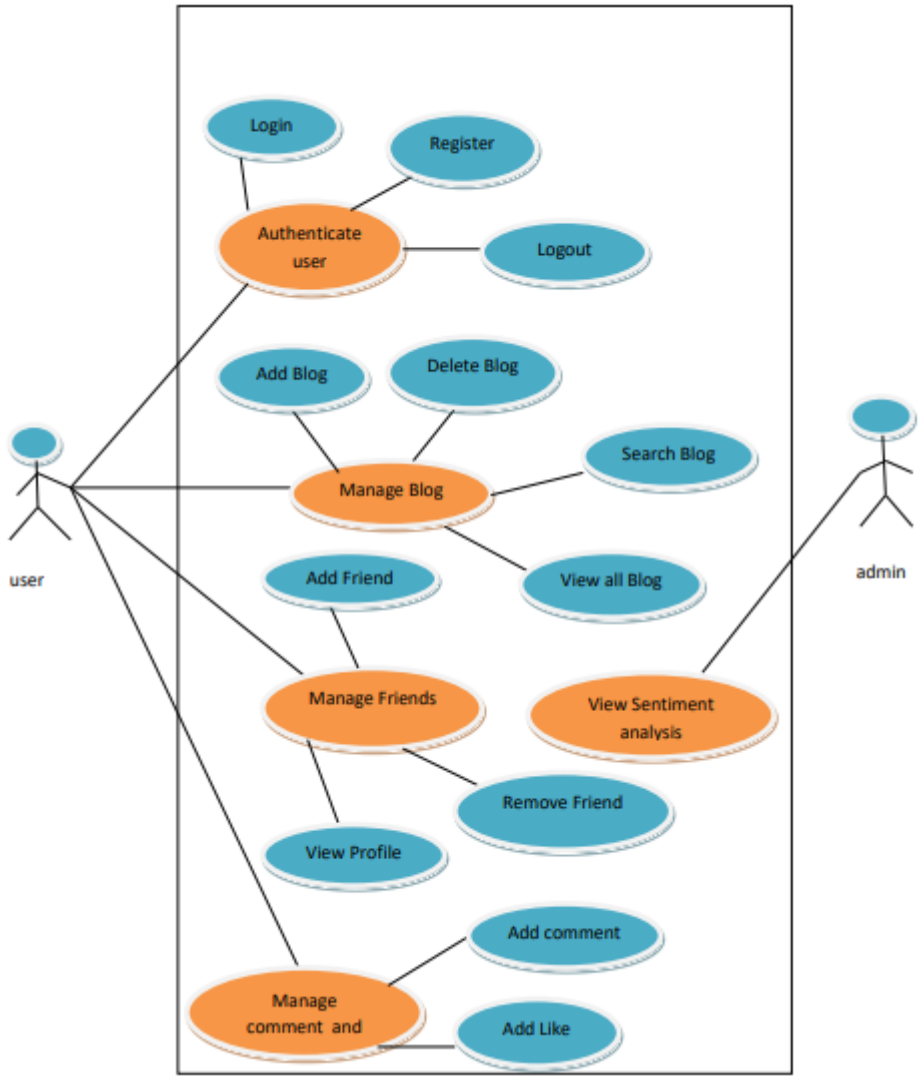
Performance Requirements:- The system response time must be less than 30 seconds for the user interface. Or else the system will show TIMED OUT.

Reliability Requirements:- The system shall have a minimum uptime of 99% excluding time pre-scheduled for maintenance and/or upgrades.

Safety Requirements:- All the system data must be backed up every day and the backup copies stored in another server at different locations for disaster recovery.

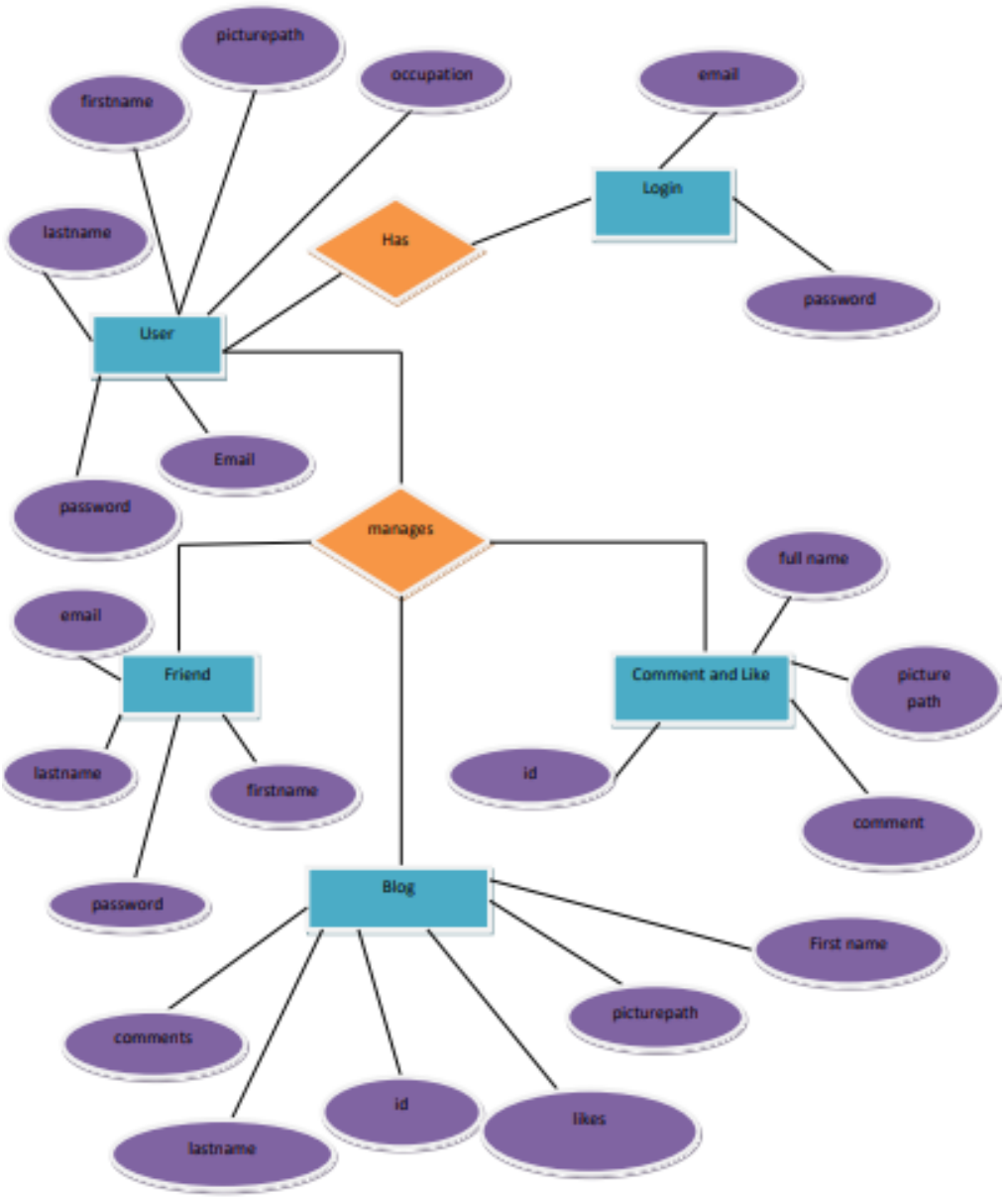
Quality Attributes:- the source code for the system is well documented for ease of maintenance and upgrading the system in future.

Use Case Diagram (UML) :-

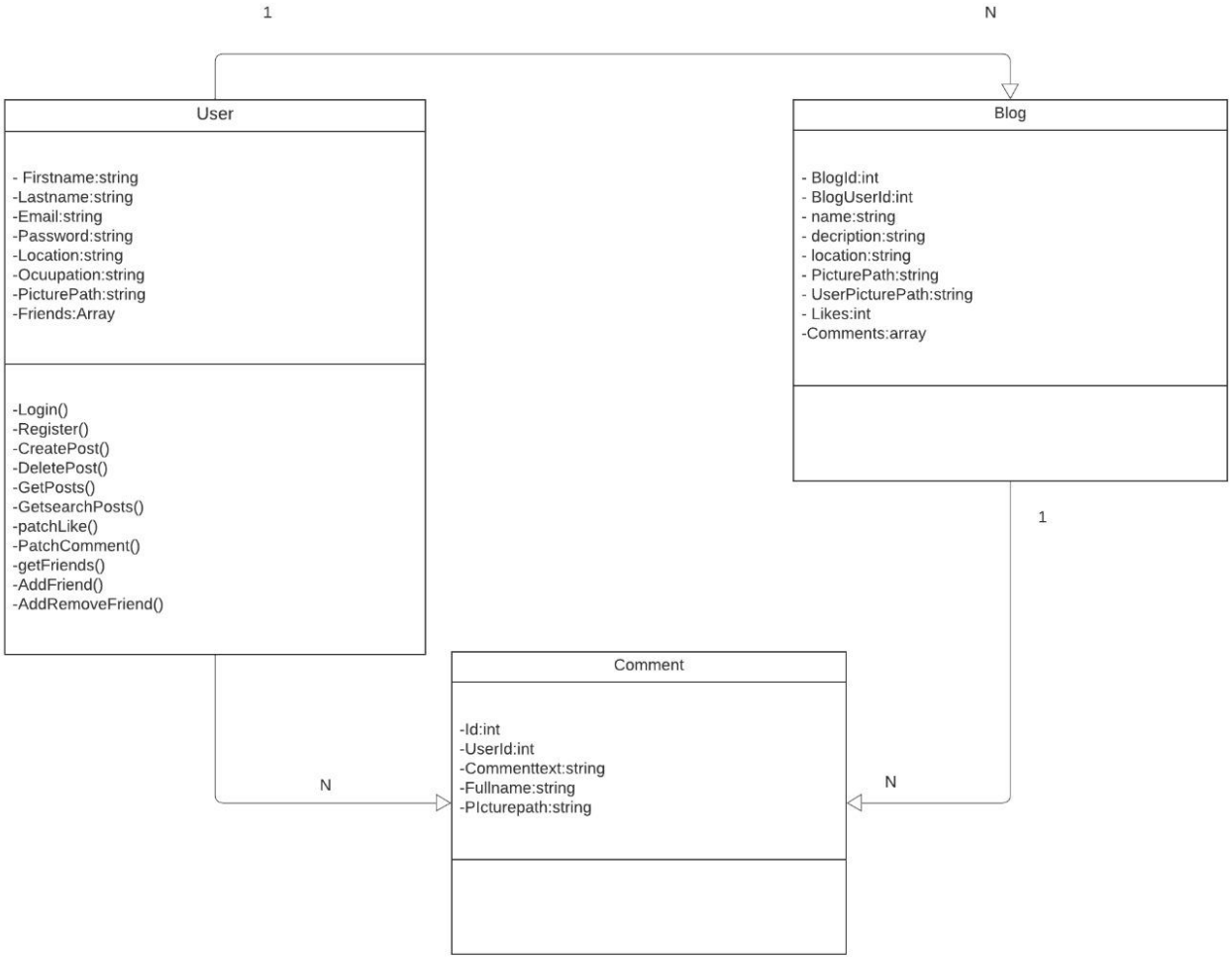


Design

ER Diagram



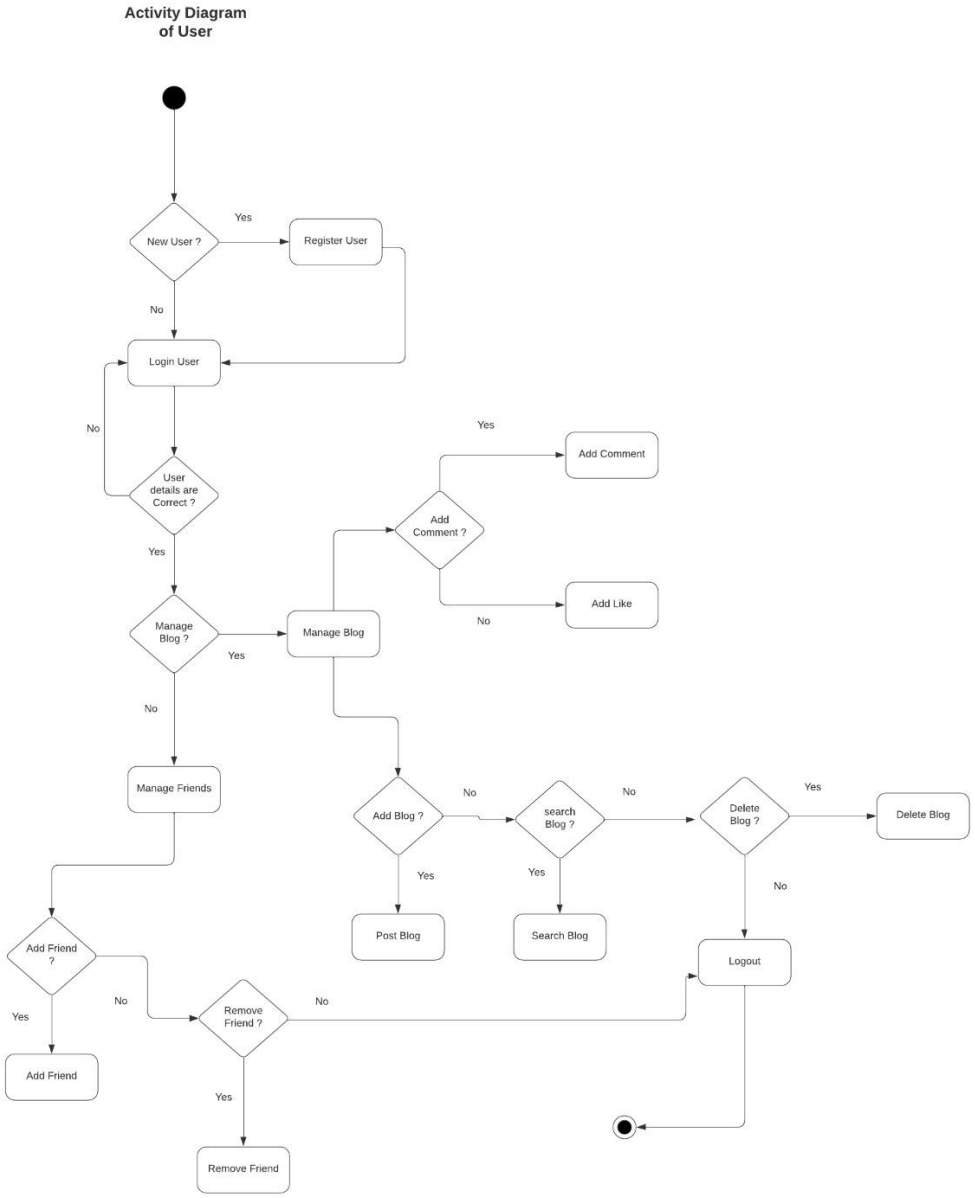
Class Diagram



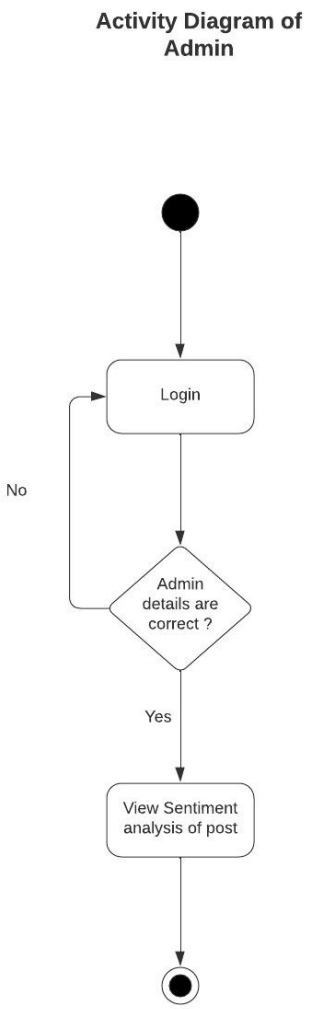
Sequence Diagram

Activity Diagram

Activity Diagram of User



Activity Diagram of Admin



Modules of System

The system consists of 2 main modules namely

1. User Module
2. Admin Module

Each module consists of several methods to implement the required functionality. Implementation is done using ReactJS, NodeJS, ExpressJS and MongoDB (for data storage).

User Module :-

Profile : - In this section , Users can view their profile.

Manage Blog :- In this section Users can add blogs , view all blogs of different users and can also delete a blog.

Search Blog :- In this section , users can search blogs based on blog title.

Comment and like :- In this section , users can add comments and like on a particular blog.

Manage friends :- In this section , users can add friends and remove friends.

Admin Module :-

Sentiment Analysis:- In this section admin can view the sentiment analysis of a particular post based on the positive,negative and neutral comments.

Database Design

User Model		
Field Name	**Datatype**	**Constraints**
FirstName	String	Required, minlength = 2, maxlength = 50
LastName	String	Required , minlength = 2, maxlength = 50
Email	String	Required. Unique , maxlength = 50
Password	String	Required , minlength = 5
PicturePath	String	Default = Null
Friends	Array	Default = Null
Location	String	Required
occupation	String	Required

Post Model		
Field Name	**Datatype**	**Constraints**
UserId	String	Required
FirstName	String	Required
LastName	String	Required
Description	String	-
picturePath	String	-
userPicturePath	String	Required
likes	Map (of Boolean)	-
comments	Map (of String)	-

Implementation Details

The project is divided into two parts: Client side and Server side.
Client side is implemented using ReactJs. It has all the required Components and Routing. Server side is implemented using NodeJs, ExpressJs and mongoose. MongoDB is used for data storage.

Technology Used :-

- **ReactJS**
- **NodeJS**
- **Express**
- **MongoDB**
- **Javascript**
- **Material UI**
- **HTML**

Tools Used:-

- **Git**
- **VS Code**
- **MongoDB Atlas**
- **Selenium Web Driver**

Function Prototype :

Login and Register :-

```
const login = async (values, onSubmitProps) => {

  if (values.email == "admin@gmail.com" && values.password == "admin1234") {
    const loggedInResponse = await fetch(`http://${process.env.REACT_APP_IP}:3001/auth/adminLogin`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(values),
    });
    console.log(loggedInResponse);
    const loggedIn = await loggedInResponse.json();
    // console.log(loggedIn);
    onSubmitProps.resetForm();
    if (loggedIn) {
      dispatch(
        setLogin({
          user: loggedIn.user,
          token: loggedIn.token,
        })
      );
      navigate("/admin");
    }
  }

  else {
    const loggedInResponse = await fetch(`http://${process.env.REACT_APP_IP}:3001/auth/login`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(values),
    });
    console.log(loggedInResponse);
    const loggedIn = await loggedInResponse.json();
    // console.log(loggedIn);
    onSubmitProps.resetForm();
    if (loggedIn) {
      dispatch(
        setLogin({
          user: loggedIn.user,
          token: loggedIn.token,
        })
      );
      navigate("/home");
    }
  }
};
```

```
/* LOGGING IN */
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await User.findOne({ email: email });
    if (!user) return res.status(400).json({ msg: "User does not exist. " });

    if(password==user.password)
      var isMatch=true;
    else
      var isMatch=false;

    // const isMatch = await compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: "Invalid credentials. " });

    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);
    delete user.password;
    // console.log(token)
    res.status(200).json({ token, user });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

```
/* REGISTER USER */
export const register = async (req, res) => {
  try {
    const {
      firstName,
      lastName,
      email,
      password,
      picturePath,
      friends,
      location,
      occupation,
    } = req.body;

    // const salt = await bcrypt.genSalt();
    // const passwordHash = await bcrypt.hash(password, salt);

    const newUser = new User({
      firstName,
      lastName,
      email,
      password,
      picturePath,
      friends,
      location,
      occupation,
      viewedProfile: Math.floor(Math.random() * 10000),
      impressions: Math.floor(Math.random() * 10000),
    });
    const savedUser = await newUser.save();
    res.status(201).json(savedUser);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Add Post :-

```
/* CREATE */
export const createPost = async (req, res) => {
  try {
    const { userId, description, picturePath } = req.body;
    const user = await User.findById(userId);
    const newPost = new Post({
      userId,
      firstName: user.firstName,
      lastName: user.lastName,
      location: user.location,
      description,
      userPicturePath: user.picturePath,
      picturePath,
      likes: {},
      comments: [],
    });
    await newPost.save();

    const post = await Post.find();
    res.status(201).json(post);
  } catch (err) {
    res.status(409).json({ message: err.message });
  }
};
```

```
const handlePost = async () => {
  const formData = new FormData();
  formData.append("userId", _id);
  formData.append("description", post);
  if (image) {
    formData.append("picture", image);
    formData.append("picturePath", image.name);
  }

  const response = await fetch(`http://${process.env.REACT_APP_IP}:3001/posts`, {
    method: "POST",
    headers: { Authorization: `Bearer ${token}` },
    body: formData,
  });
  const posts = await response.json();
  dispatch(setPosts({ posts }));
  setImage(null);
  setPost("");
};
```

Delete Post:-

```
/* DELETE */
export const deletePost = async (req, res) => {
  const { id } = req.params;
  const __filename = fileURLToPath(import.meta.url);
  const __dirname = dirname(__filename);
  console.log("Inside deletePost Outside try-catch");
  try {
    const posttodelete = await Post.findOne({ _id: id });
    const post = await Post.deleteOne({ _id: id });
    const deleteurl = posttodelete.picturePath;
    var directory = __dirname.replace("\controllers", "");
    console.log(directory);
    const directorypath = directory + "/public/assets/";

    if (deleteurl != undefined) {
      fs.unlink(directorypath + deleteurl, (err) => {
        if (err) {
          throw err;
        }
      });
    }
    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Read all Blogs :-

```
const PostsWidget = ({ userId, isProfile = false }) => {
  const dispatch = useDispatch();
  const posts = useSelector((state) => state.posts);
  const token = useSelector((state) => state.token);

  const getPosts = async () => {
    const response = await fetch(`http://${process.env.REACT_APP_IP}:3001/posts`, {
      method: "GET",
      headers: { Authorization: `Bearer ${token}` },
    });
    const data = await response.json();
    dispatch(setPosts({ posts: data }));
    console.log(posts)
  };

  const getUserPosts = async () => {
    const response = await fetch(
      `http://${process.env.REACT_APP_IP}:3001/posts/${userId}/posts`,
      {
        method: "GET",
        headers: { Authorization: `Bearer ${token}` },
      }
    );
    const data = await response.json();
    dispatch(setPosts({ posts: data }));
    console.log(posts)
  };
```

```
/* READ */
export const getFeedPosts = async (req, res) => {
  try {
    const post = await Post.find();
    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Search Blog :-

```
const searchbox = async () => {
  if (search != "") {
    const response = await fetch(`http://${process.env.REACT_APP_IP}:3001/posts/${search}`, {
      method: "GET",
      headers: { Authorization: `Bearer ${token}` },
    });
    const data = await response.json();
    dispatch(setPosts({ posts: data }));
    console.log(posts)

  }
};
```

```
export const getSearchedPosts = async (req, res) => {
  try {
    const { searchquery } = req.params;
    // if (searchquery == "") {
    //   const post = await Post.find();
    // }

    const post = await Post.find({ description: { $regex: `${searchquery}`,$options: 'i' } });

    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
}
```

Add Comment and Like :-

```
/* UPDATE */
export const likePost = async (req, res) => {
  try {
    const { id } = req.params;
    const { userId } = req.body;
    const post = await Post.findById(id);
    const isLiked = post.likes.get(userId);

    if (isLiked) {
      post.likes.delete(userId);
    } else {
      post.likes.set(userId, true);
    }

    const updatedPost = await Post.findByIdAndUpdate(
      id,
      { likes: post.likes },
      { new: true }
    );

    res.status(200).json(updatedPost);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};

export const commentPost = async (req, res) => {
  try {
    const { id } = req.params;
    const { userId } = req.body;
    const { commenttext } = req.body;
    const { fullname } = req.body;
    const { picturePath } = req.body;
    //const post = await Post.findById(id);

    const comment = { commentBy: userId, comment: commenttext, fullname: fullname, picturePath: picturePath };


    console.log(id, userId, commenttext);
    const updatedPost = await Post.findOneAndUpdate(
      { _id: id },
      { $push: { comments: comment } },
      { new: true }
    );


    res.status(200).json(updatedPost);
  } catch (err) {
    console.log(err.message);
    res.status(404).json({ message: err.message });
  }
};
```

```
const patchLike = async () => {
  // console.log("HEYD");
  const response = await fetch(`http://${process.env.REACT_APP_IP}:3001/posts/${postId}/like`, {
    method: "PATCH",
    headers: {
      Authorization: `Bearer ${token}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ userId: loggedInUserId }),
  });
  const updatedPost = await response.json();
  dispatch(setPost({ post: updatedPost }));
};

const displayComment = async () => {

  ToggleComments(!showComments)



  // dispatch(setFriends({ friends: data }));

}
const patchComment = async () => {

  const requestbody = {
    userId: loggedInUserId,
    commenttext: comment,
    fullname: fullName,
    picturePath: commentpicturePath,
  };
  console.log(JSON.stringify(requestbody))

  // ToggleComments(!showComments);
  const response = await fetch(`http://${process.env.REACT_APP_IP}:3001/posts/${postId}/comment`, {
    method: "PATCH",
    headers: {
      Authorization: `Bearer ${token}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify(requestbody),
  });
  const updatedPost = await response.json();
  dispatch(setPost({ post: updatedPost }));
  setComment("");
};
```

Manage Friends :-

```jsx
const FriendListWidget = ({ userId }) => {
  const dispatch = useDispatch();
  const { palette } = useTheme();
  const token = useSelector((state) => state.token);
  const friends = useSelector((state) => state.user.friends);

  const getFriends = async () => {
    const response = await fetch(
      `http://${process.env.REACT_APP_IP}:3001/users/${userId}/friends`,
      {
        method: "GET",
        headers: { Authorization: `Bearer ${token}` },
      }
    );
    const data = await response.json();
    dispatch(setFriends({ friends: data }));
  };

  useEffect(() => {
    getFriends();
  }, []); // eslint-disable-line react-hooks/exhaustive-deps

  return (
    <WidgetWrapper>
      <Typography
        color={palette.neutral.dark}
        variant="h5"
        fontWeight="500"
        sx={{ mb: "1.5rem" }}
      >
        Friend List
      </Typography>
      <Box display="flex" flexDirection="column" gap="1.5rem">
        {friends.map((friend) => (
          <Friend
            key={friend._id}
            friendId={friend._id}
            name={`${friend.firstName} ${friend.lastName}`}
            subtitle={friend.occupation}
            userPicturePath={friend.picturePath}
          />
        ))}
      </Box>
    </WidgetWrapper>
  );
};

export default FriendListWidget;
```

```
/* READ */
export const getUser = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findById(id);
    res.status(200).json(user);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};

export const getUserFriends = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findById(id);

    const friends = await Promise.all(
      user.friends.map((id) => User.findById(id))
    );
    const formattedFriends = friends.map(
      ({ _id, firstName, lastName, occupation, location, picturePath }) => {
        return { _id, firstName, lastName, occupation, location, picturePath };
      }
    );
    res.status(200).json(formattedFriends);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};

/* UPDATE */
export const addRemoveFriend = async (req, res) => {
  try {
    const { id, friendId } = req.params;
    const user = await User.findById(id);
    const friend = await User.findById(friendId);

    if (user.friends.includes(friendId)) {
      user.friends = user.friends.filter((id) => id !== friendId);
      friend.friends = friend.friends.filter((id) => id !== id);
    } else {
      user.friends.push(friendId);
      friend.friends.push(id);
    }
    await user.save();
    await friend.save();

    const friends = await Promise.all(
      user.friends.map((id) => User.findById(id))
    );
    const formattedFriends = friends.map(
      ({ _id, firstName, lastName, occupation, location, picturePath }) => {
        return { _id, firstName, lastName, occupation, location, picturePath };
      }
    );

    res.status(200).json(formattedFriends);
```

Sentiment Analysis :-

```
import Post from "../models/Post.js";
import {spawn} from 'child_process';

/** For Getting Post Analysis */
export const getPostStats=async (req,res)=>{
    try {
        const {postId}=req.params;
        var id=postId;
        console.log("Data sent to python file:"+JSON.stringify(id));
    } catch (err) {
        res.status(404).json({ message: err.message });
        console.log("error finding the post");
    }
    const childPython=spawn('python',['./controllers/sentimentAnalysis.py',JSON.stringify(id)]);
    childPython.stdout.on('data',(data)=>{
        console.log("Output of python file:"+ data);
        res.status(200).json({msg:`${data}`});
    });
    childPython.stderr.on('data',(err)=>{
        console.log(`Error of python file: ${err}`);
    });
};
```

```python
import sys
import pymongo
from bson.objectid import ObjectId
import ast
import pandas as pd
import numpy as np
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import matplotlib.pyplot as plt

postIdString=sys.argv[1]
postId=ast.literal_eval(postIdString)

#for connection to mongodb
connString="mongodb+srv://aadit:1234@blog-management-db.9g0czhk.mongodb.net/test"
client=pymongo.MongoClient(connString)
db=client['test_aadit']

post=db.posts.find_one({"_id":ObjectId(postId)})

# print(type(sys.argv[1]))
# dict_obj=ast.literal_eval(post[1])
commentsList=[]
commentsList=np.array(commentsList)
ind=0
for values in post['comments']:
    commentsList=np.append(commentsList,values['comment'])

#############################################################################################
sid=SentimentIntensityAnalyzer() #responsible for giving sentiment scores to each post
sentiment_scores=[] #for distiguishing the post's sentiment

for i in commentsList: #needs changes
    score=sid.polarity_scores(i)
    sentiment_scores.append(score)

# dict_to_send={}
# dict_to_send["sentiment_scores"]=sentiment_scores #needs changes


label=[] #for storing label of each posts
postiveCount=0
negativeCount=0
neutralCount=0

for s in sentiment_scores:
    if s['pos']>=0.05:
        label.append('Postive')
        postiveCount+=1
    elif s['neg']<=-0.05:
        label.append('Negative')
        negativeCount+=1
    else:
        label.append('Neutral')
        neutralCount+=1

# print(label)

if neutralCount>=negativeCount and neutralCount>=postiveCount:
    print("neutral")
elif negativeCount>=postiveCount and negativeCount>=neutralCount:
    print("negative")
else:
    print("positive")
```

Testing

For Testing our application , integration testing and manual testing is performed.

Integration Testing :- Each small part of the application is tested first and after that combine them and whole application testing is performed.

Manual testing :-

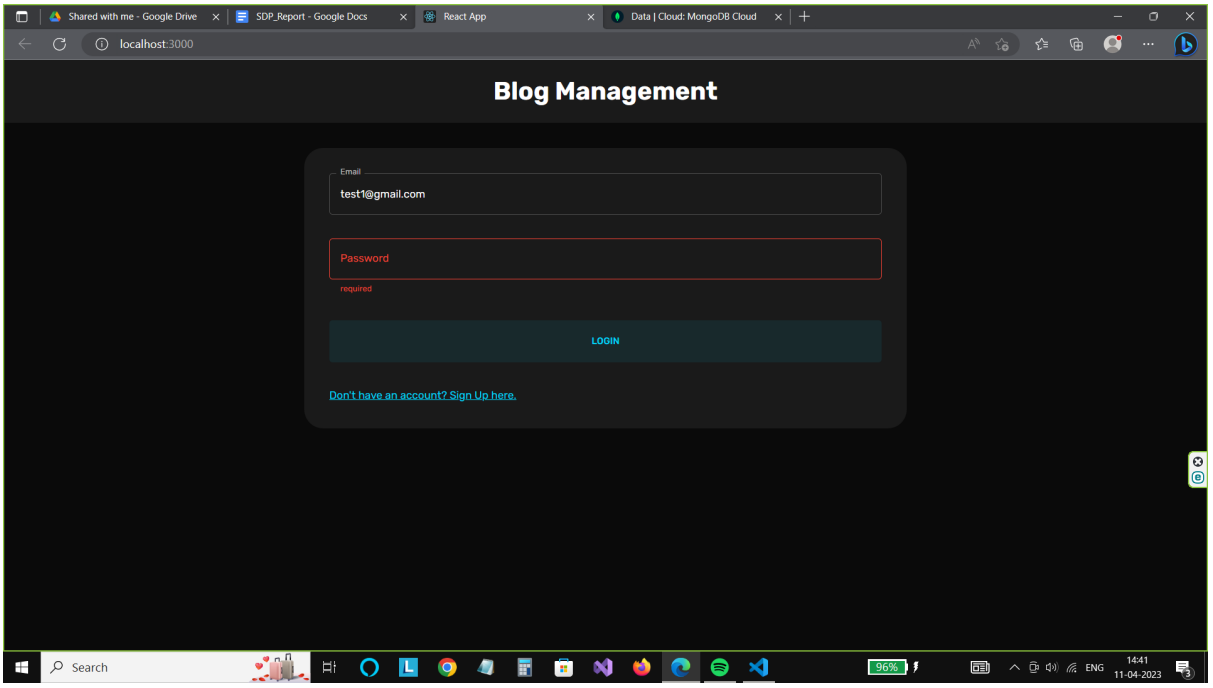
Sr No.	Test Scenario	Expected Result	Actual Result	Status
1	Login with empty fields	User should not be able to login	User not able to login.	Success
2	Login with incorrect credentials	User should not be able to login	User not able to login.	Success
3	Login with correct credentials	User able to login	User gets logged in.	Success
4	Register with some empty fields	User should not get registered	User doesn't get registered	Success
5	Register with valid details	User should able to register	User registers successfully	Success
6	User clicks on change theme button	The theme of webpage should change	The theme gets changed	Success
7	Add a post without Description and image	Post should not be added	Post doesn't get added	Success
8	Add a post with valid details	Post should be added.	Post gets added	Success

9	Like a blog	Like should be added on blog	Like gets added on the blog	Success
10	Comment on a blog with no details	Comment should not be added	Comment doesn't get added	Success
11	Comment on a blog with details	Comment should get added	Comment gets added	Success
12	Search a blog with no details	All Blogs should be displayed	All blogs get displayed	Success
13	Search a blog with details	Searched blog should be displayed	Searched blog gets displayed	Success
14	Remove a blog	Respective blog should get removed	The blog gets removed	Success
15	View a User	Respective User's Profile should get displayed	The User's profile gets displayed	Success
16	Add a friend	Respective User should get added in the friends list	The User gets added in the friends list	Success
17	Remove a friend	Respective User should get removed from the friends list	The User gets removed from the friends list	Success
18	View Sentiment Analysis of a Post	Respective analysis of post should be done	Analysis of the posts gets displayed	Success

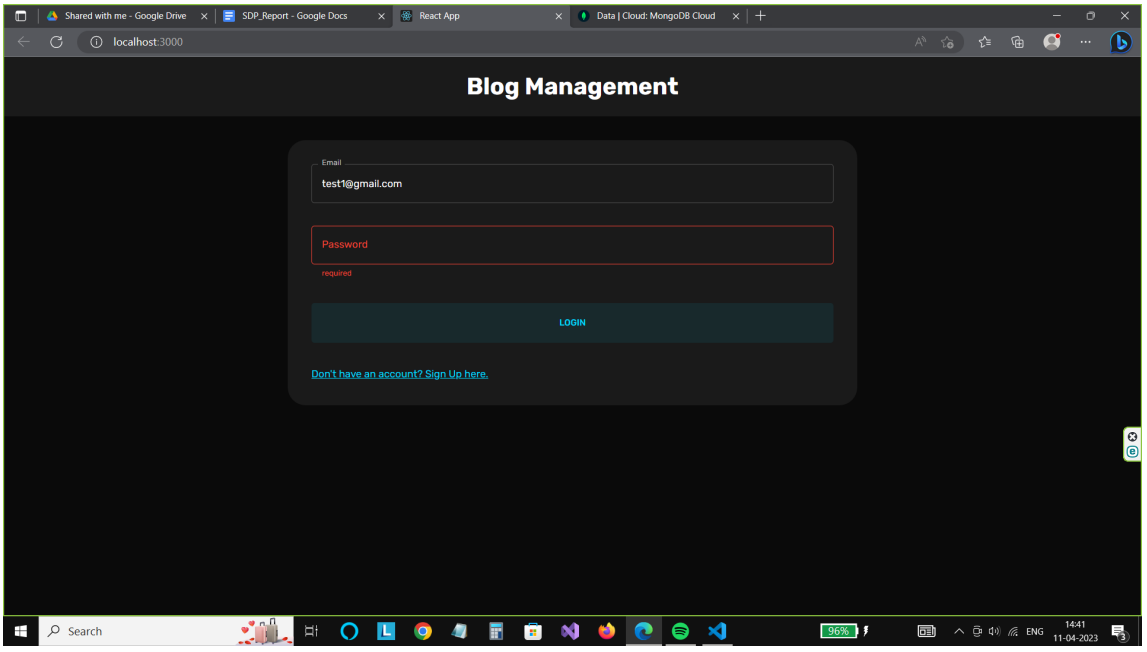
19	Logout	User should logout successfully	User gets logged out	Success

Screen-Shots

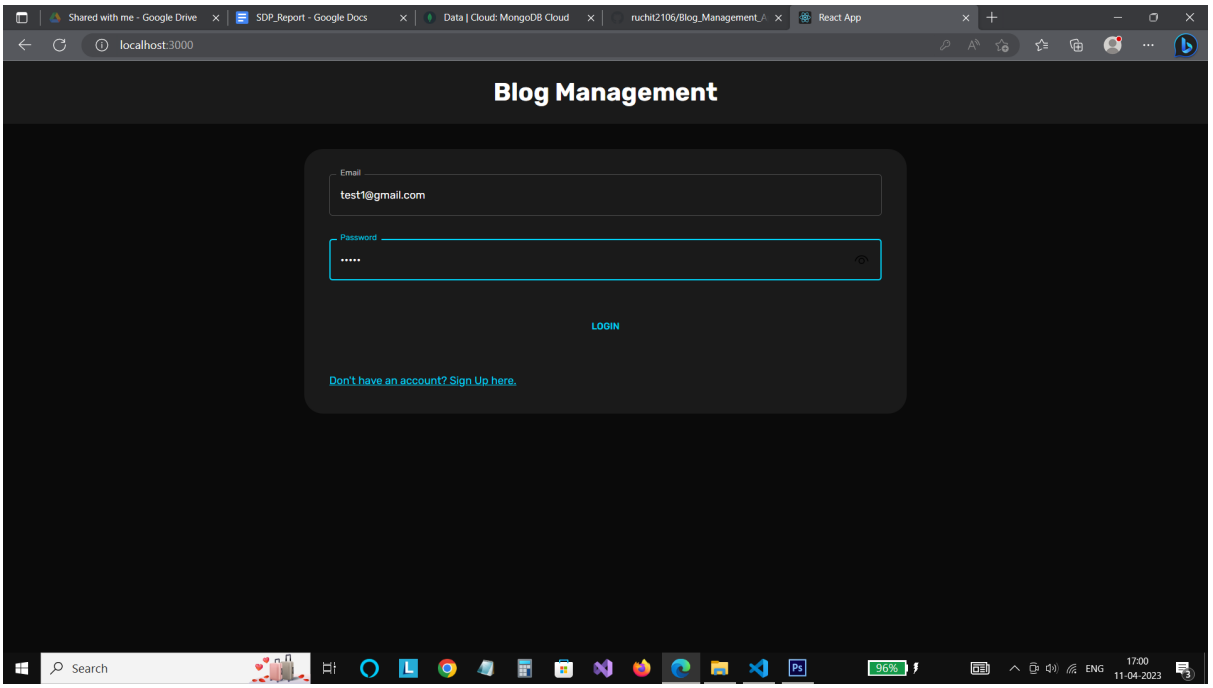
Login with empty values:-



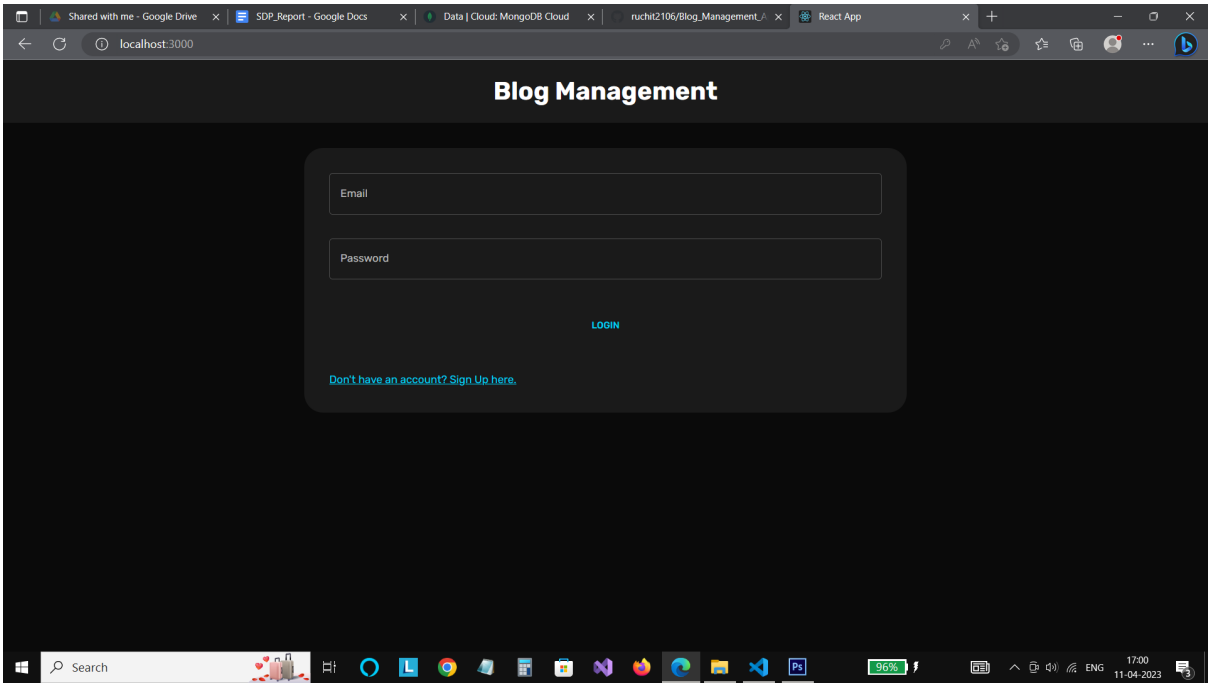
Output:-



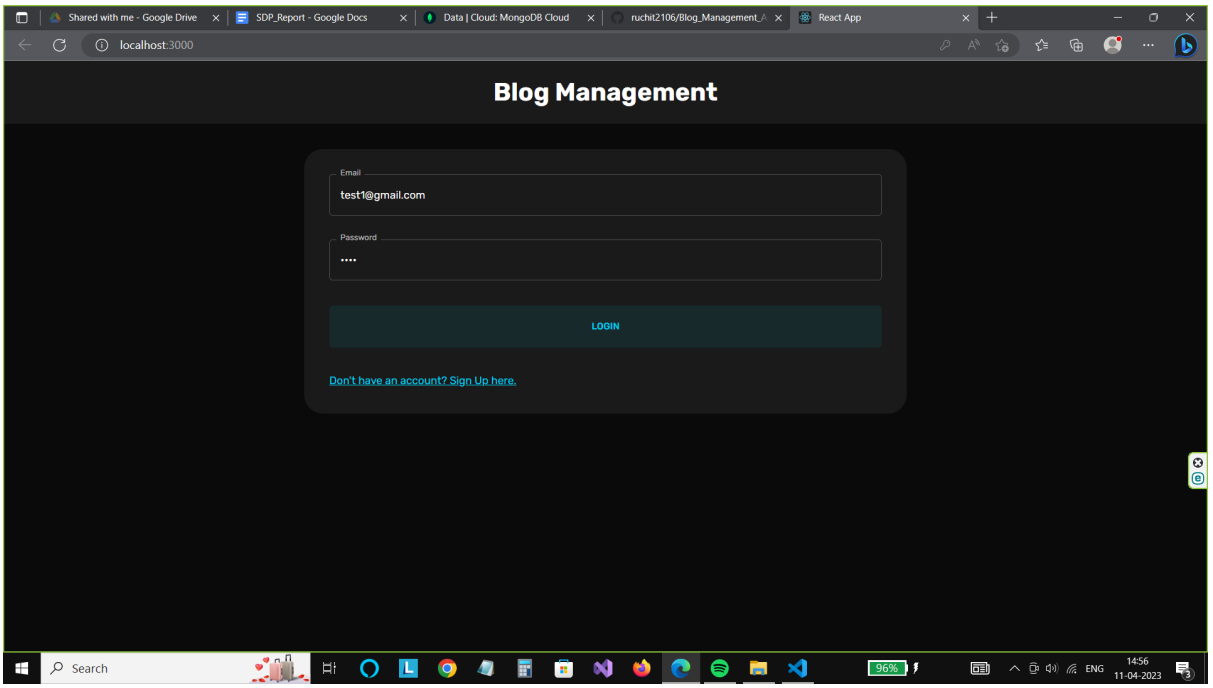
Login with incorrect credentials:-



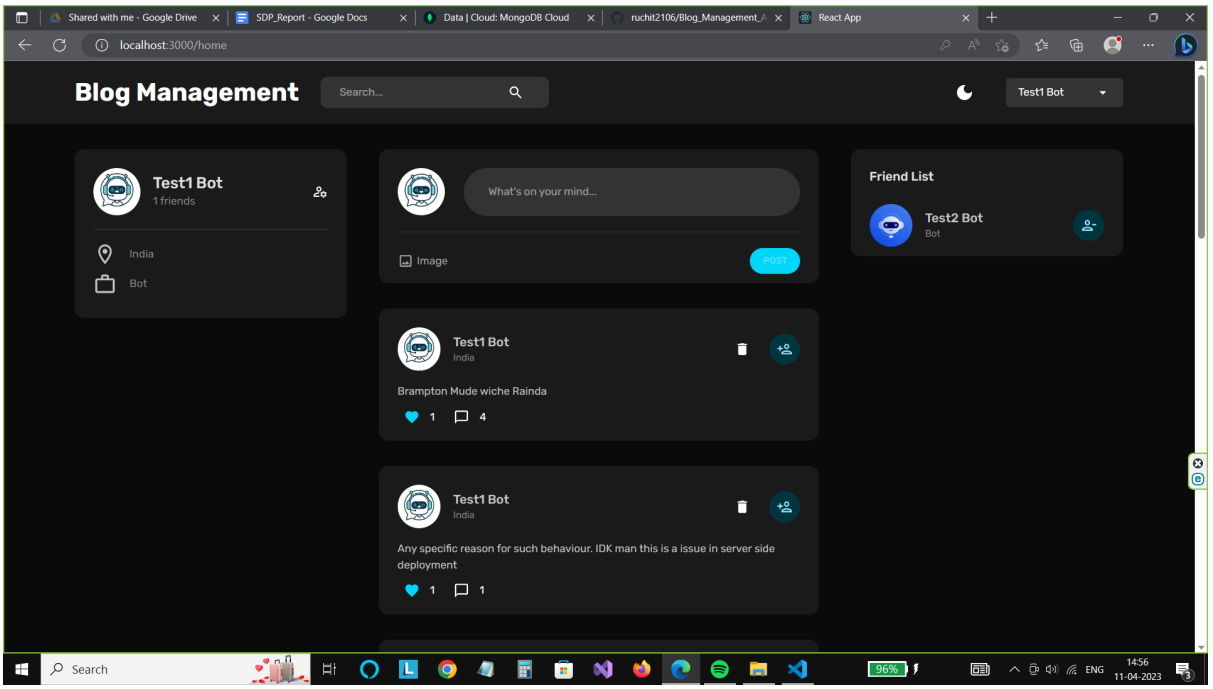
Output:-



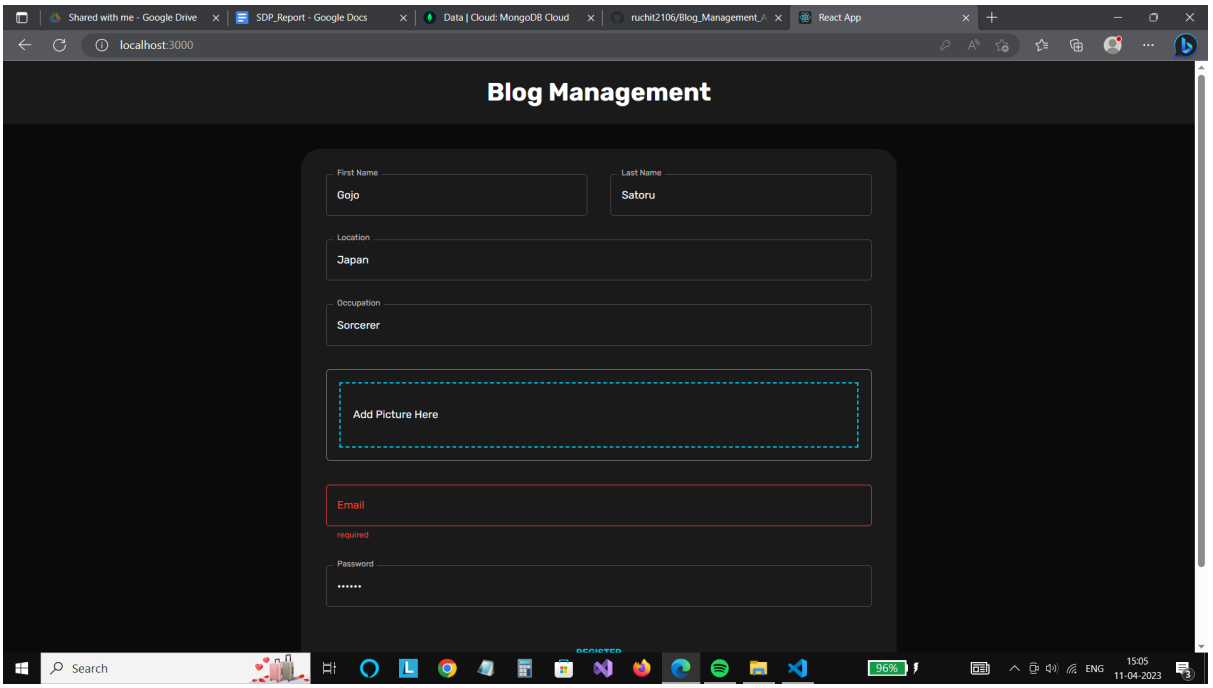
Login with correct credentials:-



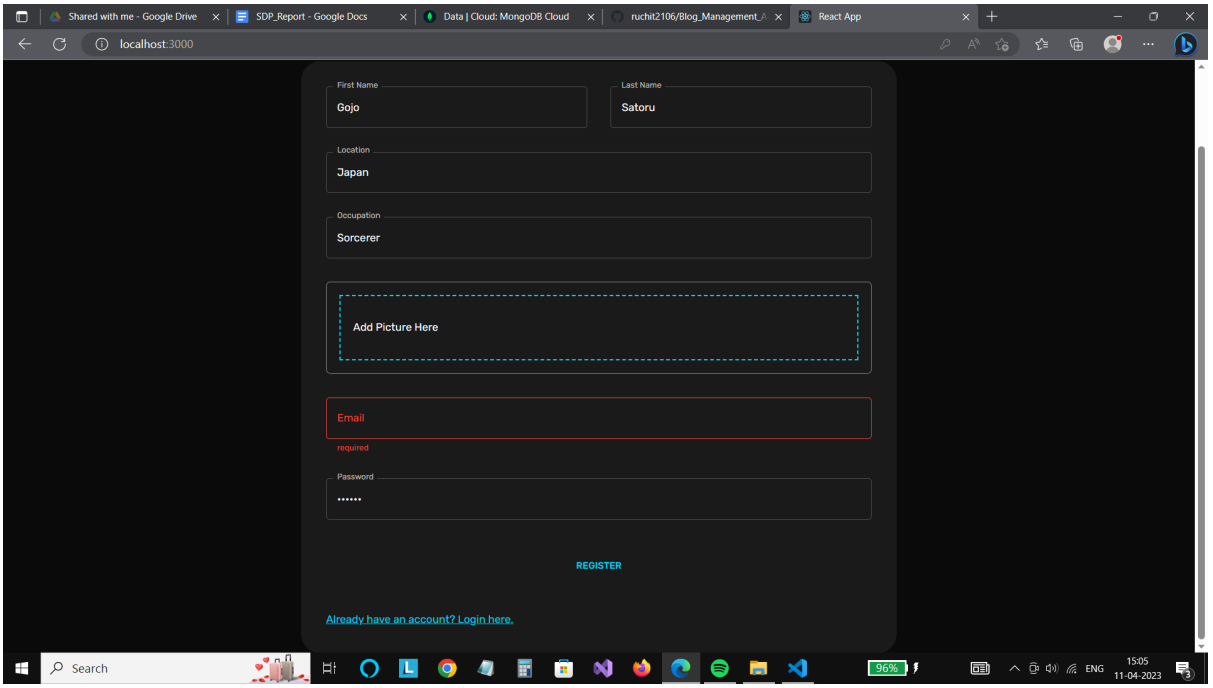
Output:-



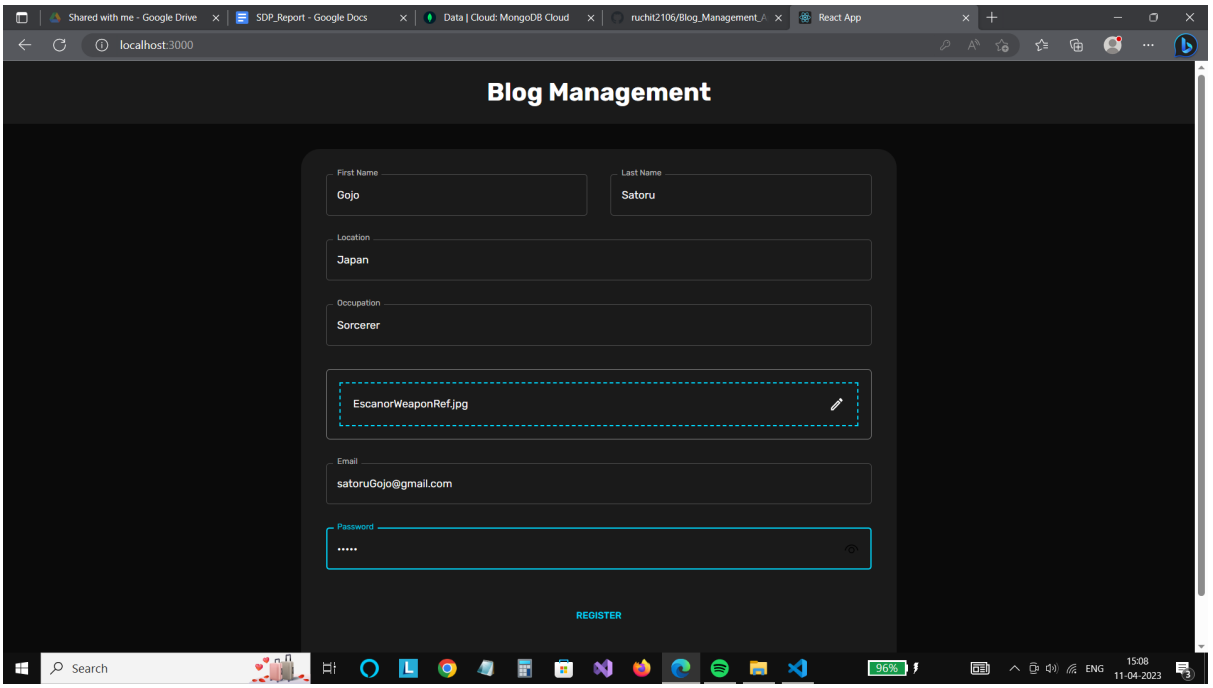
Register with some empty fields:-



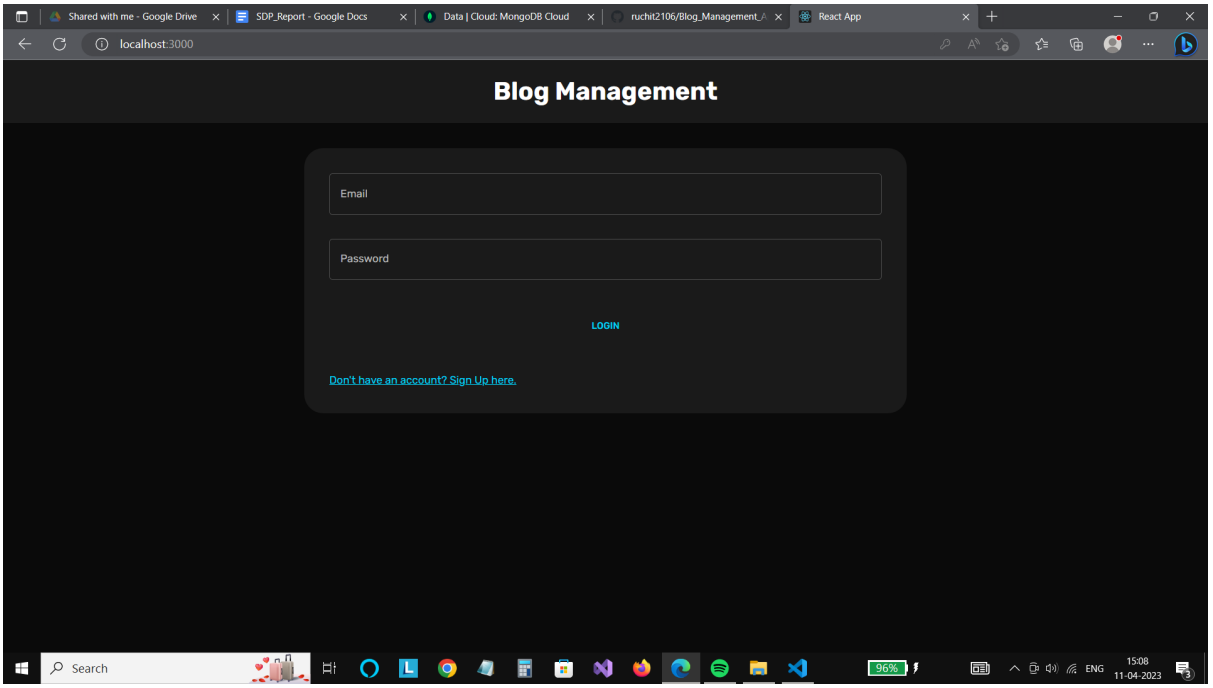
Output:-



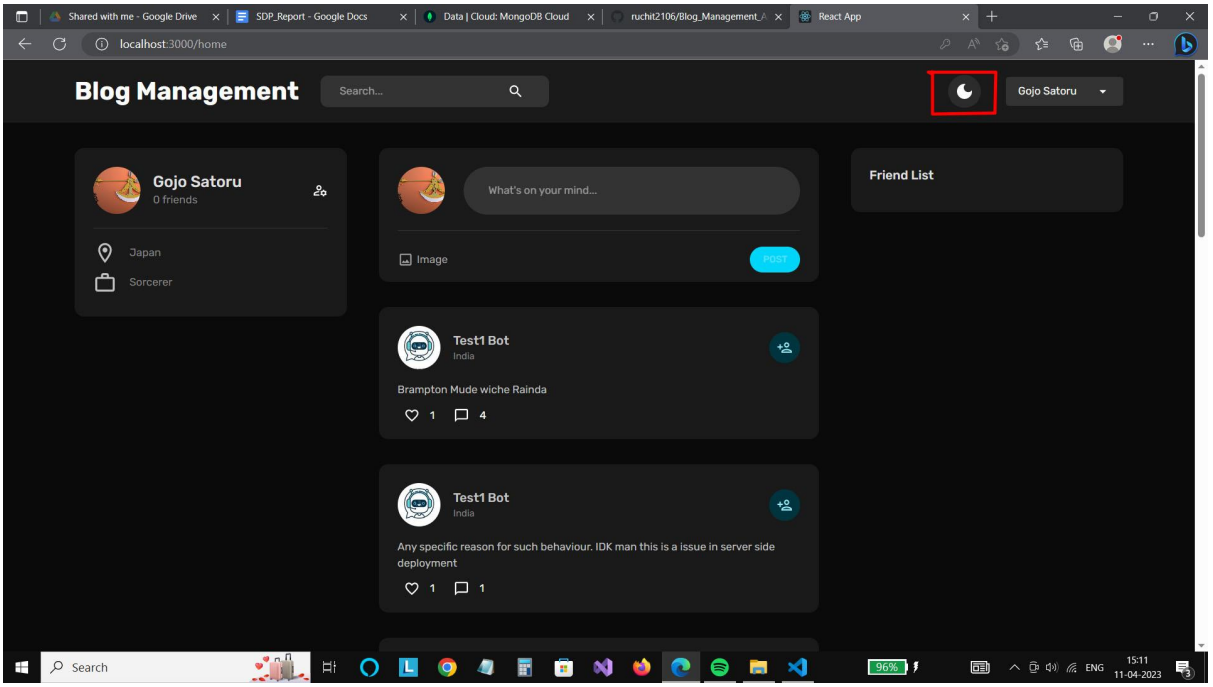
Register with valid details:-



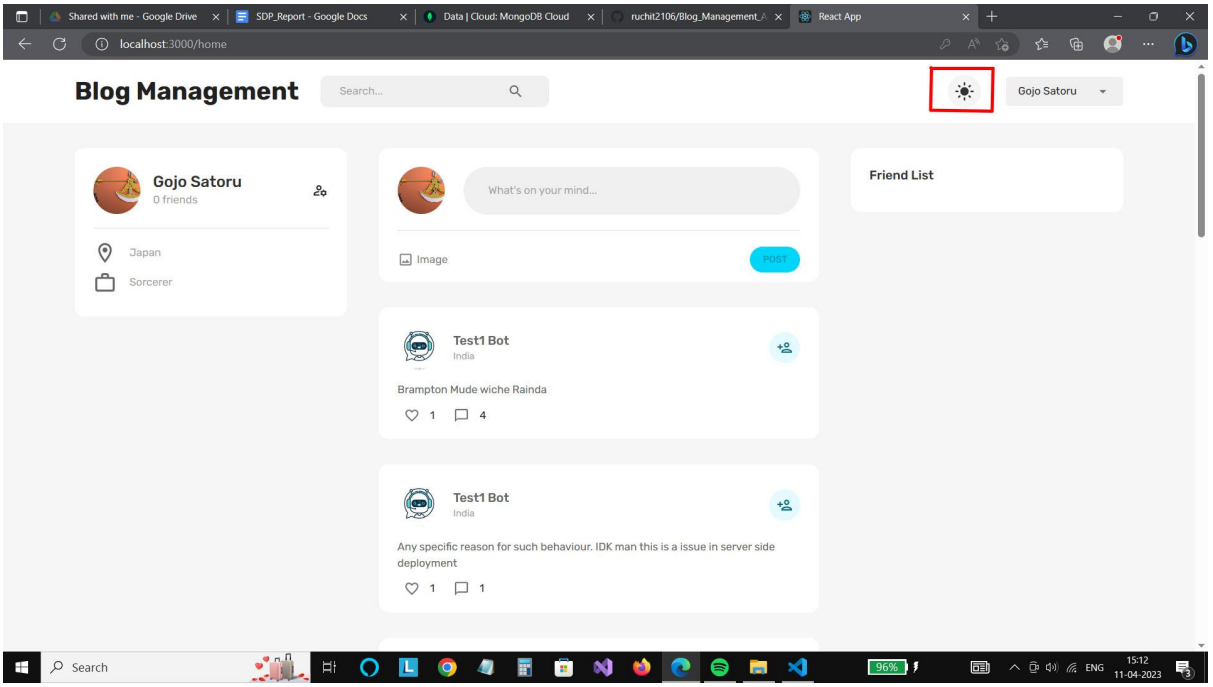
Output:-



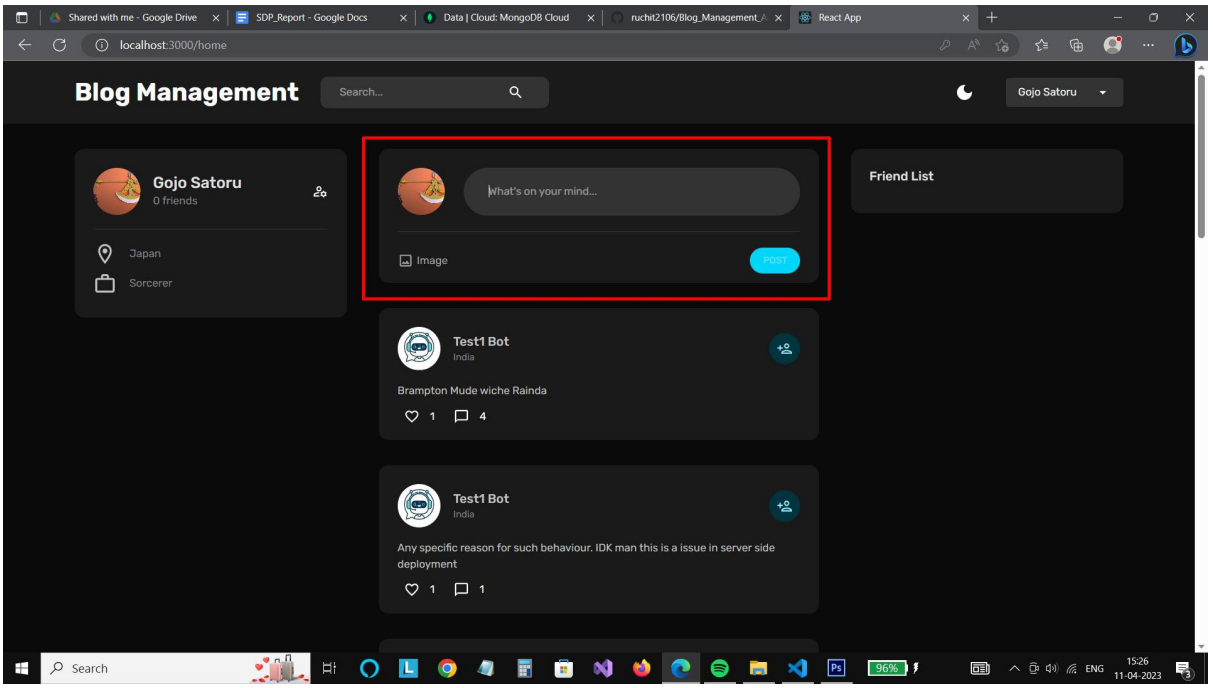
User clicks on change theme button:-


Shared with me - Google Drive
SDP_Report - Google Docs
Data | Cloud: MongoDB Cloud
ruchit2106/Blog_Management
React App
localhost:3000/home
Blog Management
Search...
Gojo Satoru
Gojo Satoru
0 friends
Japan
Sorcerer
What's on your mind...
Image
POST
Friend List
Test1 Bot
India
Brampton Mude wiche Rainda
1
4
Test1 Bot
India
Any specific reason for such behaviour. IDK man this is a issue in server side deployment
1
1
Search
96%
ENG
15:11
11-04-2023

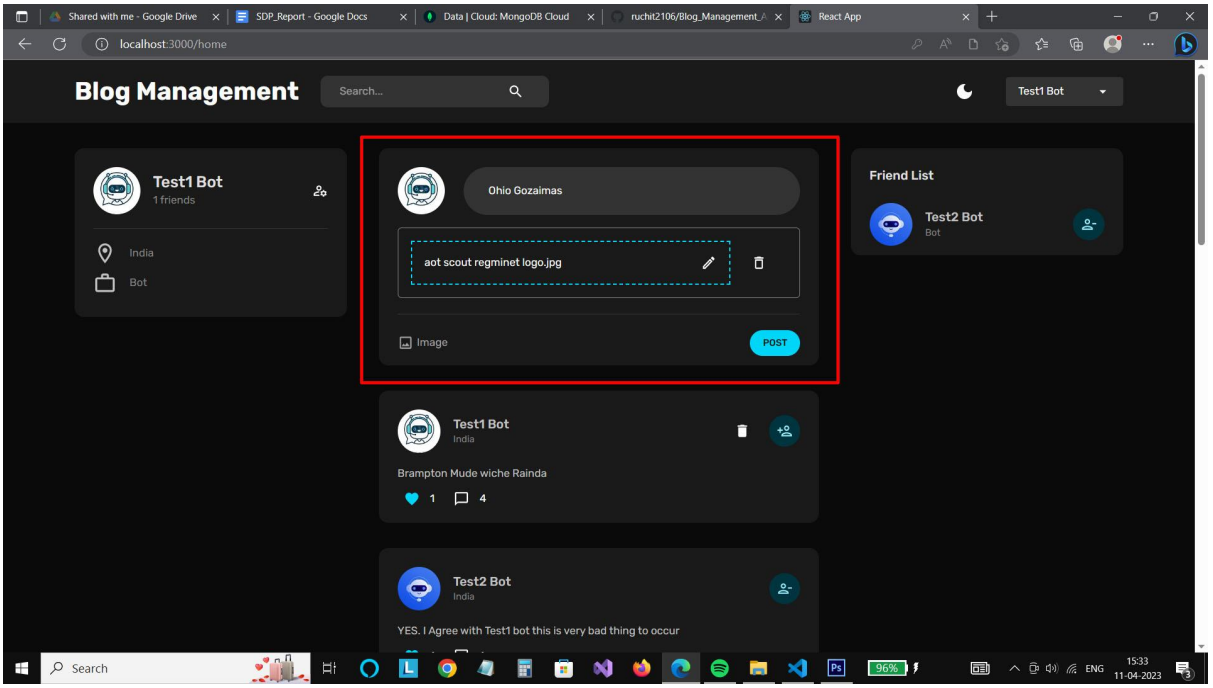
Output:-


Shared with me - Google Drive
SDP_Report - Google Docs
Data | Cloud: MongoDB Cloud
ruchit2106/Blog_Management
React App
localhost:3000/home
Blog Management
Search...
Gojo Satoru
Gojo Satoru
0 friends
Japan
Sorcerer
What's on your mind...
Image
POST
Friend List
Test1 Bot
India
Brampton Mude wiche Rainda
1
4
Test1 Bot
India
Any specific reason for such behaviour. IDK man this is a issue in server side deployment
1
1
Search
96%
ENG
15:12
11-04-2023

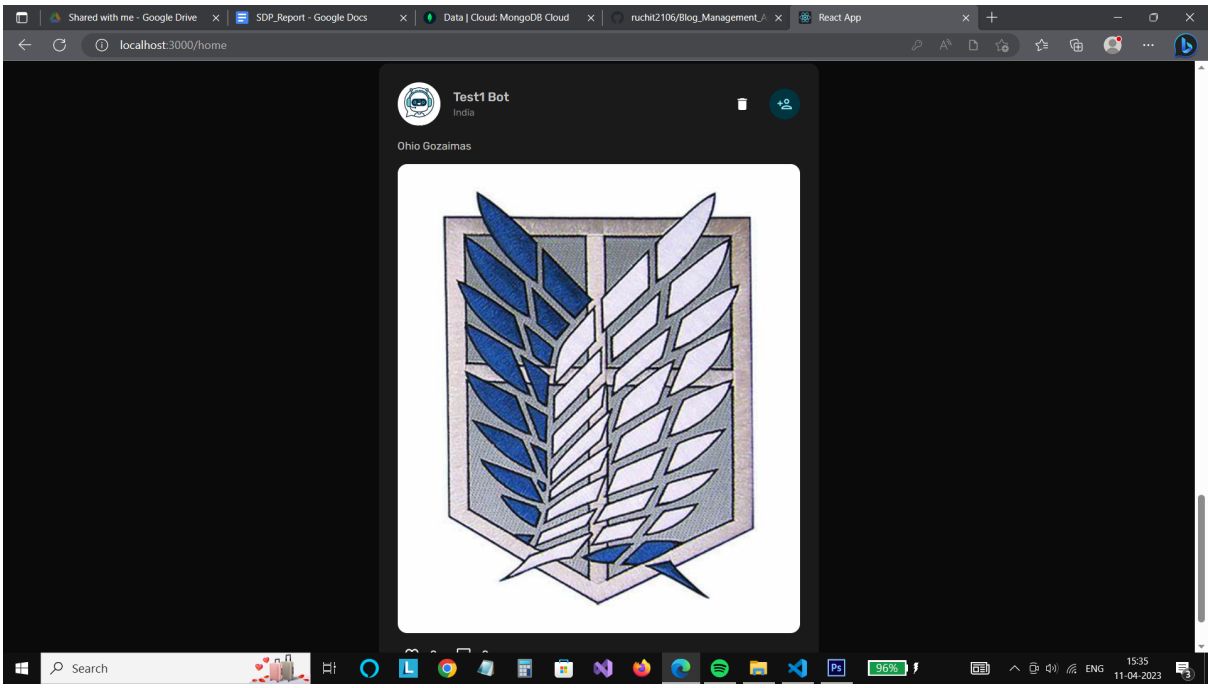
Add a Blog without Description and Image:- (Post button not clickable)



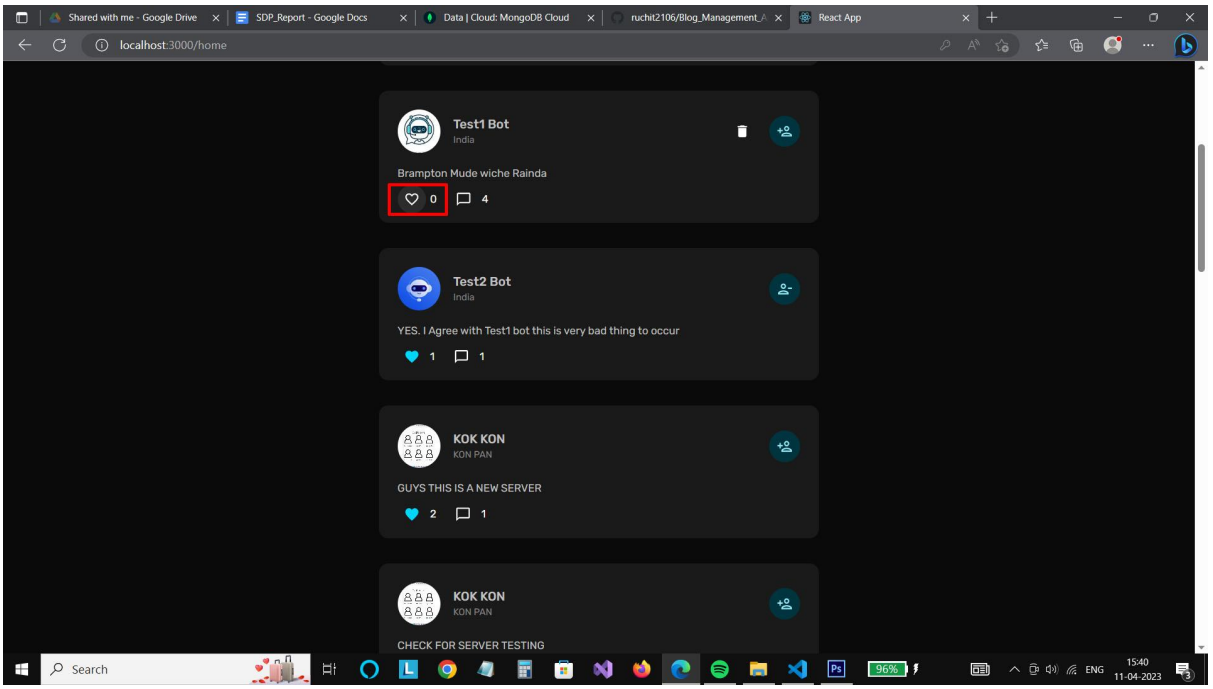
Add a Blog with valid details:- (Post button now clickable)



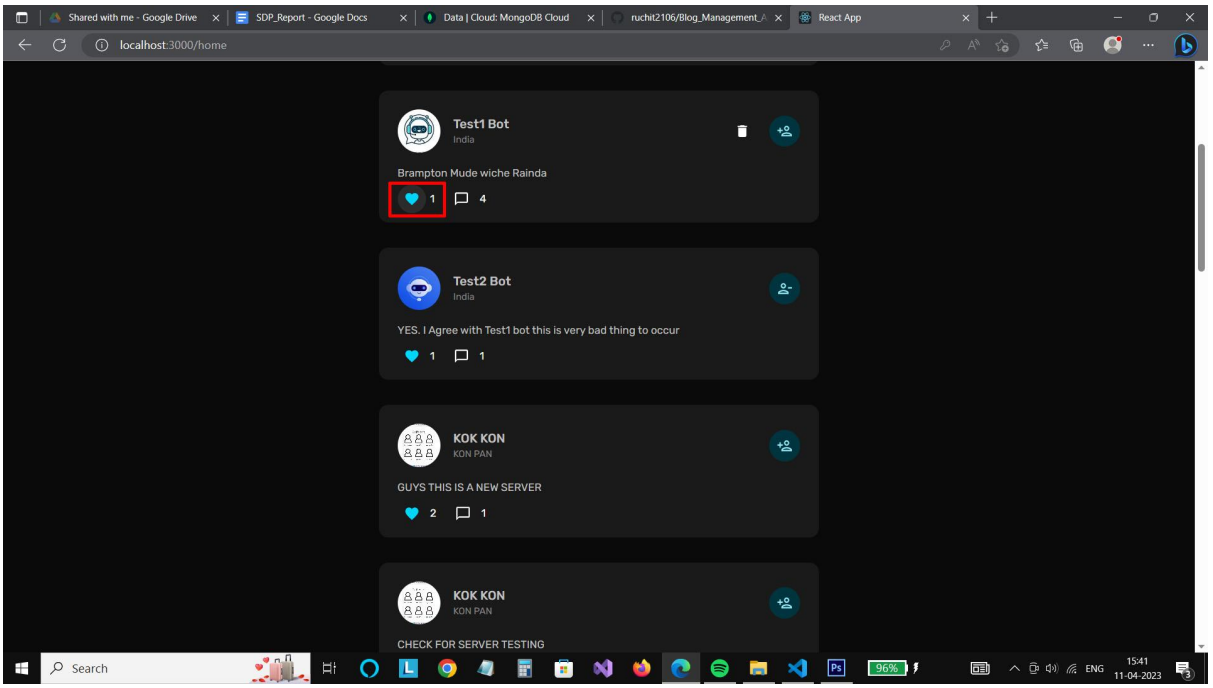
Output:-



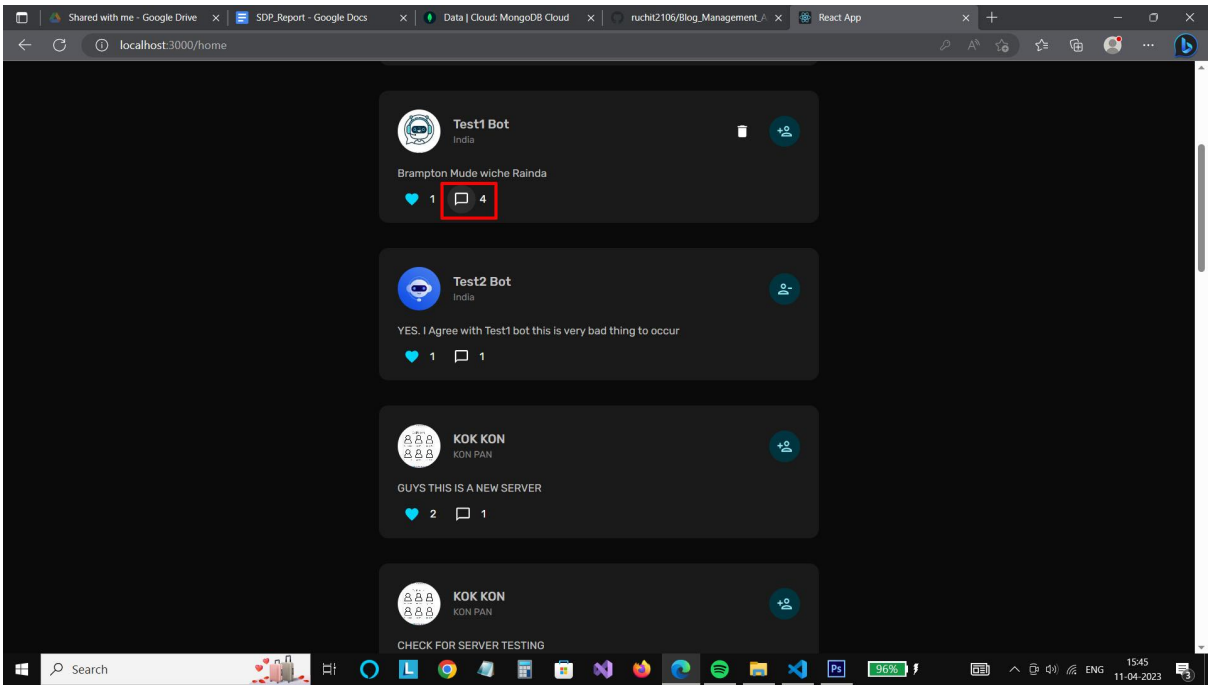
Like a blog:-



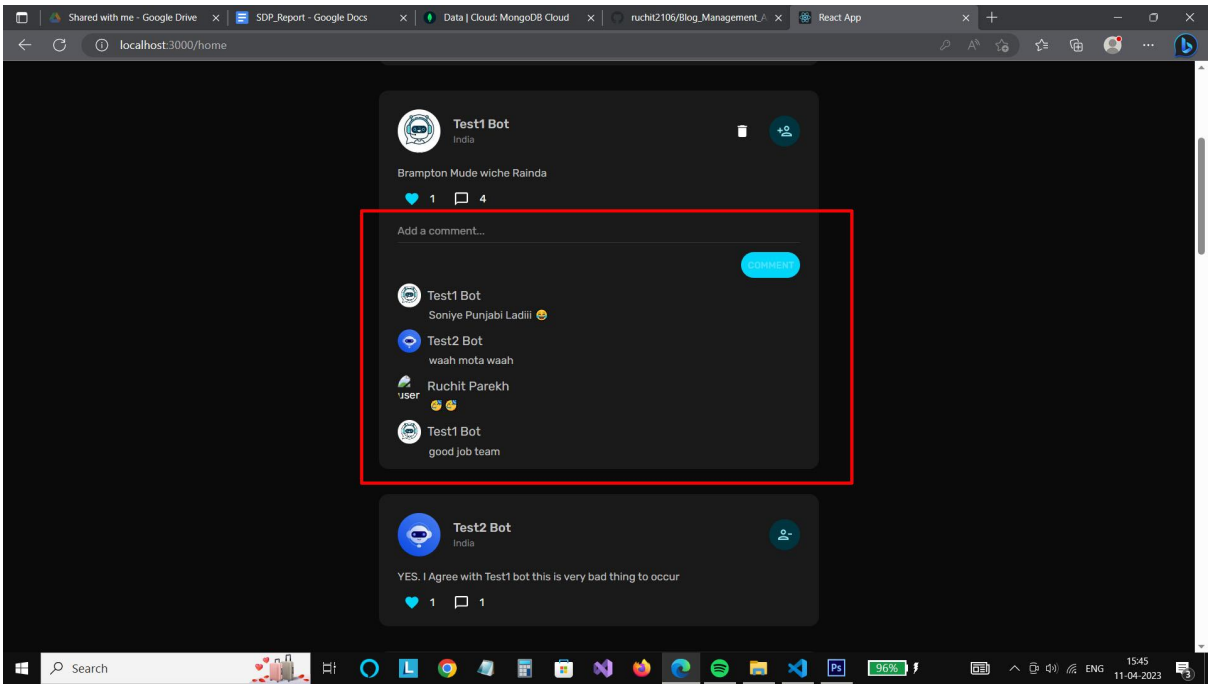
Output:-



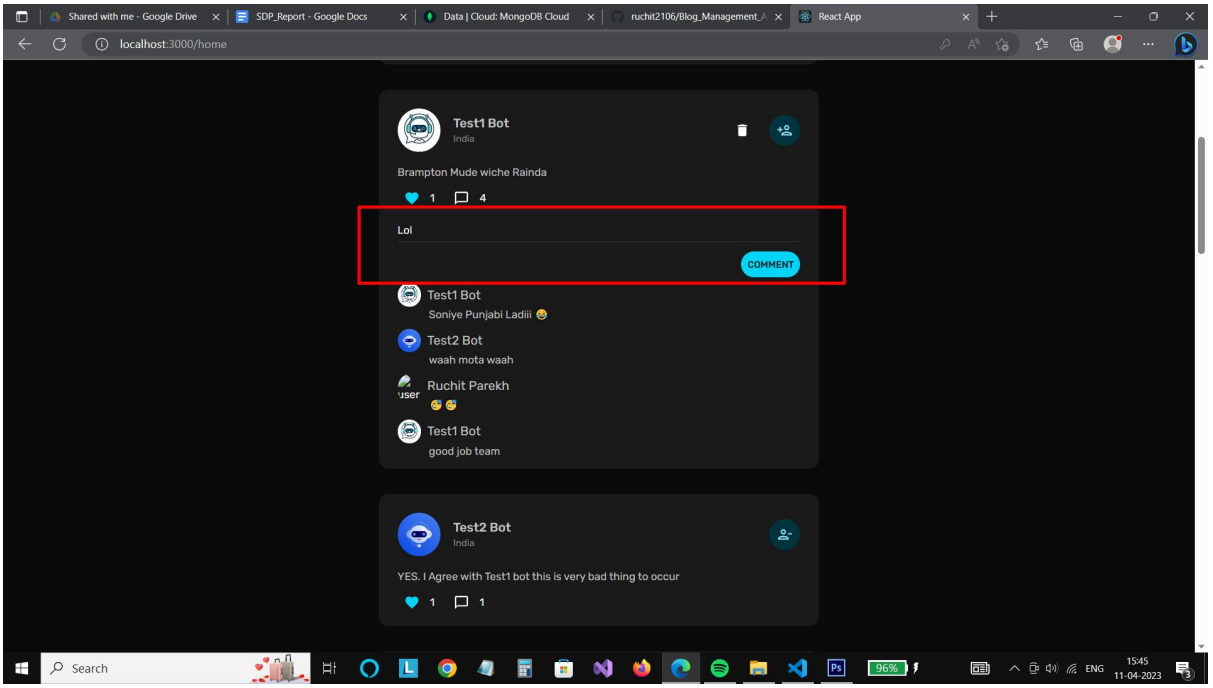
Comment on a blog:-



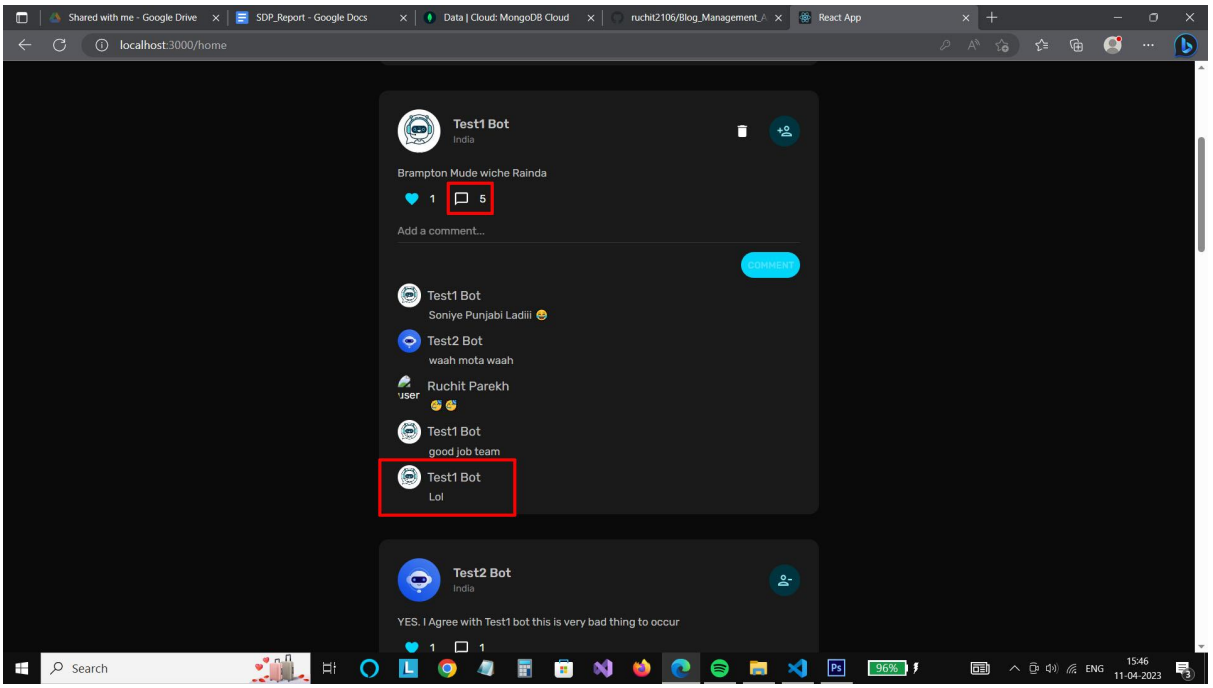
Comment button not clickable because there is no text written in the field



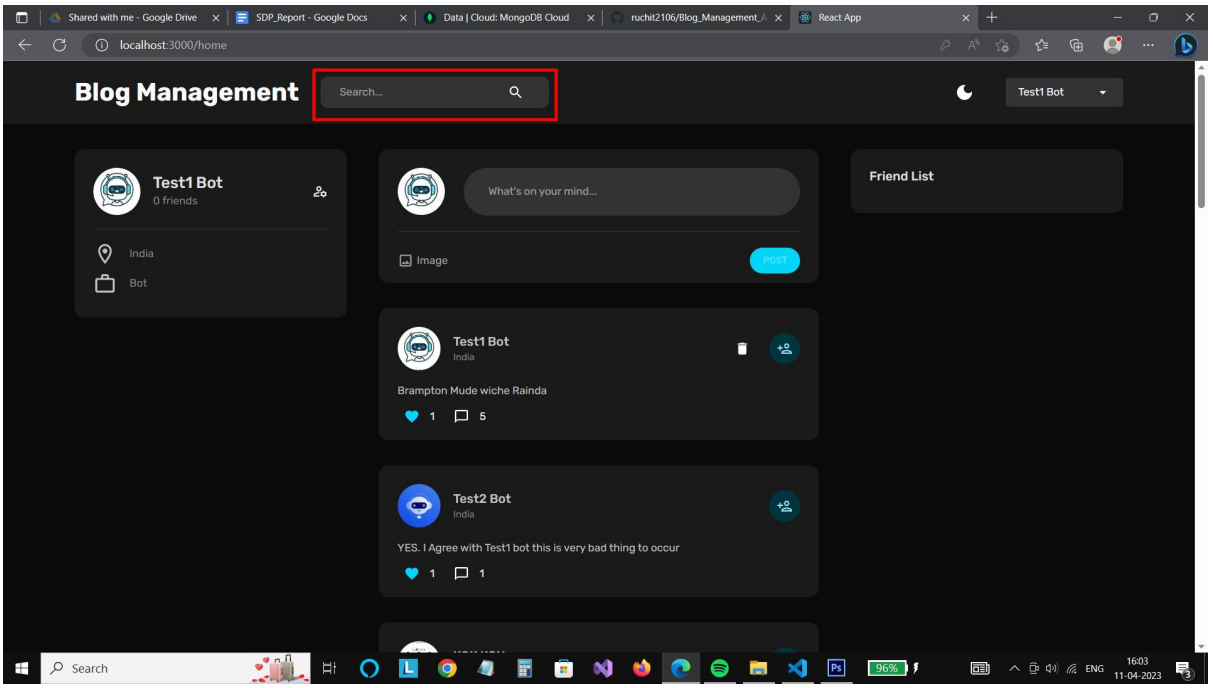
Comment button now clickable



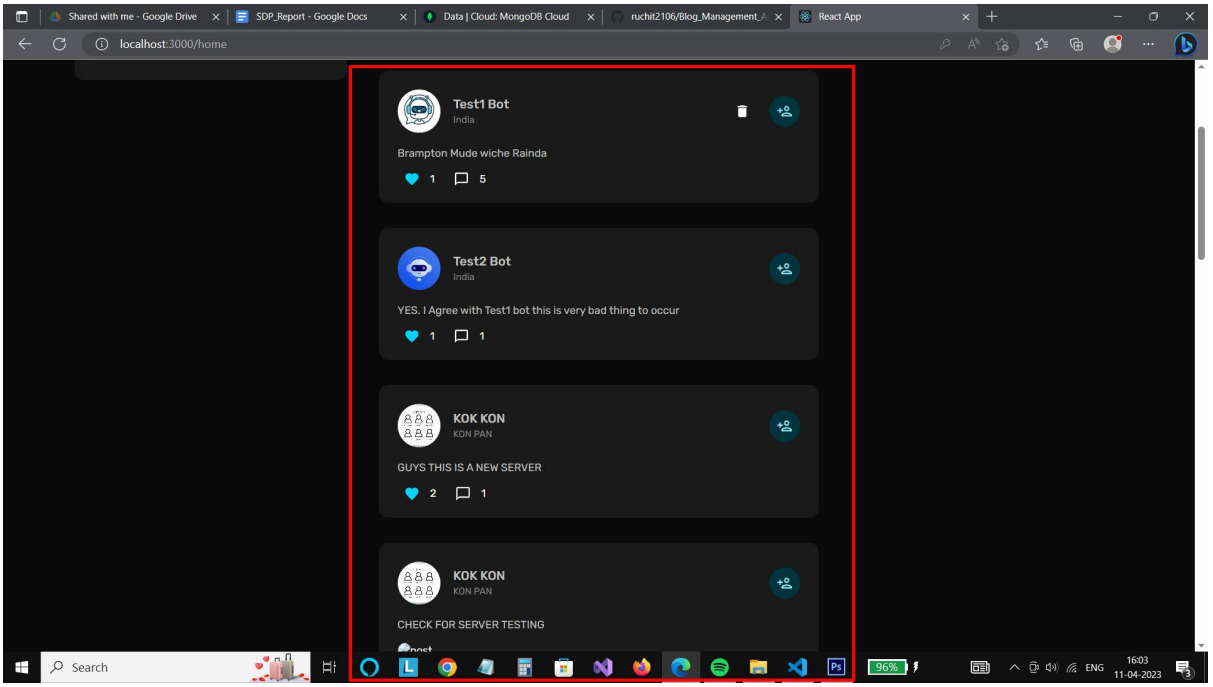
Output:-


Shared with me - Google Drive
SDP_Report - Google Docs
Data | Cloud: MongoDB Cloud
ruchit2106/Blog_Management_A
React App
localhost:3000/home
Test1 Bot
India
Brampton Mude wiche Rainda
1
5
Add a comment...
COMMENT
Test1 Bot
Soniye Punjabi Ladiii
Test2 Bot
waah mota waah
Ruchit Parekh
Test1 Bot
good job team
Test1 Bot
Lol
Test2 Bot
India
YES. I Agree with Test1 bot this is very bad thing to occur
1
1
Search
96%
ENG
15:46
11-04-2023

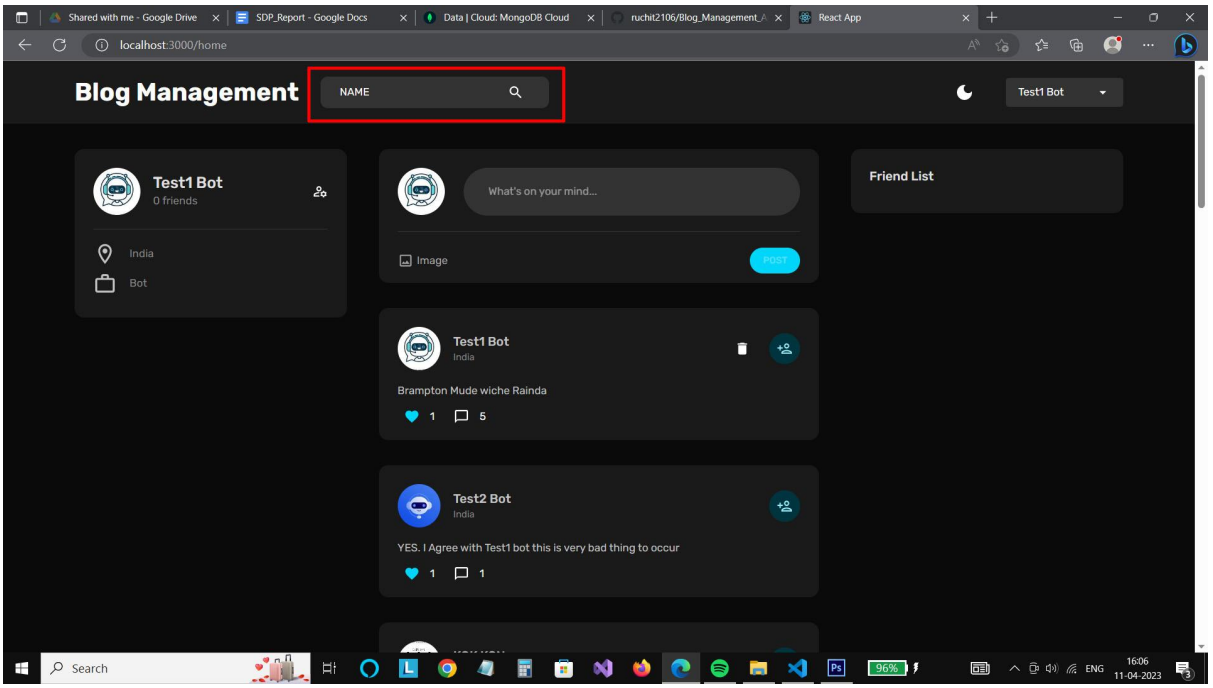
Search a blog with no details:-



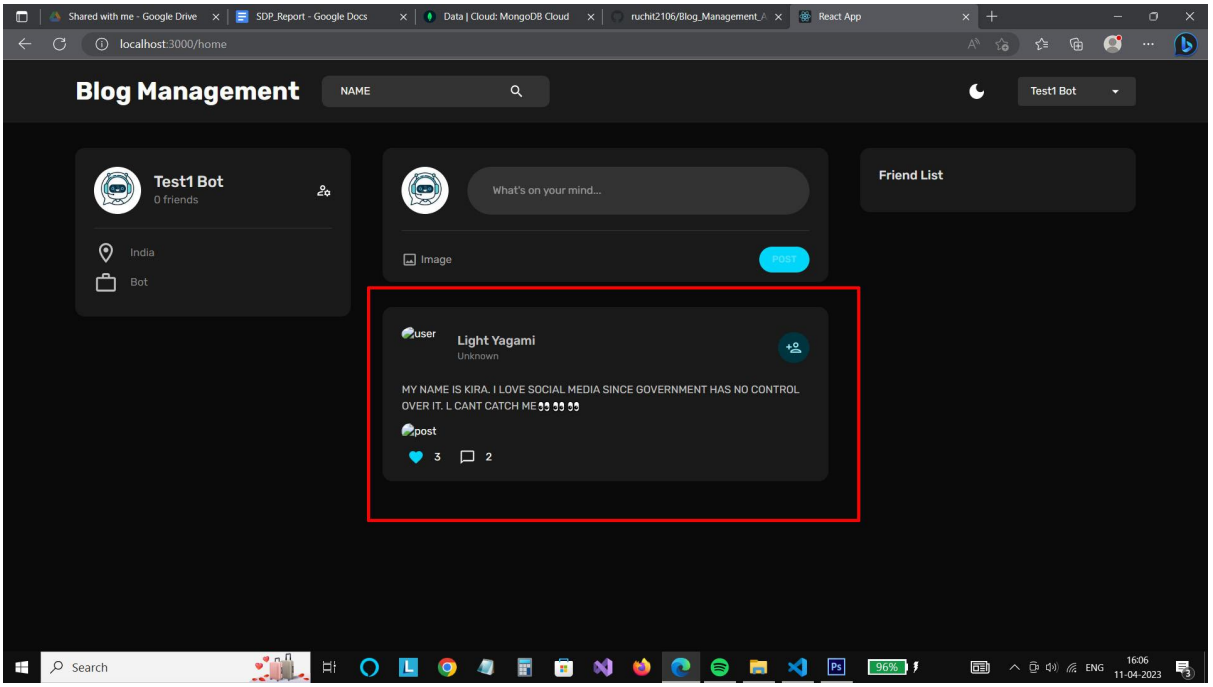
Output:-



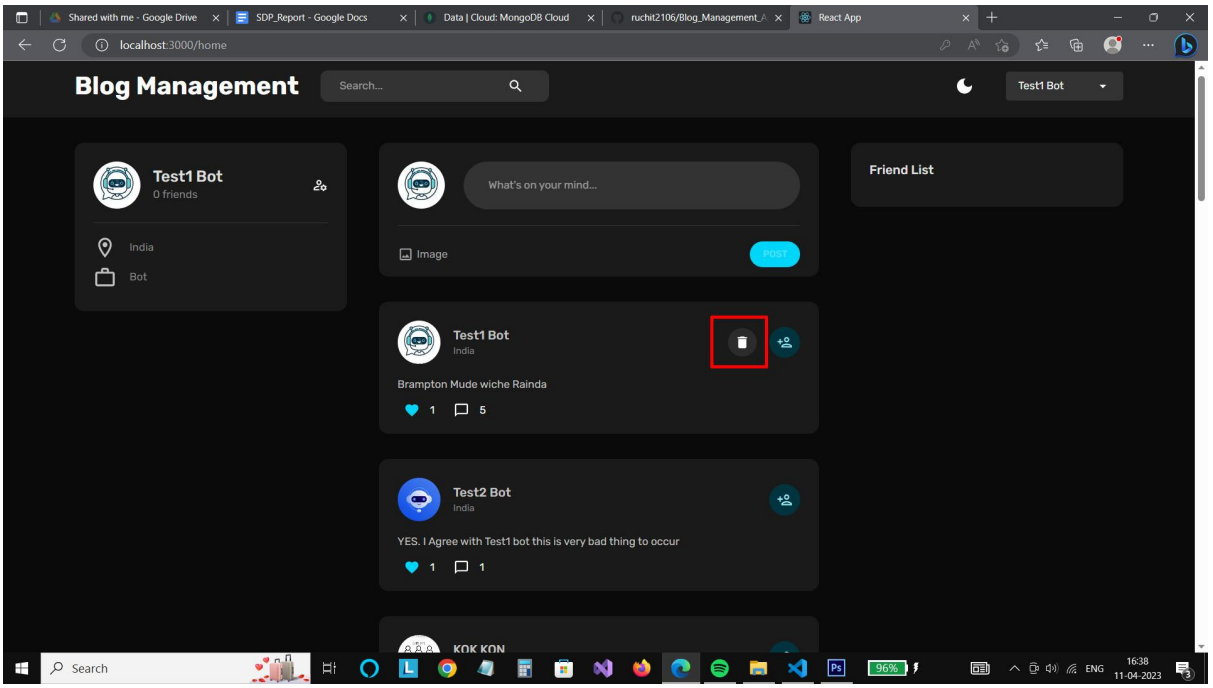
Search a blog with details:-



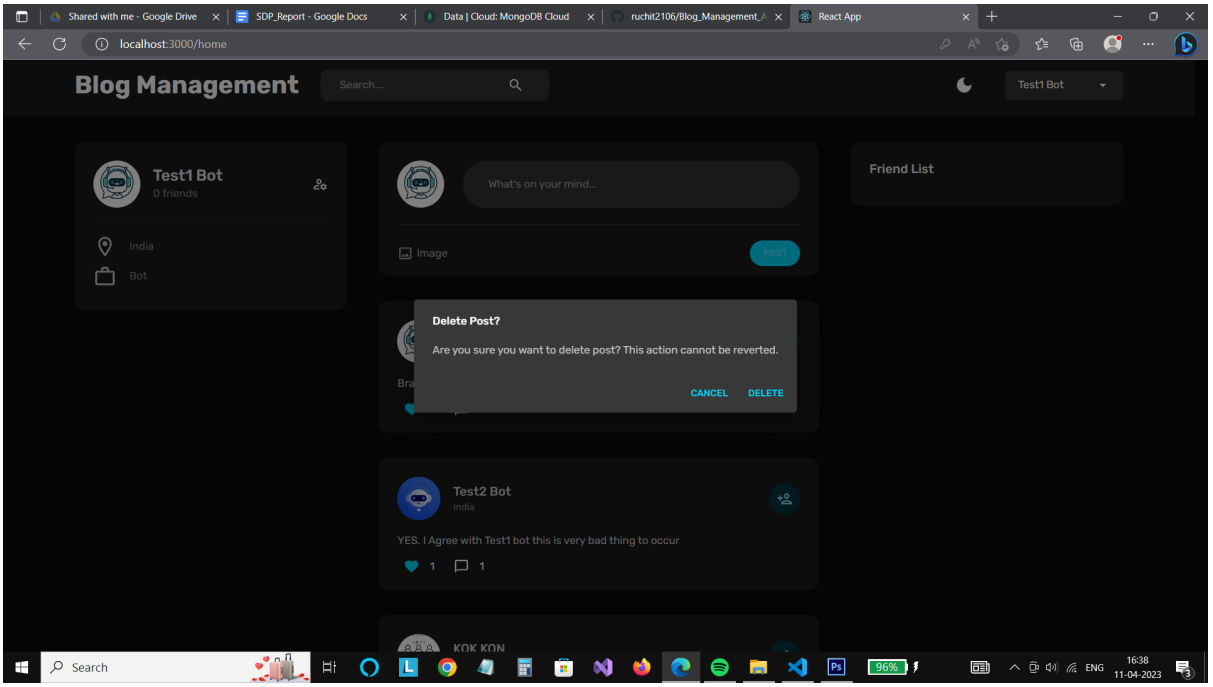
Output:-

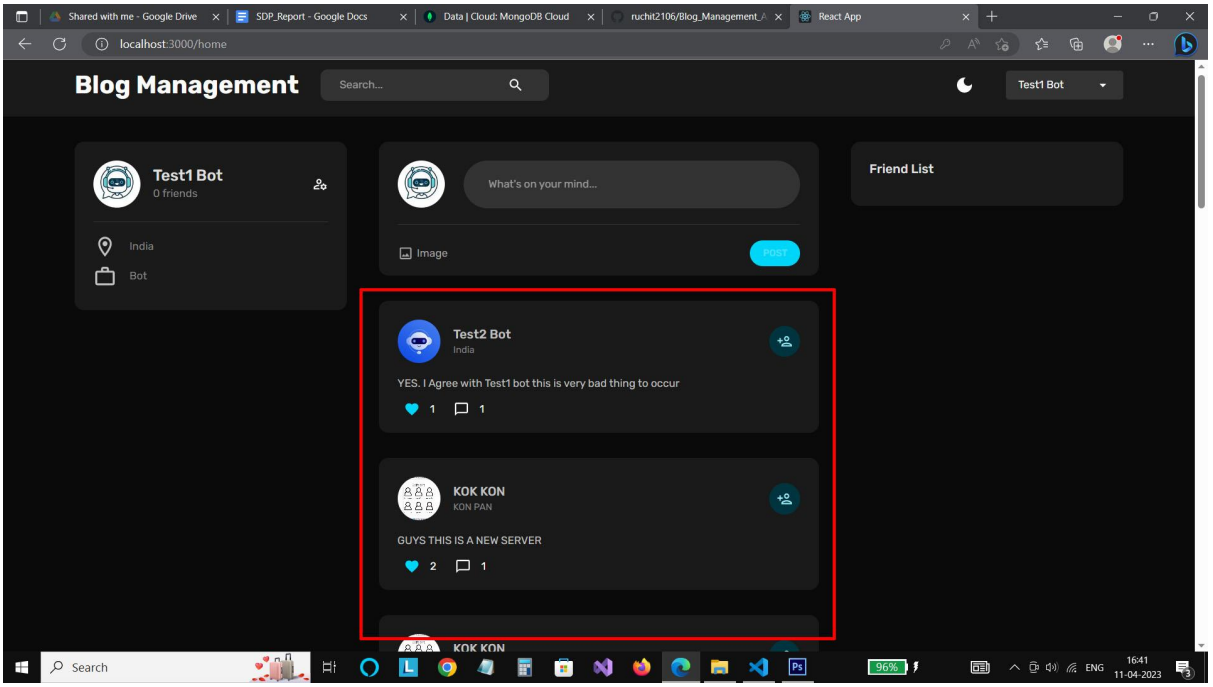


Remove a Post:-

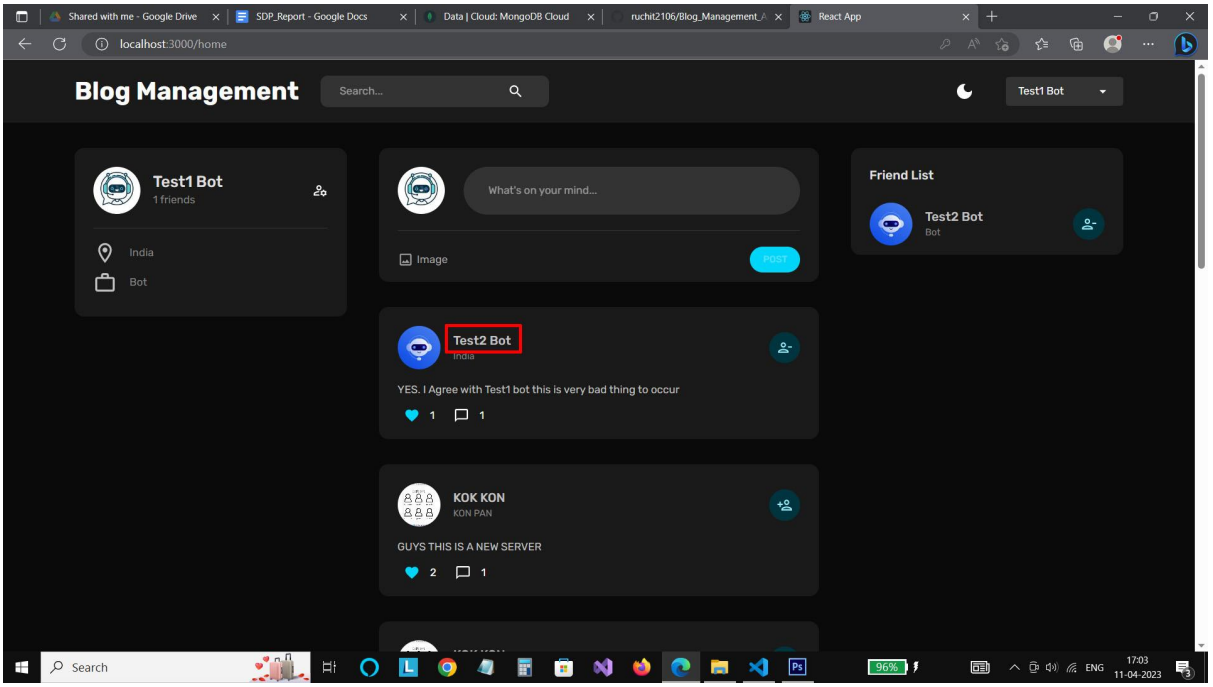


Output:-

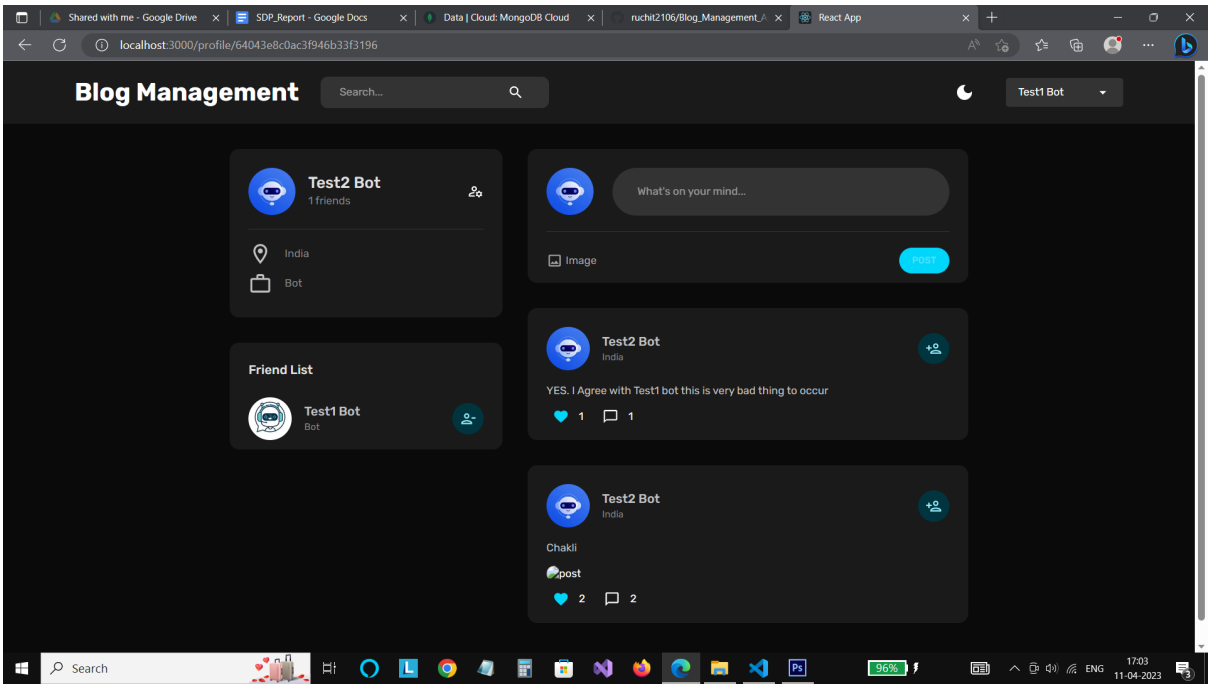



Blog Management
Search...
Test1 Bot
Test1 Bot
0 friends
India
Bot
What's on your mind...
Image
POST
Friend List
Test2 Bot
India
YES. I Agree with Test1 bot this is very bad thing to occur
1
1
KOK KON
KON PAN
GUYS THIS IS A NEW SERVER
2
1
Search
96%
ENG
16:41
11-04-2023

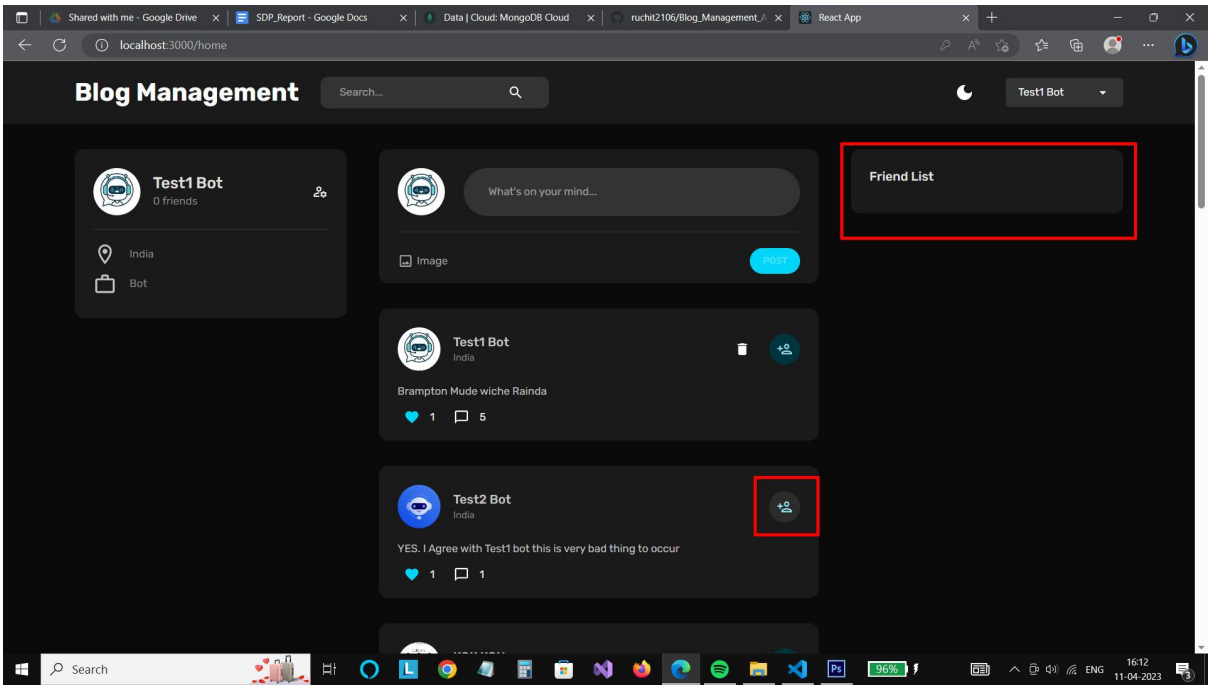
View a User:-


Blog Management
Search...
Test1 Bot
Test1 Bot
1 friends
India
Bot
What's on your mind...
Image
POST
Friend List
Test2 Bot
Bot
Test2 Bot
India
YES. I Agree with Test1 bot this is very bad thing to occur
1
1
KOK KON
KON PAN
GUYS THIS IS A NEW SERVER
2
1
Search
96%
ENG
17:03
11-04-2023

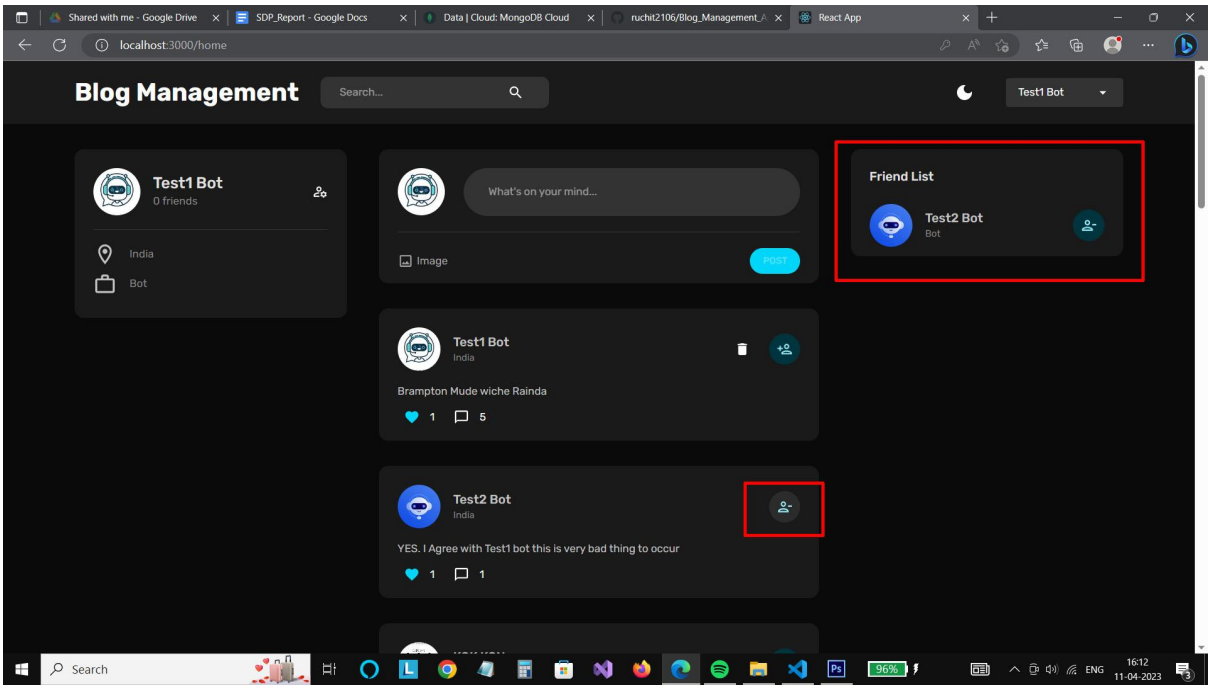
Output:-



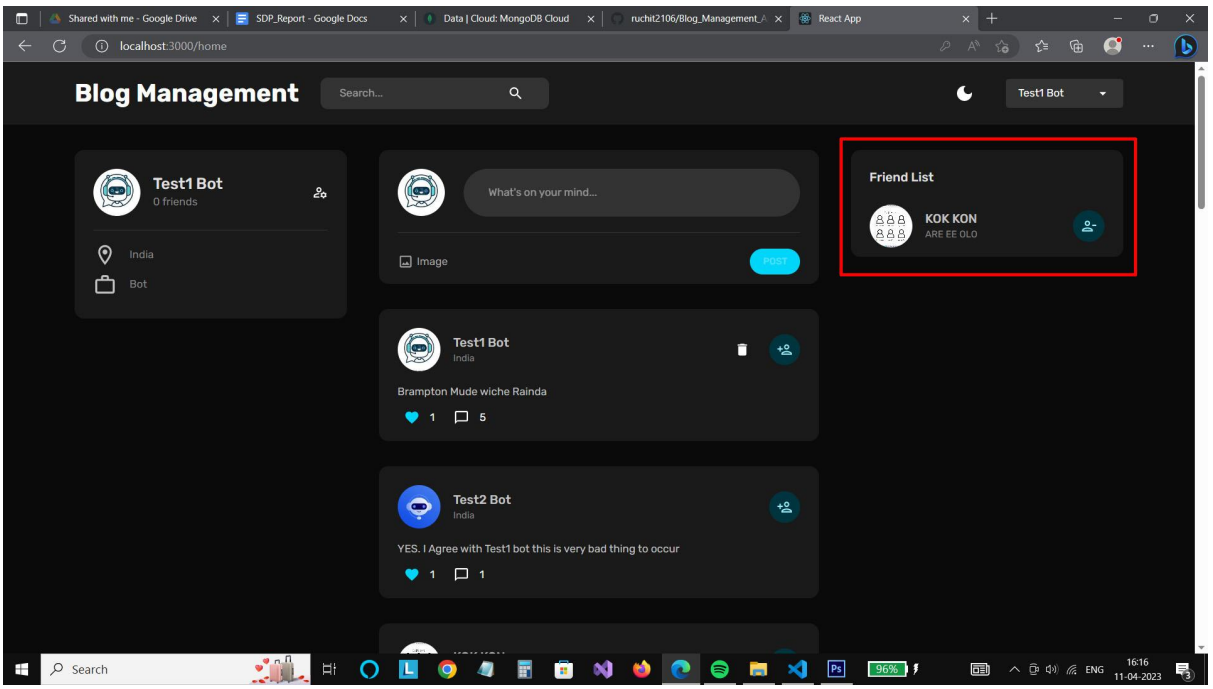
Add a friend:-

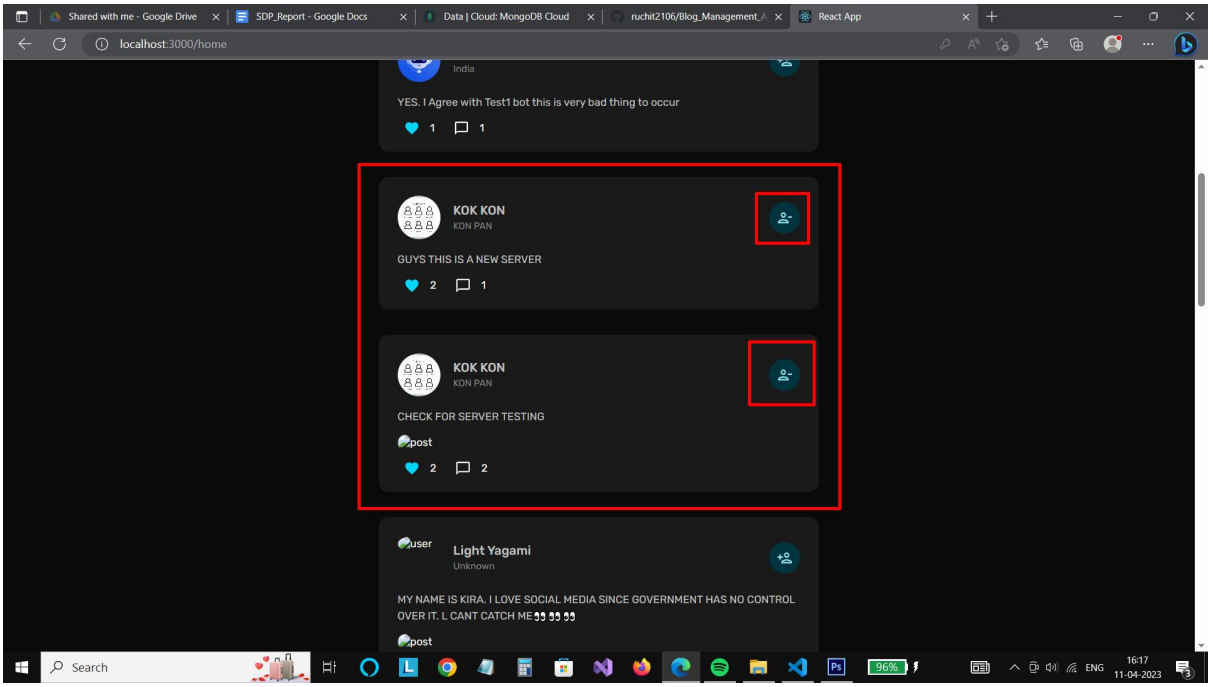


Output:-

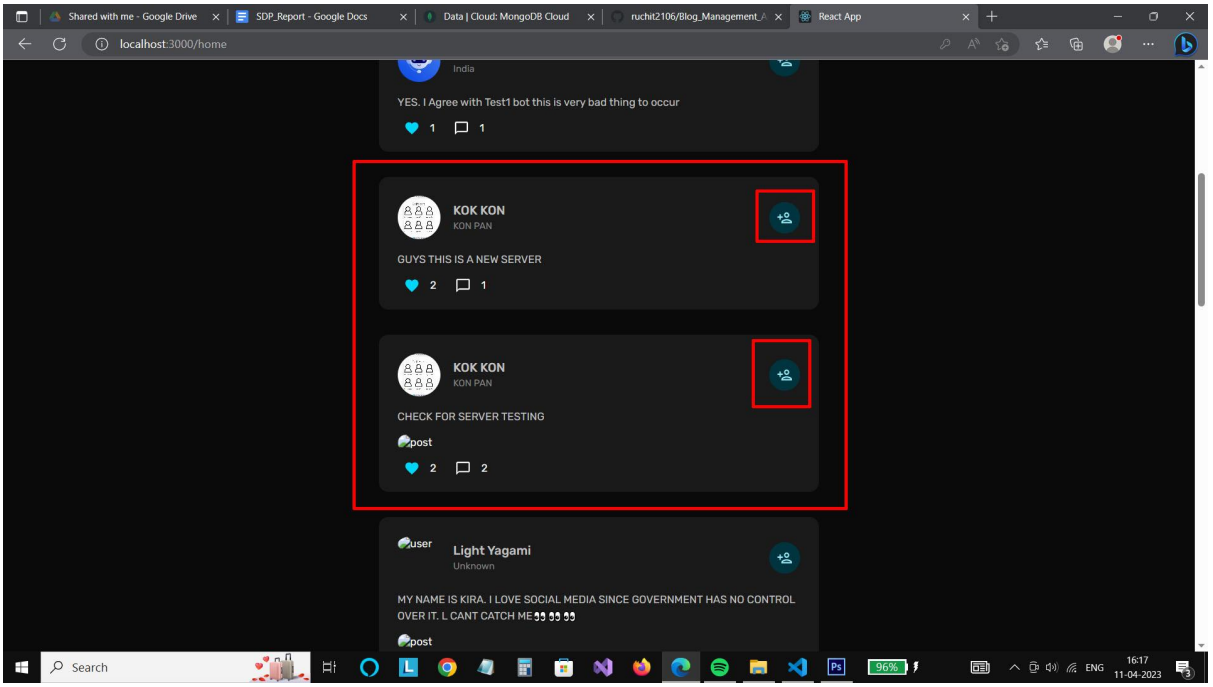


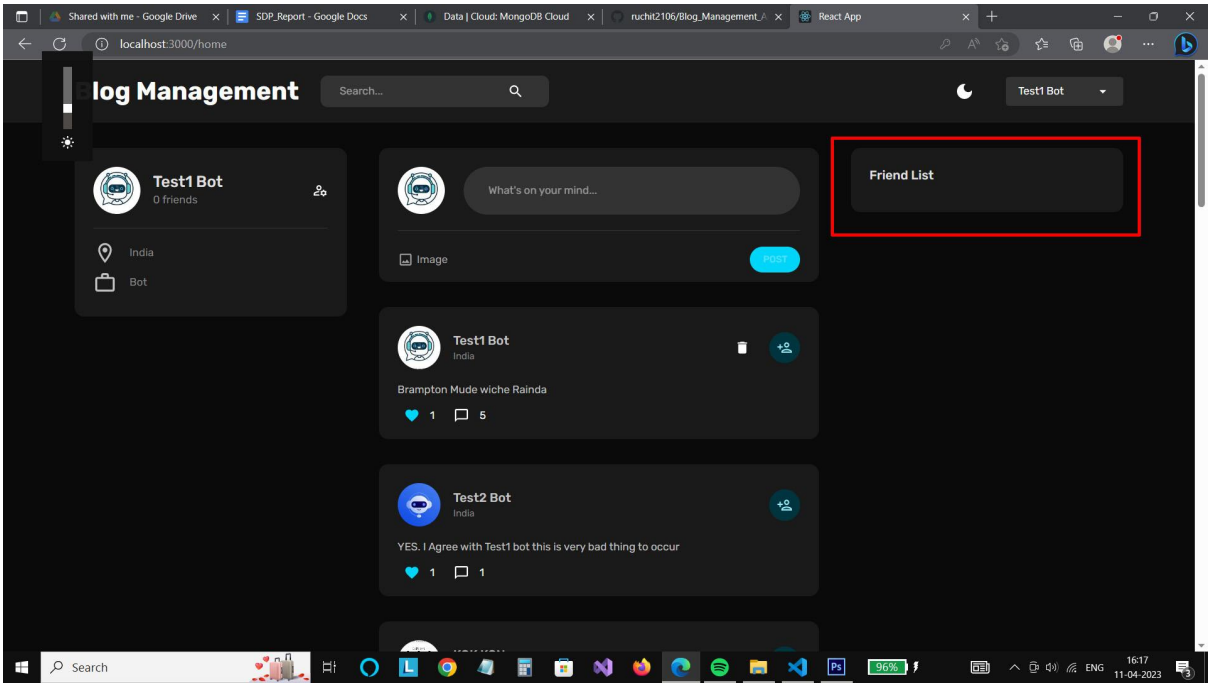
Remove a friend:-



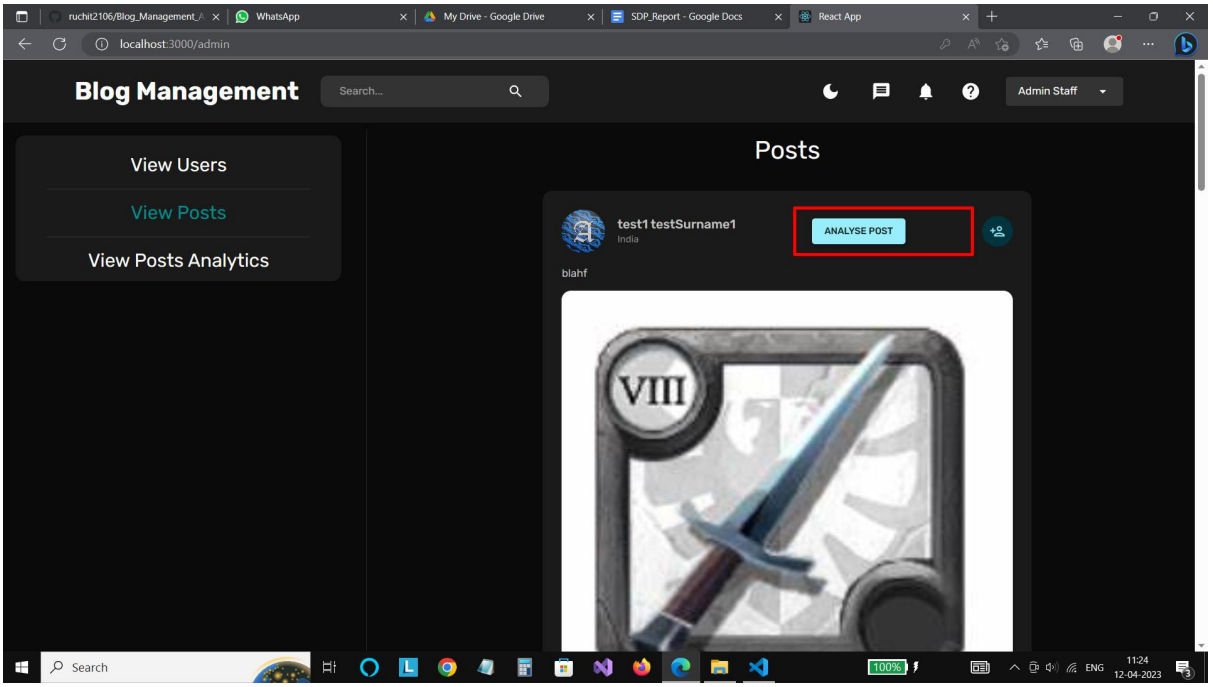

Shared with me - Google Drive
SDP_Report - Google Docs
Data | Cloud: MongoDB Cloud
ruchit2106/Blog_Management
React App
localhost:3000/home
India
YES. I Agree with Test1 bot this is very bad thing to occur
1 1
KOK KON
KON PAN
GUYS THIS IS A NEW SERVER
2 1
KOK KON
KON PAN
CHECK FOR SERVER TESTING
post
2 2
user
Light Yagami
Unknown
MY NAME IS KIRA. I LOVE SOCIAL MEDIA SINCE GOVERNMENT HAS NO CONTROL OVER IT. L CANT CATCH ME 👀👀👀
post
Search
96%
ENG
16:17
11-04-2023

Output:-

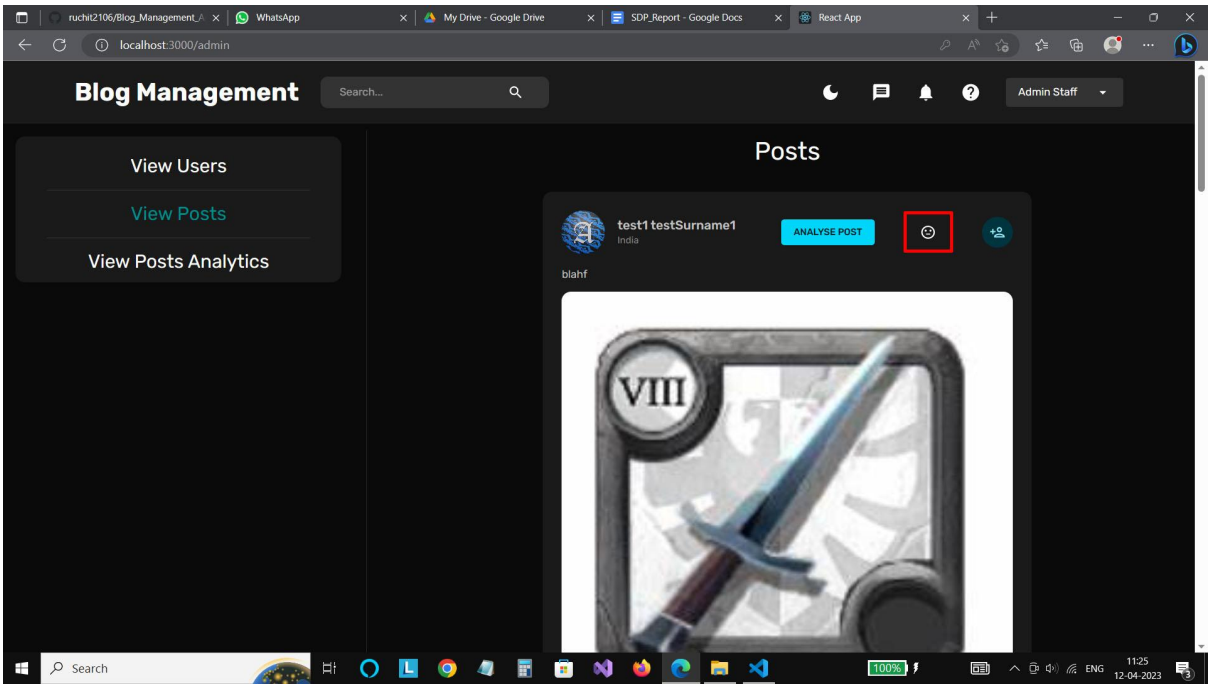

Shared with me - Google Drive
SDP_Report - Google Docs
Data | Cloud: MongoDB Cloud
ruchit2106/Blog_Management
React App
localhost:3000/home
India
YES. I Agree with Test1 bot this is very bad thing to occur
1 1
KOK KON
KON PAN
GUYS THIS IS A NEW SERVER
2 1
KOK KON
KON PAN
CHECK FOR SERVER TESTING
post
2 2
user
Light Yagami
Unknown
MY NAME IS KIRA. I LOVE SOCIAL MEDIA SINCE GOVERNMENT HAS NO CONTROL OVER IT. L CANT CATCH ME 👀👀👀
post
Search
96%
ENG
16:17
11-04-2023



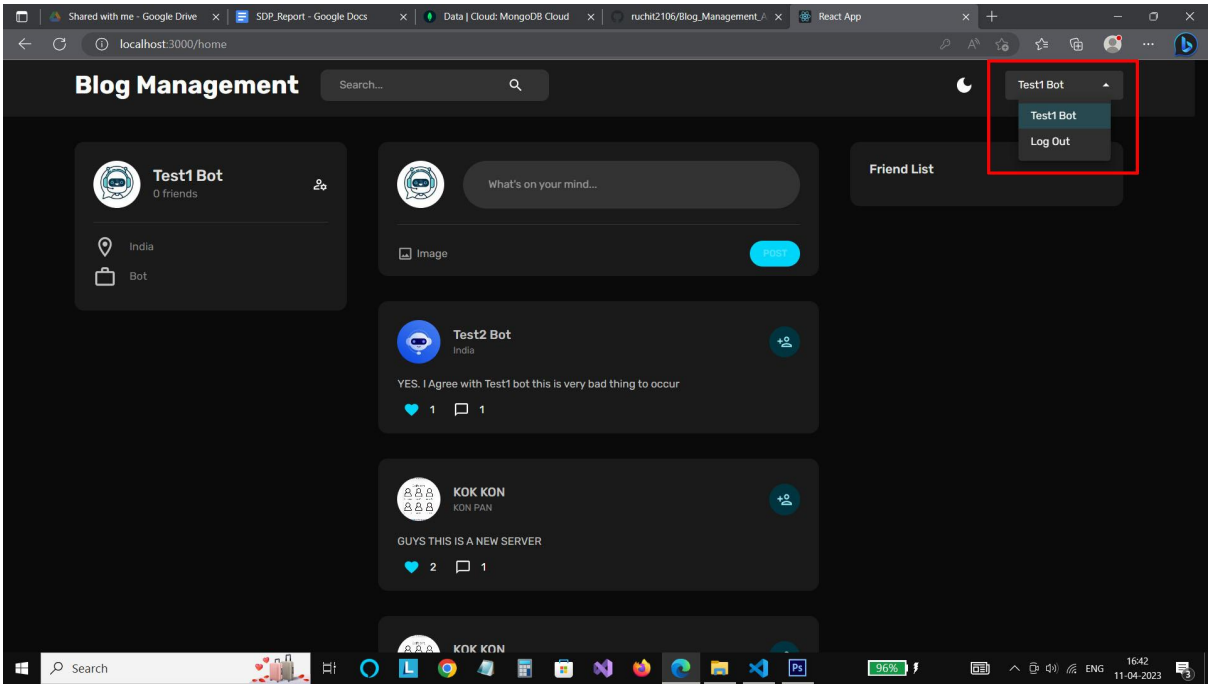
View Sentiment Analysis of a Post:-



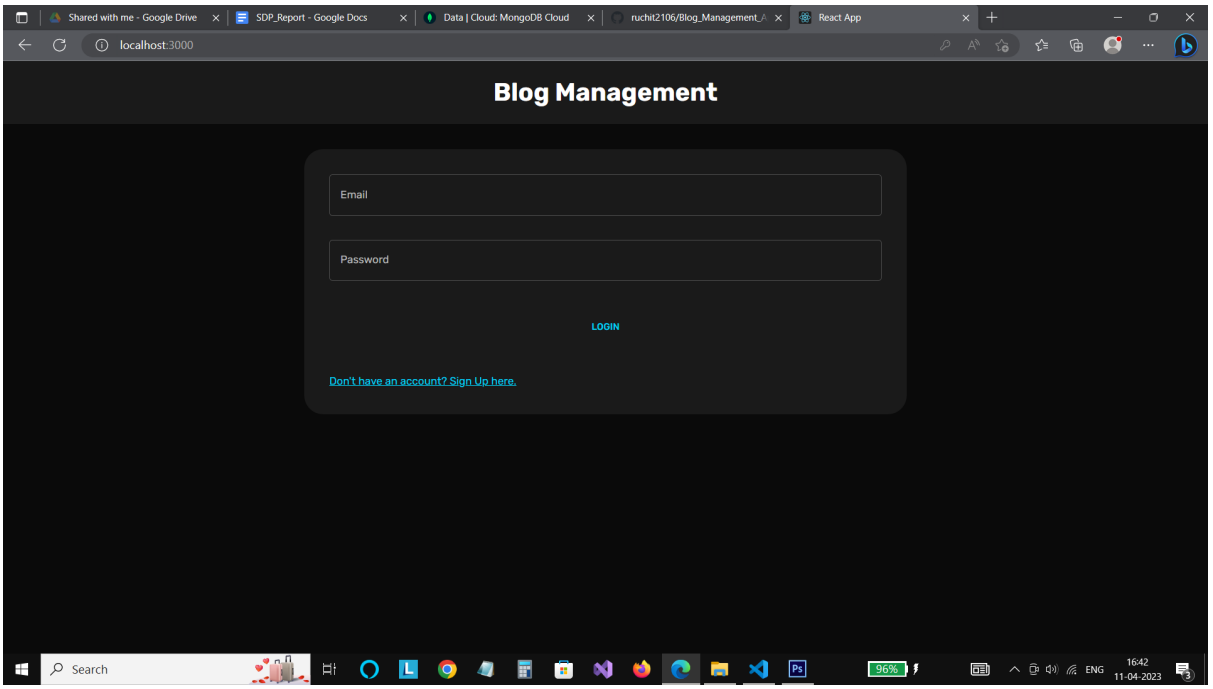
Output:-



Logout:-



Output:-


Shared with me - Google Drive
SDP_Report - Google Docs
Data | Cloud: MongoDB Cloud
ruchit2106/Blog_Management
React App
localhost:3000
Blog Management
Email
Password
LOGIN
Don't have an account? Sign Up here.
Search
96%
ENG
16:42
11-04-2023

Conclusion

The functionalities implemented in the system were done after understanding all the system modules according to the requirements.

Function implemented in the system are :-

- Authenticate User
- Manage Blogs
- Add Like and Comment to particular Blog
- Manage Friends
- View Profile
- View Sentiment Analysis

After the implementation of the system manual testing was performed on the system to determine possible flow of the system.

Limitation and Future Extensions

Limitations of project :-

- If User wants to update a Blog ,then the user will not be able to update it.
- Reporting of a blog cannot be done by the user.
- Categorization of blog based on their description (Topic Modelling) cannot be done

Future Extension :-

- Users will be able to update a blog.
- Users will be able to report a blog.
- Admin will be able to categorize blogs based on their description (Topic Modelling) can be done.

Bibliography

ReactJS :- React
ExpressJS :- Express - Node.js web application framework (expressjs.com)
NodeJs :- Node.js (nodejs.org)
MongoDB :- MongoDB Atlas: Cloud Document Database | MongoDB
Material UI:- React Components - Material UI (mui.com)
For Error solving :- Stack Overflow - Where Developers Learn, Share, & Build Careers