Dharmsinh Desai University, Nadiad
Faculty of Technology

Department of Computer Engineering
B.Tech - C.E. Sem - VI

Subject:- System Design Practice

Project Title:- Blog Management System

Guided By:- Prof. Brijesh S. Bhatt

Name:- Ruchit Parekh (CE006)
Gautam Rathod (CE009)
Aadit Baldha (CE016)

Dharmsinh Desai University, Nadiad

Department of Computer Engineering

Faculty of Technology

CERTIFICATE

This is to certify that the practical / term work carried out in the subject of

Software Design Practice
is the bonafide work of

Ruchit Parekh (CE-006) [20CEUS035]
Gautam Rathod (CE-009) [20CEUTG136]
Aadit Baldha (CE-016) [20CEUON123]

of B. Tech. Semester-VI Computer Engineering during the academic year

2022-2023.

Prof. Brijesh S. Bhatt
Associate Professor
Computer Engg. Department
Faculty of Technology
D.D.U., Nadiad

Dr. C. K. Bhensdadia
Professor and HoD,
Computer Engg. Department
Faculty of Technology
D.D.U.,Nadiad

Index

Abstract

We have built a MERN Stack-based Online Blog Management System. Users of this application can post blogs and view others blogs to share information around the world. This application allows users to interact with other users and upload blogs to spread useful information in an informative way. Moreover the application allows admin to analyze the sentiment of a particular post based on their comments.

Introduction

A blog is a type of website that is updated regularly with new content. Most blogs contain short, informal articles called blog posts. These posts usually contain some combination of text, photos, videos, and other media. At its core, a blog is just a space on the Web that you can create to record and express your opinions, experiences, and interests.

The people who write blogs are called bloggers. From what you hear on the news, you might think bloggers are all a certain type of people—young, politically inclined, and tech- savvy. Or maybe you've heard about bloggers who've written about amazing experiences or ambitious projects, then turned their blogs into bestselling books. While some bloggers do fit these descriptions, most bloggers don't. In fact, there's no "average" blogger—blogs are written by people of all ages and backgrounds and from all walks of life.

Technology Used :-

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

Tools Used:-

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

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 blog name in search blog if it matches with the 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 like button of 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 want 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.

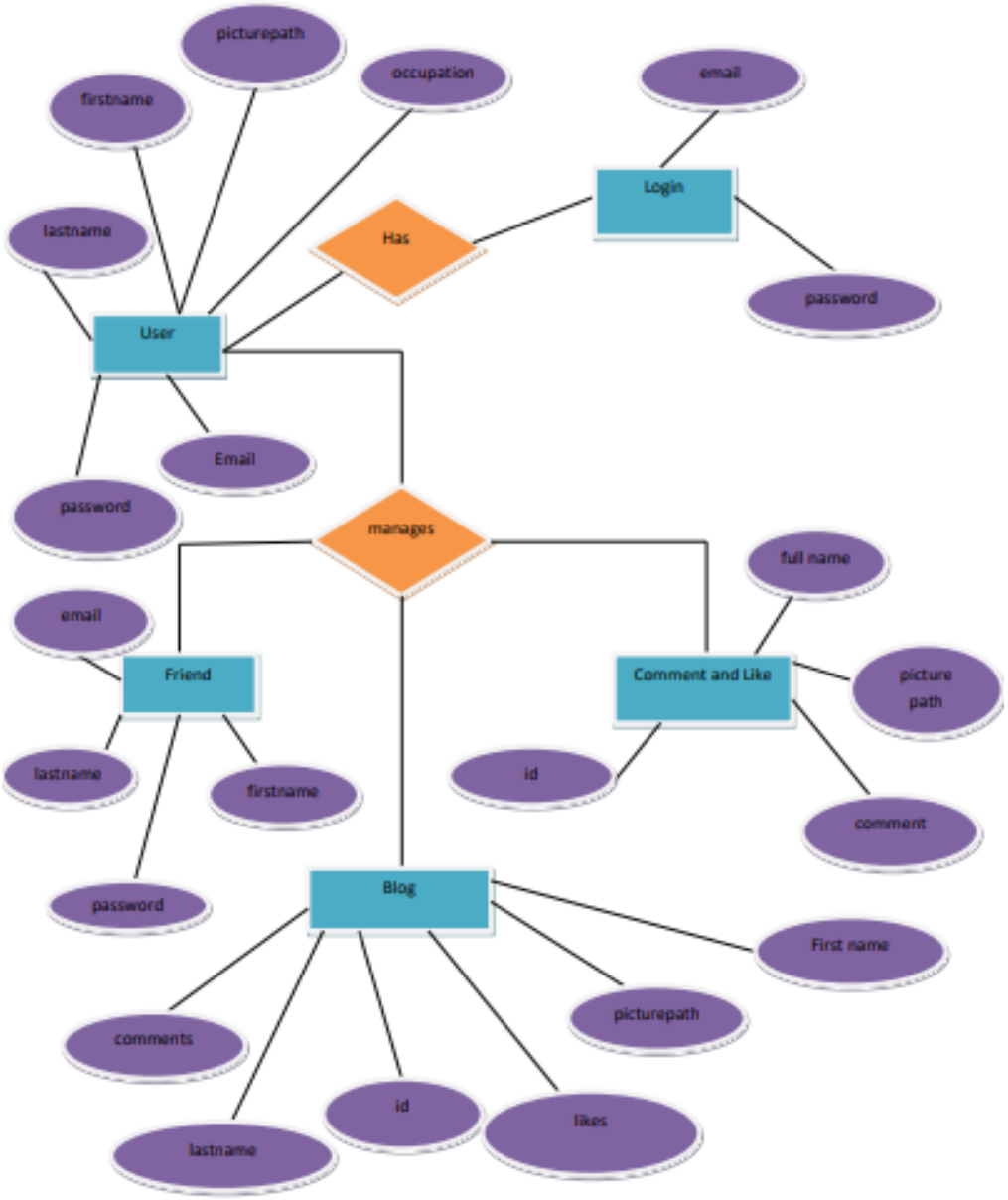
Reliability Requirements:- The system shall have a minimum uptime of 99% excluding time pre-scheduled for maintenance and/or upgrades.

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.

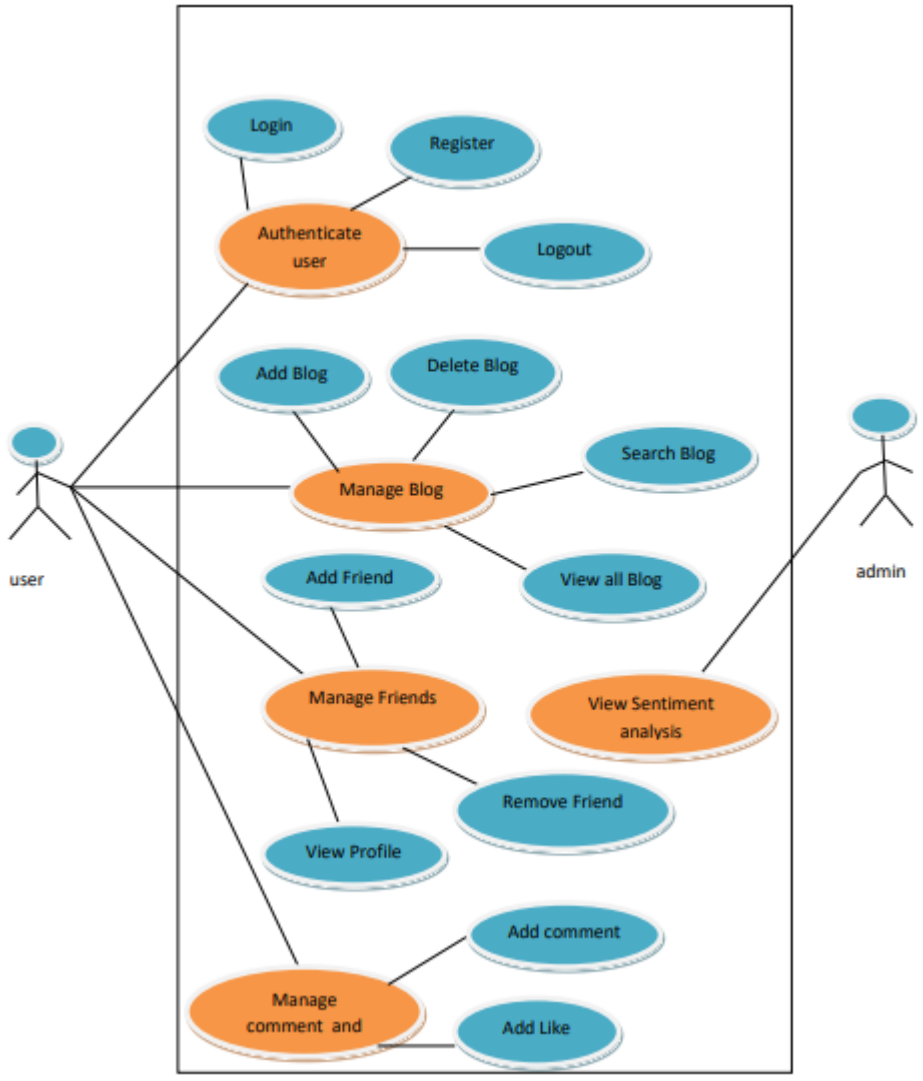
Quality Attributes:- the source code for the system is well documented for ease of maintenance and upgrading the system in future.

Design :-

ER Diagram :-



Use Case Diagram (UML) :-



Models :-

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.
Server side is implemented using NodeJS , ExpressJS and mongoose.
MongoDB is used for data storage.

Module Description :-

1. User 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.

.

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("");
};
```

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 :-

```
#for data conversion from json to readable in python
posts=sys.argv
dict_obj=ast.literal_eval(posts[1])
commentsList=[]
commentsList=np.array(commentsList)
ind=0
for values in dict_obj.values():
    for i in values['comments']:
        commentsList=np.append(commentsList,i)

#################################################################################
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

dict_to_send["sentiment_labels"]=label #needs changes

totalCount=postiveCount+negativeCount+neutralCount #total no. of posts
dict_to_send["totalComments"]=totalCount
dict_to_send["postiveComments"]=postiveCount
dict_to_send["negaticeComments"]=negativeCount
dict_to_send["neutralComments"]=neutralCount

labelSeries=pd.Series(label)
sentiment_counts=labelSeries.value_counts() #needs changes

print(dict_to_send)

plt.pie(sentiment_counts,labels=sentiment_counts.index,autopct='%1.1f%%')
plt.title("Sentiment Distribution")

plt.savefig("D://Sem VI//Projects//SDP//Blog_Management_App-main-Aadit//BLOG_MANAGEMENT_TEST_AADIT//client//public//assets//pieChart.png")
plt.show()
```

Testing :-

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

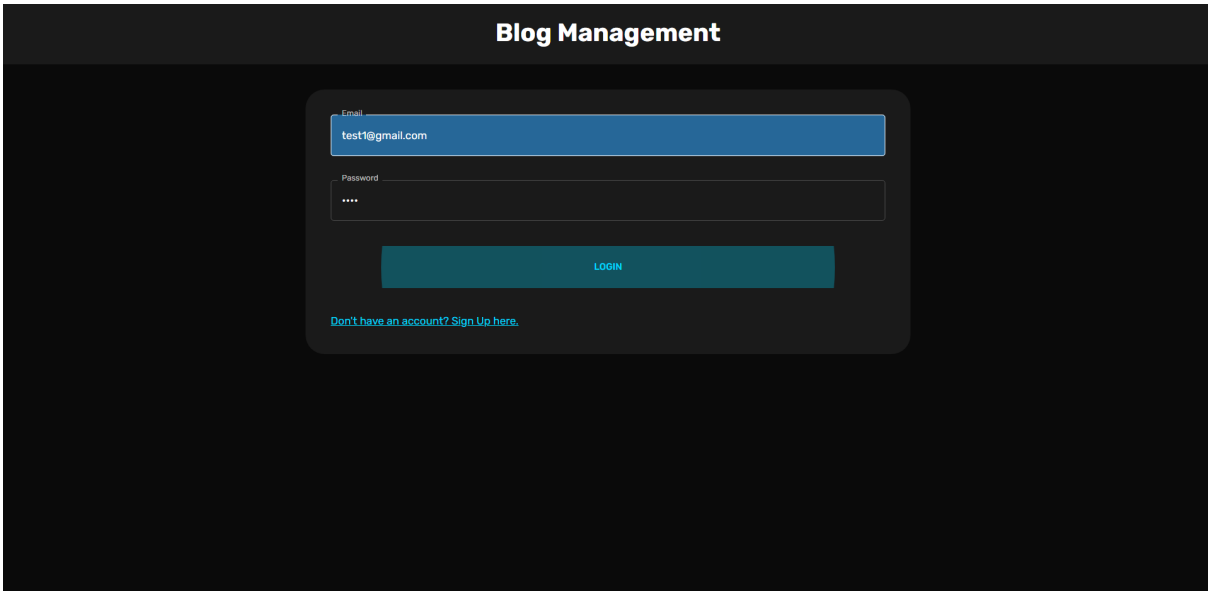
Integration Testing :- Each small part of the application is tested first and after that combine them and whole application testing is performed.

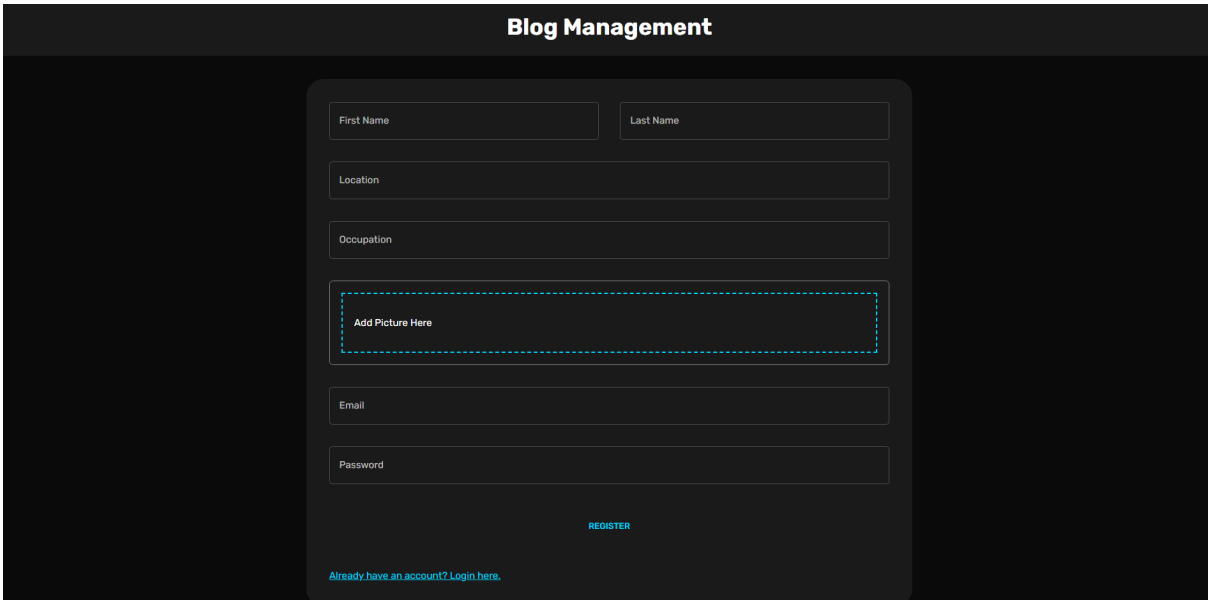
Manual testing :-

Sr No.	Test Scenario	Expected Result	Actual Result	Status
1	Login with incorrect credentials	User should not be able to login	User not able to login	Success
2	Login with correct credentials	User should be able to login	User has login	Success
3	Registration of user with valid details	User should able to register	User register successfully	Success
4	Add a post without Description and image	Post should not be added	Post not added	Success
5	Add a post with valid details	Post should be added.	New Post added	Success
6	Add like on blog	Like should be added on blog	Like added on blog	Success
7	Add a comment on blog	Comment should be added	Comment added	Success
8	Search a Blog	Searched Blog should be displayed	Searched blog displayed	Success
9	Logout	User should logout successfully	User Logout Successfully	Success

Screenshots :-

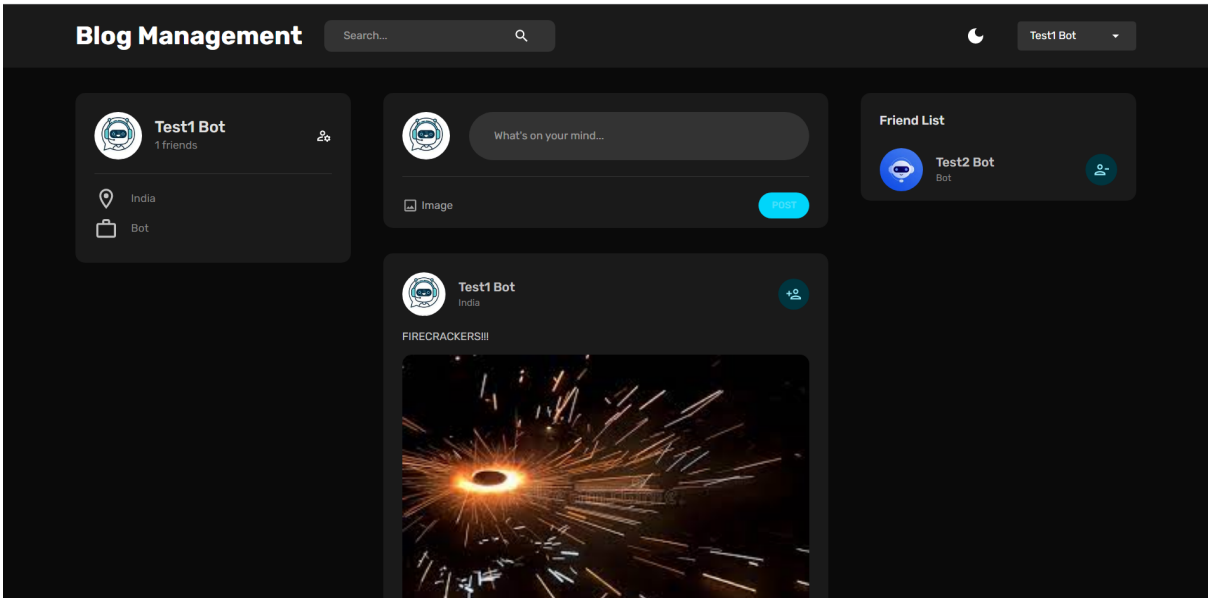
Login and Register :-



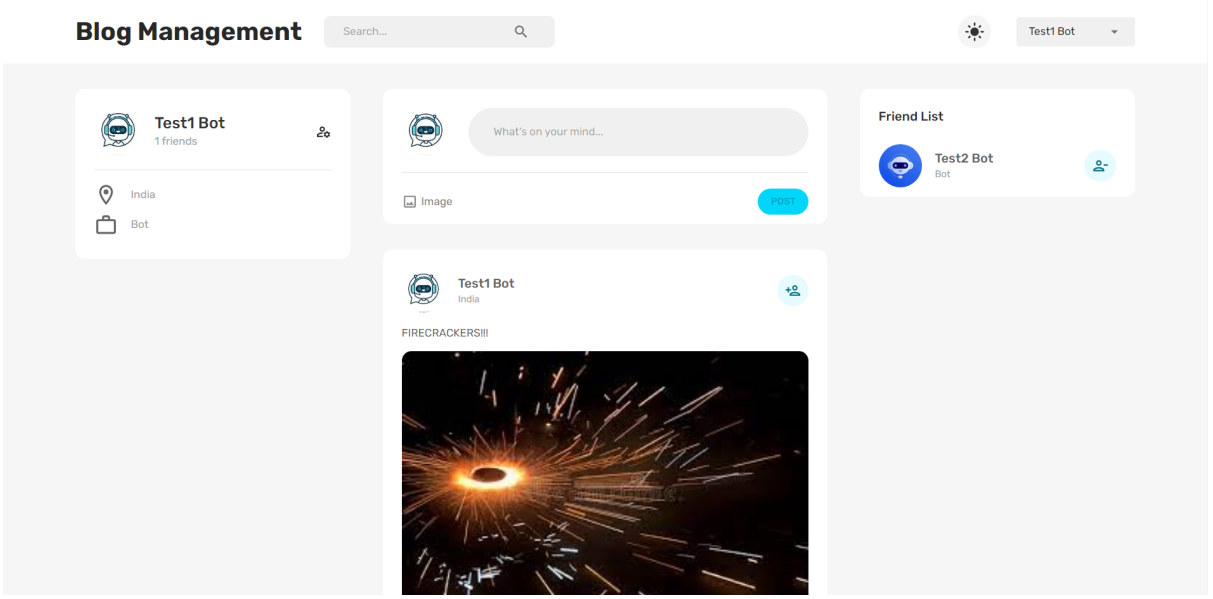


Home Page :-

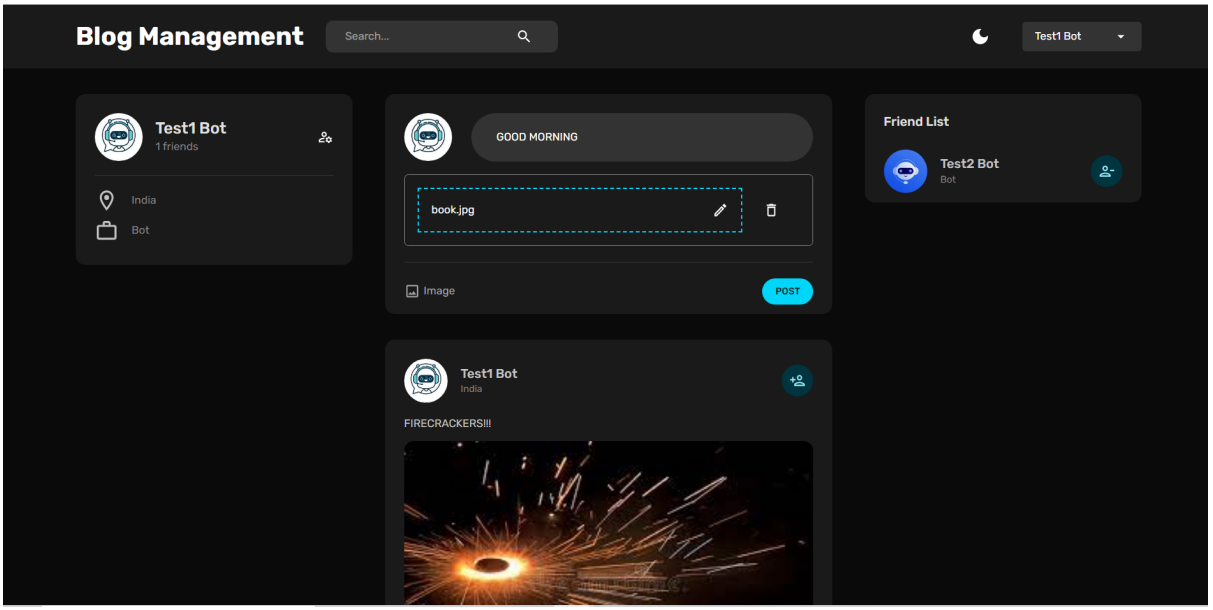
(Dark Mode)



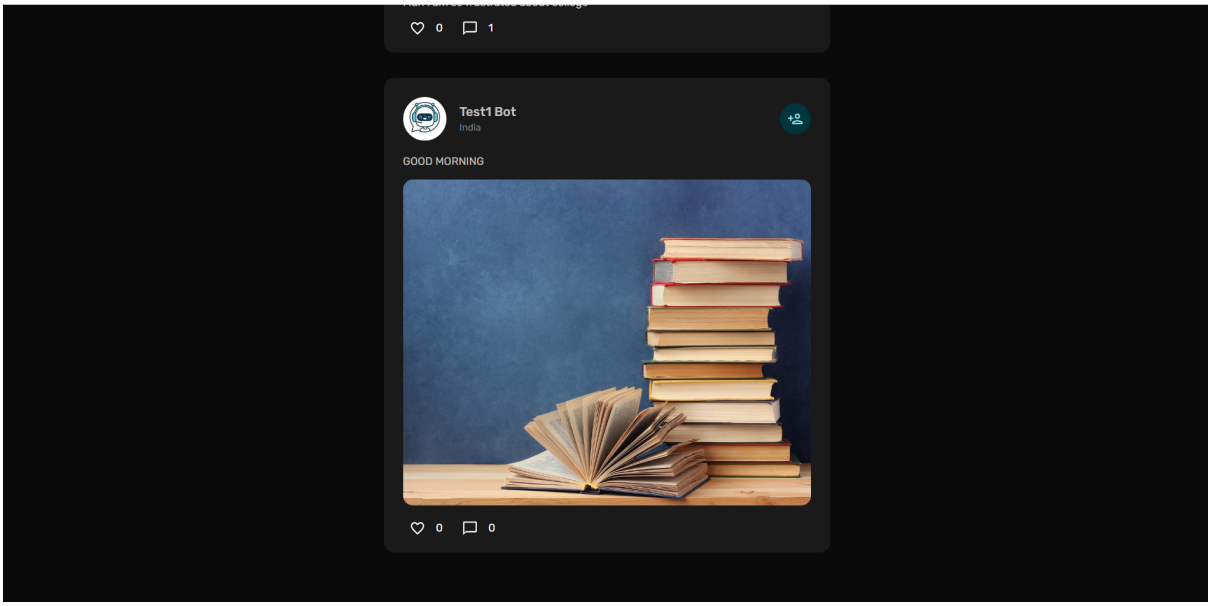
(Light Mode)



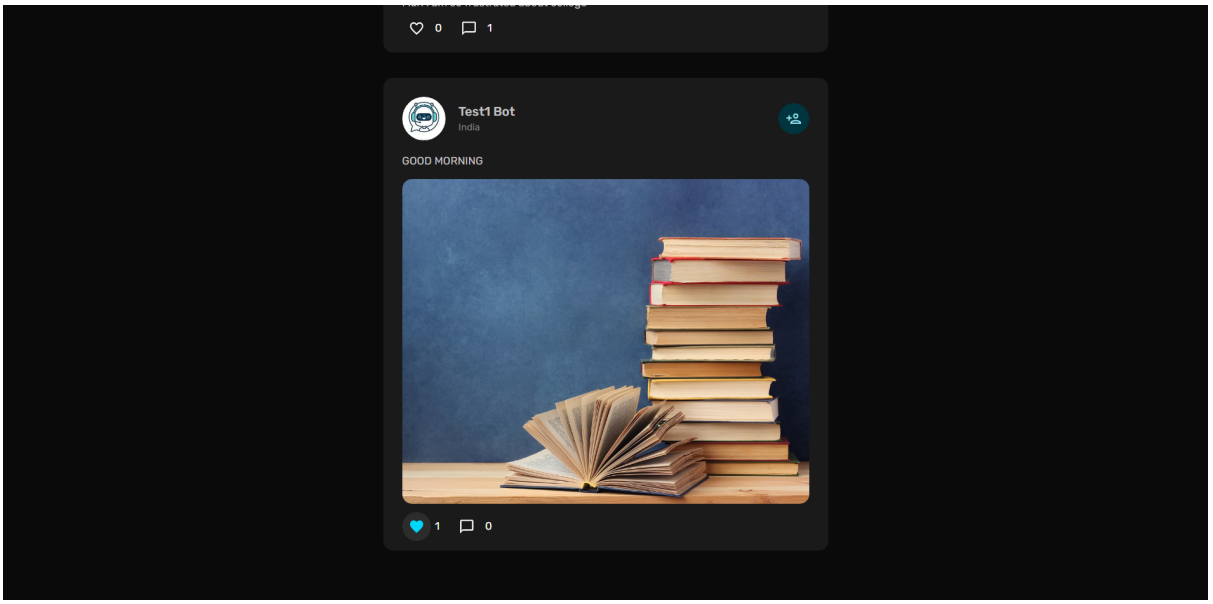
Add a Blog :-



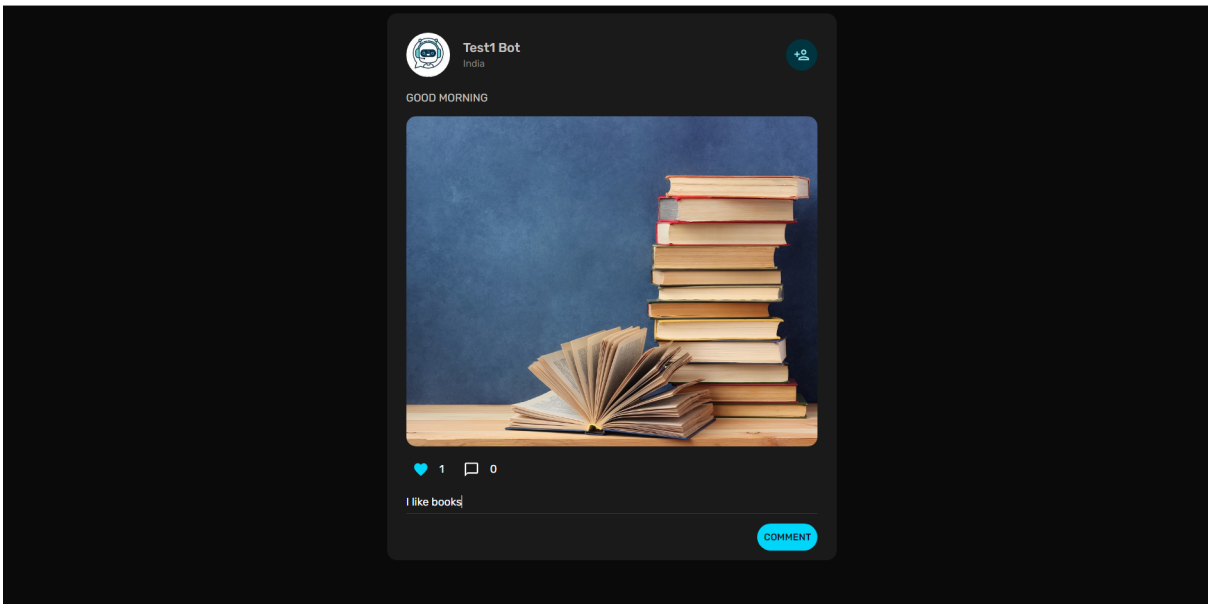
Result After Adding a blog :-

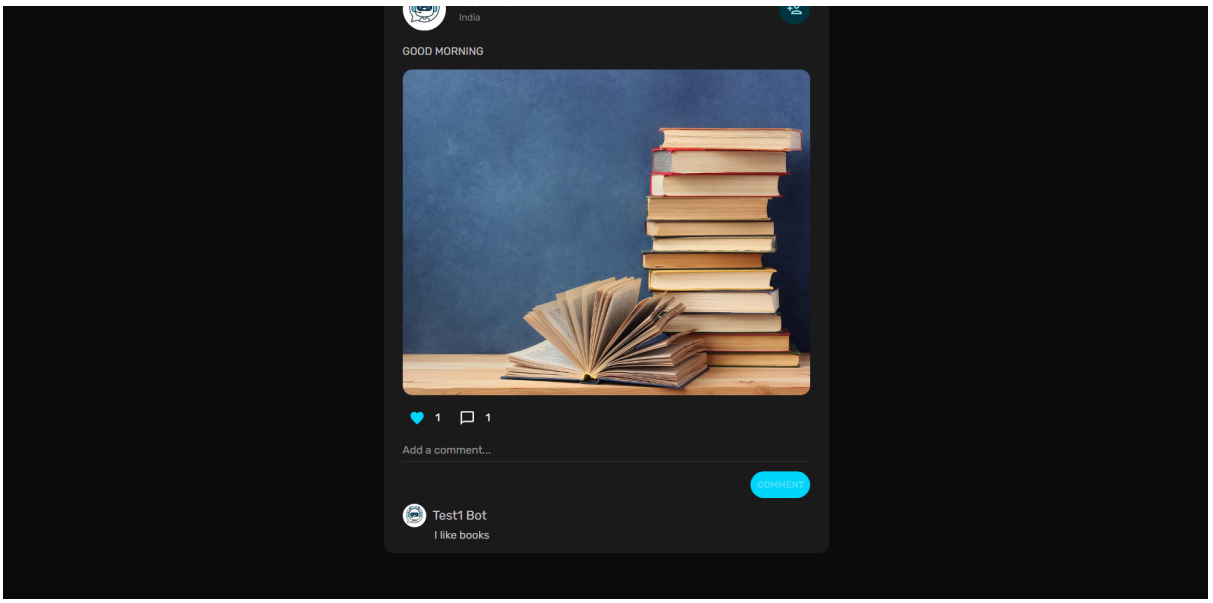


Like a Blog :-

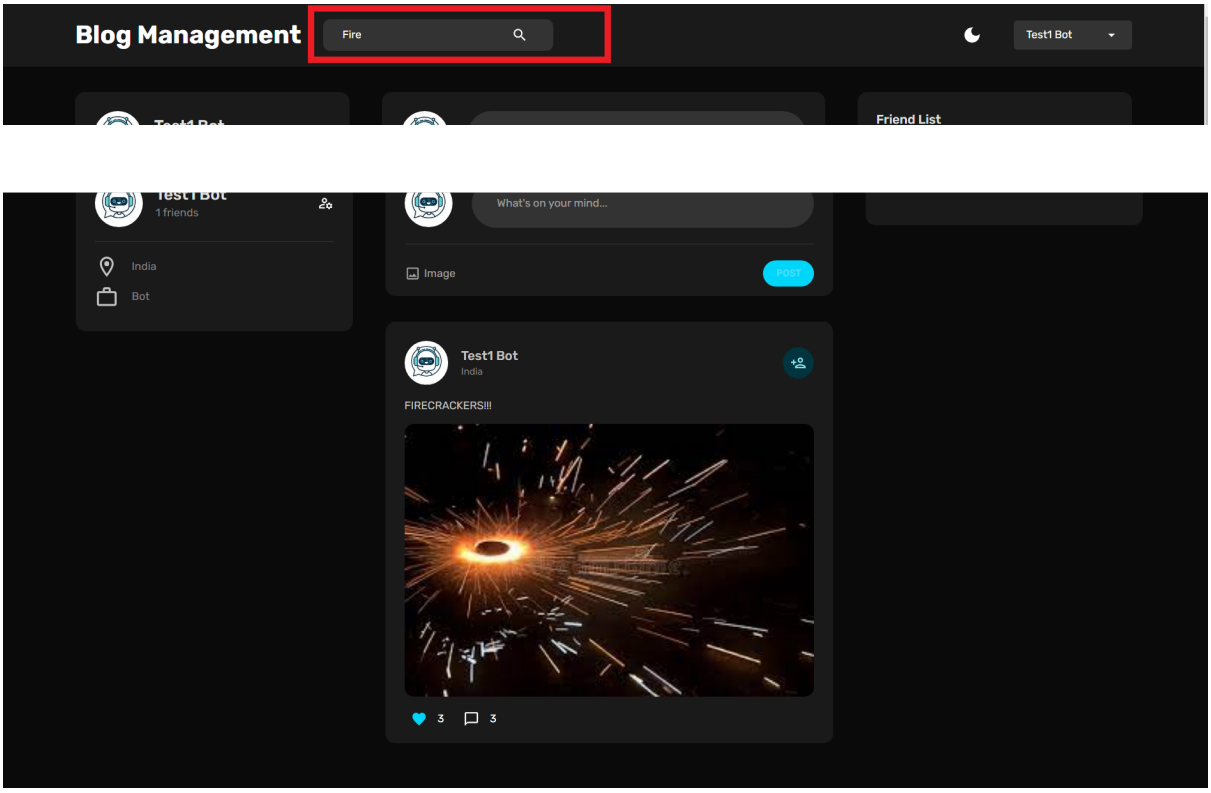


Add Comment :-

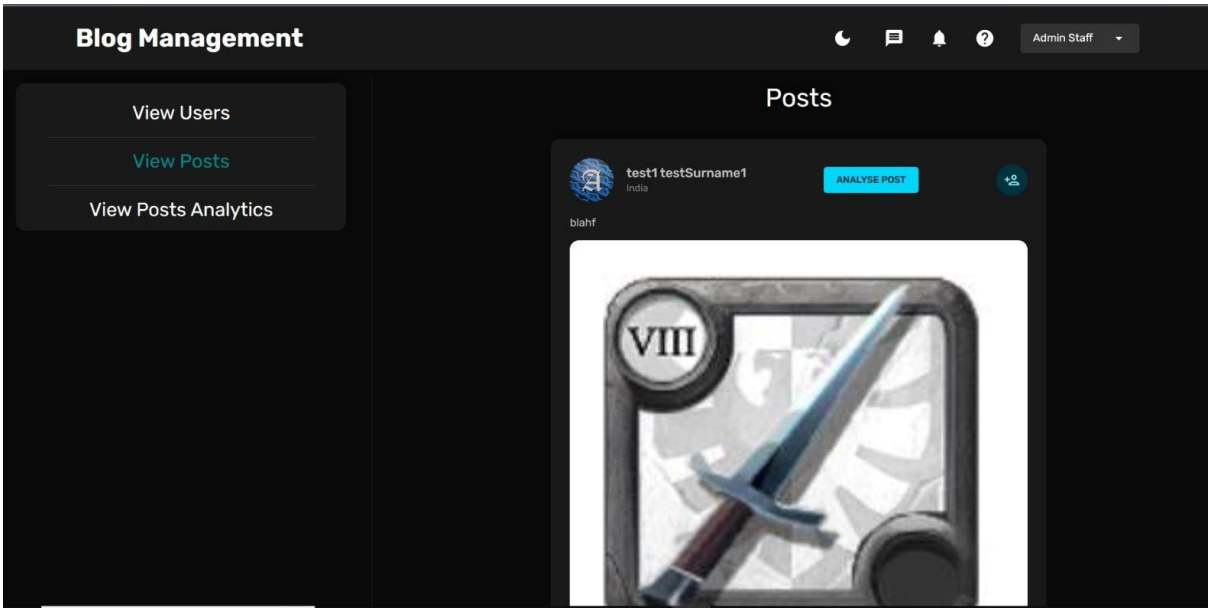



India
GOOD MORNING
1
1
Add a comment...
COMMENT
Test1 Bot
I like books

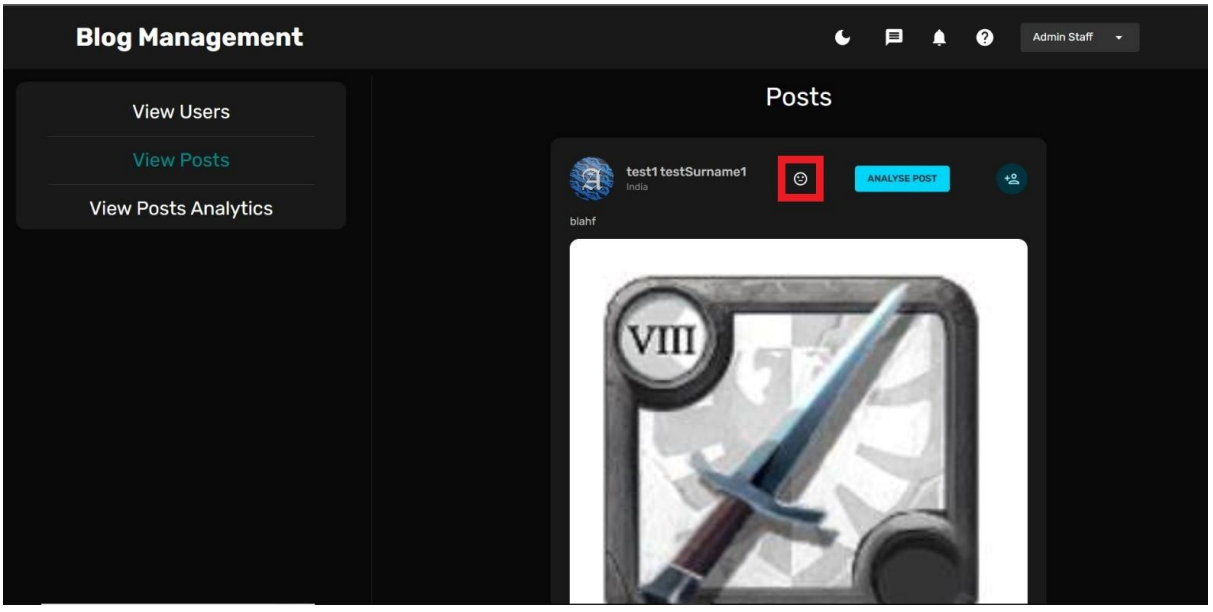
Search Blog :-


Blog Management
Fire
Test1 Bot
Friend List
1 friends
India
Bot
What's on your mind...
Image
POST
Test1 Bot
India
FIRECRACKERS!!!
3
3

View Sentiment Analysis :-



After Clicking on analyze Post Button :-



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 Extension :-

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 add report to 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