FULLSTACK USING NODE JS PROJECT REPORT

ON

MEMORIES SOCIAL MEDIA APPLICATION

Department of Computer Engineering & Application

Institute of Engineering & Technology

GLA University

Mathura- 281406, INDIA

Submitted By:

Shubham Sharma

Univ. Roll no.: - 181500696

Project Supervisor:

Mr. Pankaj Kapoor,

Senior Technical Trainer

Declaration

I the undersigned solemnly declare that the project report MEMORIES SOCIAL MEDIA APPLICATION is based on my own work carried out during the course of our study under the supervision of Mr. Pankaj Kapoor. I assert the statements made and conclusions drawn are an outcome of my work. I further certify that I. The work contained in the report is original and has been done by me under the general supervision of my supervisor. II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad. III. We have followed the guidelines provided by the university in writing the report. IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

Signature of Candidate: Shubham

Name of Candidate: Shubham Sharma

Roll. No. :181500696

Course: B. Tech. CSE

Year: 2020-21

Semester: VI

Acknowledgement

This project is an acknowledgement to the intensity drive and technical competence of many persons who have contributed to it. I express my heartiest gratitude and deepest thanks to, Project Instructor Mr. Pankaj Kapoor for his proper guidance, suggestions and helping me in completing the project and I am very grateful towards the support he showed throughout the whole project. I am highly grateful to my parents who have been the source of money and encouragement during the course of my work. I would also like to thank all those who directly or indirectly supported or helped me in completing my project in time. I would like to express my gratitude towards my parents and members of my college for their kind cooperation and encouragement which helped me in completion of this project. All of them have willingly helped me out with their abilities.

Table of Contents

1. Introduction to the project

1.1 Reason for selecting the topic: -

I chose the topic because I wanted to build something really dynamic system for posting memories for the users. The main aim of developing this platform is to create a platform, allowing multiple users to share their beautiful memories among other users. User can like the memories shared by other users. Only the user posting the memory will be allowed or authorized to delete the post posted by him and he can modify any of his posts at any given point of time. Also, the user can signup to the application using their google accounts. This project will be highly scalable and it could be easily be modified as per the user convenience.

1.2 Objectives of the project: -

Publishing content on the application is frequently seen as a decent method to connect with people with similar interest. This memories application is a kind of site that is included sections either made by the understudies or different individuals. A memory is an extraordinary path for an individual to figure out how to make their own site. In this we can create only individual memories on which we can post or share anything that we want which is publicly available. This will be done using the MERN stack which basically is an open-source JavaScript software for building web sites and web application. ***MERN*** is an acronym for MongoDB, ExpressJS, ReactJS and NodeJS. The main objective of the Memories Social Media Application is to manage the small print of Posts, Topic ,Idea, Content, Entries. The project is completely engineered at body finish and so solely the administrator is secure the access. the aim of the project is to create an application to cut back the manual work for managing the Posts, Topic, Views, Idea. It tracks all the small print regarding the thought, Content, Entries.

1.3 Feasibility Study: -

After doing the project Memories Social Media Application, study and analyzing all the existing or required functionalities of the system, the next task is to do the feasibility study for the project. All projects are feasible – given unlimited resources and infinite time.

A) Economical Feasibility

B) Technical Feasibility

C) Operational Feasibility

1.4 Future Scope: -

This web application will be allowing a user to post some of his/her best memories which will be publicly available for the users. Understudies can post their recommendations and input for the association. In the event that somebody can't communicate her/his view before individuals those likewise share the things through this contributing to a post framework.

It can be summarized that the future scope of the project circles around maintaining information regarding:

1) Implement the backup mechanism for taking backup.

2) Automatic detection of errors like grammar.

3) Automatic notification to followers after posting a new memory.

1.5. Modules in Memories Project: -

- **Posts Management Module:** Will be used for managing the Posts details.

- **Views Module:** Will be used for managing the details of views.
- **Content Module:** Will be used for managing the details of content.
- **Login Module:** Will be used for managing the login details.
- **Users Module:** Will be used for managing the users of the system.

1.6. Input Data and Validation of Project: -

- All the fields such as Posts, Topic, Views are validated and does not take invalid values.
- Each form in the application will not accept the blank value fields.
- Validations for user input will be done.
- Checking of the Coding standards to be maintained during coding.
- Testing the module with all the possible test data.

2. About Memories Social Media Application

The application is reduced as much as possible to avoid errors while entering the data. It also provides error messages while entering the invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it provides it is user-friendly. Memories Social Media Application, as described above, can lead to error free, reliable, secure and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on record keeping. The purpose of Memories Social Media Application is to manage all the posts, comments and all other basic information about the post. Making User able to see the posts posted by other users. The user can edit their own posts as per their requirements. The aim is to make adding, updating and removing the posts as efficient as possible as to improve the resource management of the posts data.

3. Methodology

This web application will be designed using HTML, CSS, JavaScript, ReactJS, mongo dB, NodeJS, and Express. The frontend part of the project will be done by HTML, CSS, JavaScript and the backend part will be done by NodeJS. All the data will be stored in mongoDB.

Mern Stack refers to a collection of JavaScript technologies used to develop web applications. Therefore, from the client to the server and from server to database, everything is based on JavaScript. ***MERN*** is a full-stack development toolkit used to develop a fast and robust web applications. MERN is a user-friendly stack which is the ideal solution for building dynamic websites and applications. This free and open-source stack offers a quick and organized method for creating rapid prototypes for web-based applications.

Components of MERN Stack

MERN is comprised of four different technologies:

- **M**ongoDB express is a schema less NoSQL database system
- **E**xpress JS is a framework used to build web applications in Node
- **R**eactJS is a JavaScript framework developed by Facebook
- **N**ode.js is a server-side JavaScript execution environment

The memories social media application has been developed to override the problems prevailing in the practicing manual system. This software is supported to eliminate and, in some cases, reduce the hardships faced by this existing system. Moreover, this system is designed for the particular need of the company to carry out operations in a smooth and effective manner.

The Application is reduced as much as possible to avoid errors while entering the data. It also provides error message while entering invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it proves it is user-friendly. Memories Social Media Application It can Assist the user to concentrate on their other activities rather to concentrate on the record keeping.

HTML: HTML, which stands for Hypertext Mark-Up Language, is the language for describing structured documents as well as the language used to create web pages in the Internet.

***CSS*:** Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

JavaScript: JavaScript is a scripting language and it is used on both client-side and server-side for making web pages interactive. It is a text-based language. Open and cross-platform.

MongoDB: MongoDB is a NoSQL document-oriented database; it is used for storing large amounts of data.

This platform provides an environment for posting or sharing memories throughout the organization. This gives detailed information about functional and non-functional requirements of users i.e. whatever they want to share then they can easily write on their post and it can be viewed by the member in the organization. The purpose meets the goal of sharing the information in formal or in secure way.

4. Introduction to Technologies of MERN Stack

- **M: -** M stands for MongoDB. Working with MongoDB NoSQL database is much easier than working with any relational database. There are no tables in MongoDB. All the data is stored in JSON format i.e. key-value pairs. In JSON, we define a unique key with a value associated with it. These key-value pairs are stored in a document, which in turn is stored in a collection. A collection in MongoDB can have any number of documents and such documents can have any number of key-value pairs. As I mentioned earlier, data in the MongoDB database is stored in BSON. BSON is nothing but extended JSON. It supports more data types than JSON. We store anything like, string, integer, boolean, double, binary data, object, array, javascript code and many more.

These documents are grouped inside a collection. A collection can be equivalent to a table in a relational SQL database. A collection always exists in a database and there is no pre-defined structure of a collection. In SQL, the database contains tables and in MongoDB, the database contains collections.

- **E: -** E stands for Express.js. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy. Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.
  Many popular frameworks are based on Express.

- **R: -** R stands for React. React is a JavaScript library and React applications built on it run in the browser, NOT on the server. Applications of this kind only communicate with the server when necessary, which makes them very fast compared to traditional websites that force the user to wait for the server to re-render entire pages and send them to the browser.
  React is used for building user interfaces - what the user sees on their screen and interacts with to use your web app. This interface is split up into components, instead of having one huge page you break it up into smaller pieces known as components. In more general terms, this approach is called Modularity.
  - It's declarative: React uses a declarative paradigm that makes it easier to reason about your application.
  - It's efficient: React computes the minimal set of changes necessary to keep your DOM up-to-date.
  - And it's flexible: React works with the libraries and frameworks that you already know.

- **N: -** N stands for Node.js. As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

5. Hardware and Software Requirements

1) For VS Code Editor: -

Hardware
• 1.6 GHz or faster processor
• 1 GB of RAM

Platforms
• OS X Yosemite (10.10+)
• Windows 7 (with .NET Framework 4.5.2), 8.0, 8.1 and 10 (32-bit and 64-bit)
• Linux (Debian): Ubuntu Desktop 16.04, Debian 9
• Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 8, Fedora 24 2)

For MongoDB: - Recommended Platforms
• Amazon Linux 2 • Debian 9 and 10
• RHEL / CentOS 6, 7, and 8 • SLES 12 and 15
• Ubuntu LTS 16.04, 18.04, and 20.04
• Windows Server 2016 and 2019

3) Technologies Used HTML, CSS, JAVASCRIPT, NodeJS, ReactJS, ExpressJS, MongoDB

4) Browser Opera, Chrome, Firefox

6. Test Technologies

For testing the web applications we will use "**burp suite"** to find and flaws in the website like XXS(cross site scripting), broken authentication, Security Misconfiguration and etc. With these testing we will make sure that the web application that we created is safe for the users and any errors will be fixed in real time. Apart from software we will make sure to make some test cases to check the website's feasibility individually. On the other hand, we will make sure to check OWASP to stay update with new bugs and injections.

7. What contribution would the project make and where?

We think that the project will be best suited for someone who likes to get exposer or want to expand their domain, personally or business wise. Because it will help them to send their target audience a message about themselves. This will help them grow and connect to more people. The field it will be effective the most would be for e-commerce for expanding business.

8. Problem Statement

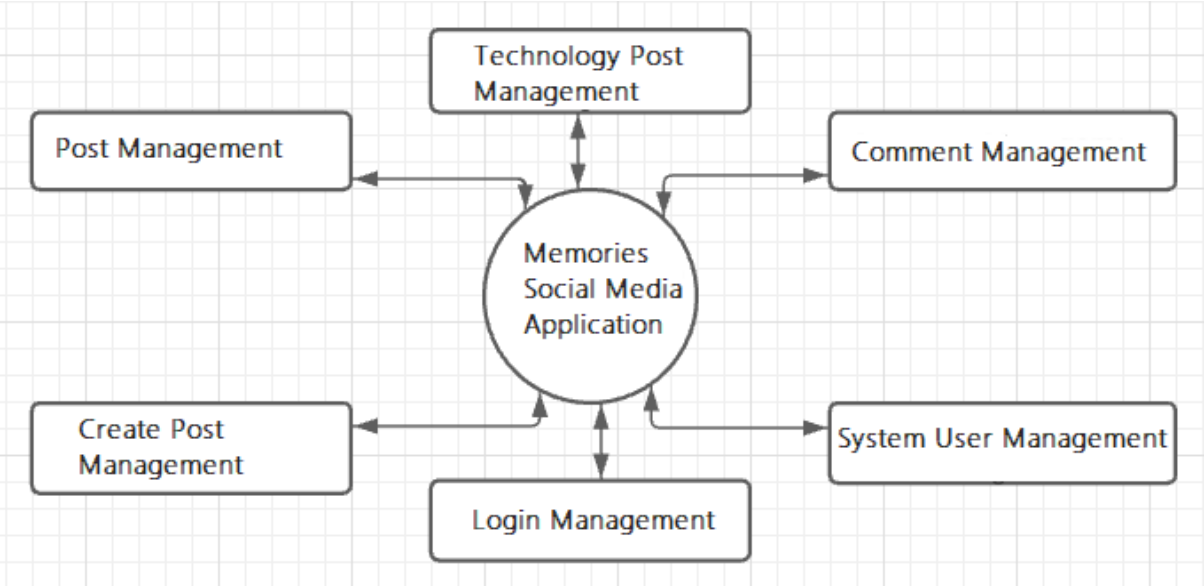
The problems we are going to solve through this project is the common problem that every user face. Through this we are trying to give exposer to the user and convey a message to the target audience that the user need. It will be a place where people can share their memories to the world.

9. Scope for Extension of the Project

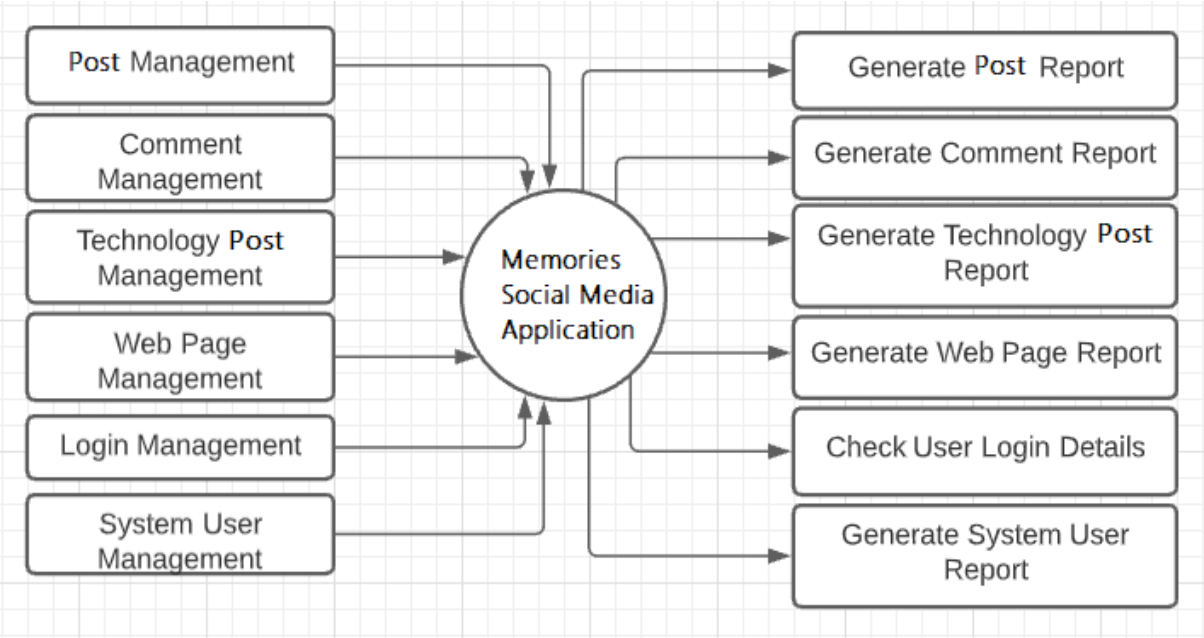
This project can further be extended to by adding some of the features which will make it easy for the user to find the most relevant content according to his or her interests by providing a certain set of different categories to the user. Further more the people who are following a person will get notified when that person will post some new content. If a user finds something interesting then he can save that post to favourites and at the same time they can leave a like for that post to which the author will get notified if somebody likes their post. Then the user will also be able to share the content to other users.

10. Software Designing

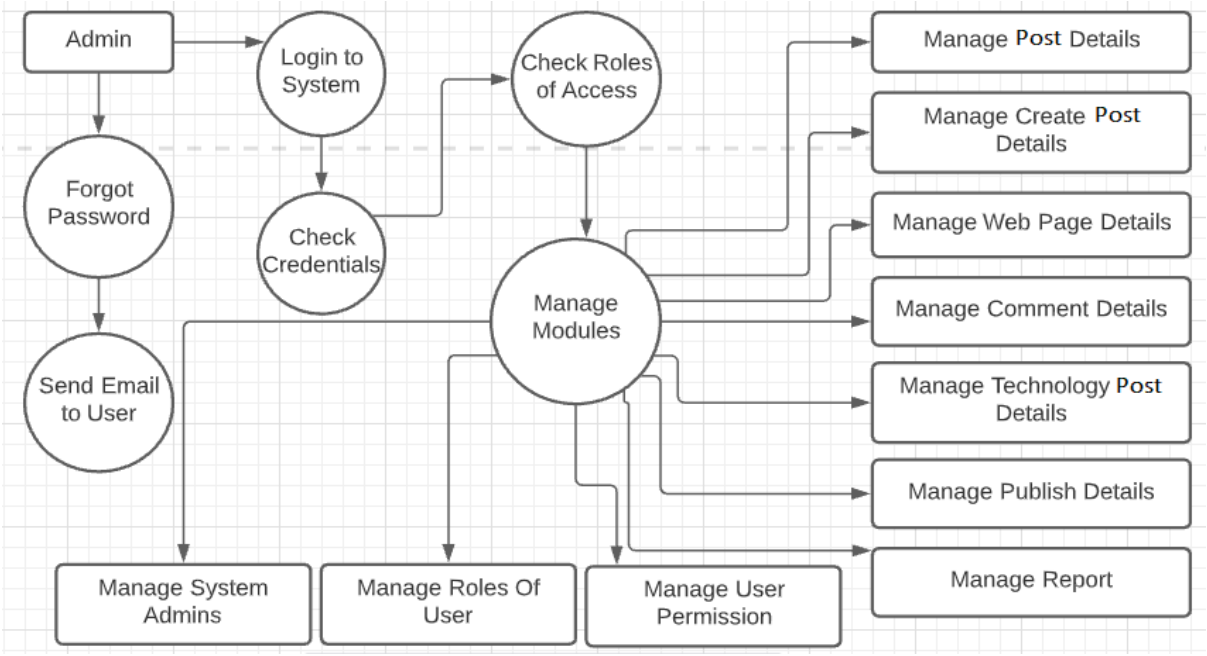
10.1. Zero Level DFD: -



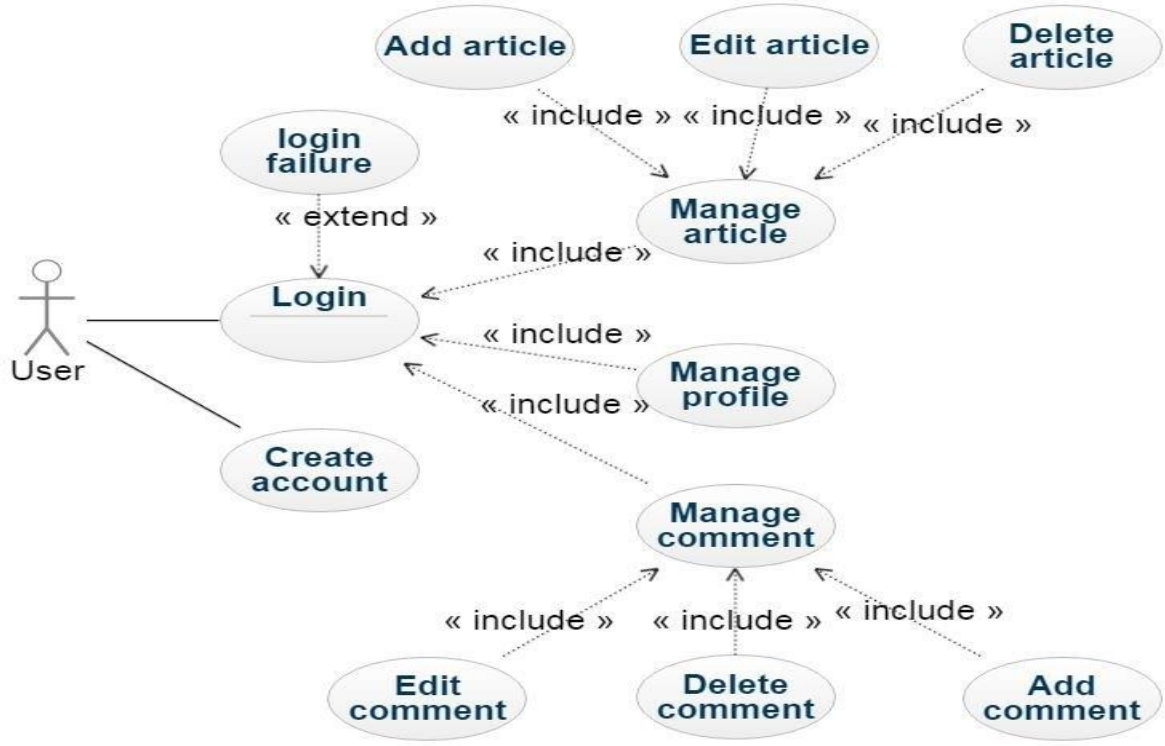
10.2. First Level DFD: -



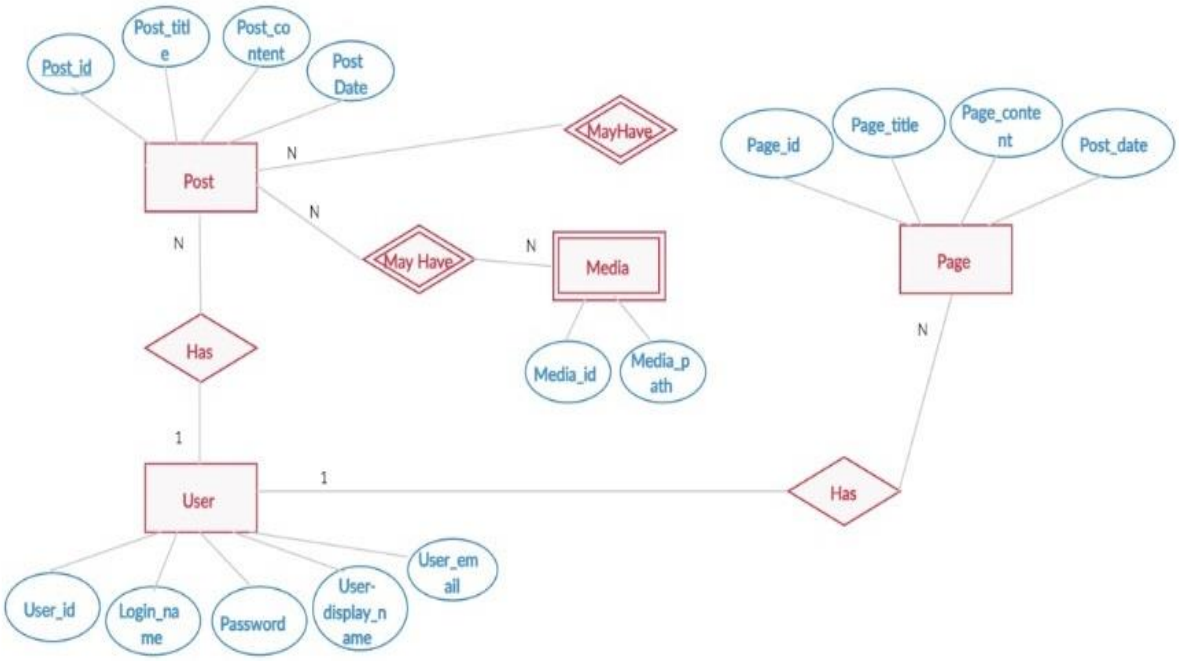
10.3. Second Level DFD: -



10.4. Usecase Diagram: -



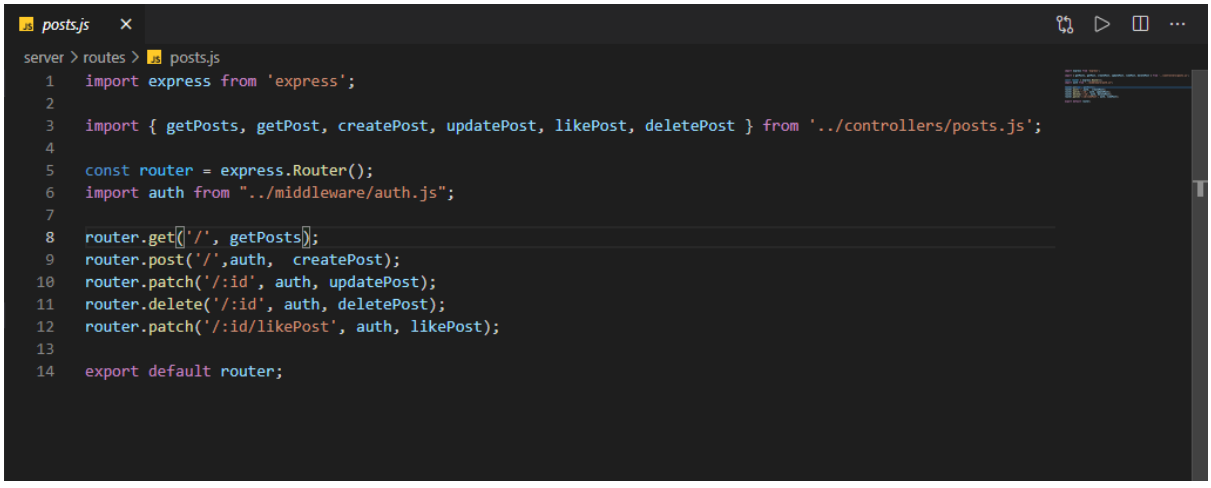
10.5. ER Diagram: -



11. Screenshots for Memories Social Media Application

11.1. Implementation Screenshots: -

11.1.1 Server Side: -



user.js

server > routes > user.js > ...

```
1  import express from "express";
2  const router = express.Router();
3
4  import { signin, signup } from "../controllers/user.js";
5
6  router.post("/signin", signin);
7  router.post("/signup", signup);
8
9  export default router;
```

user.js

server > models > user.js > ...

```
1  import mongoose from "mongoose";
2
3  const userSchema = mongoose.Schema({
4    name: { type: String, required:  true },
5    email: { type: String, required: true },
6    password: { type: String, required: true },
7    id: { type: String },
8  });
9
10 export default mongoose.model("User", userSchema);
```

postMessage.js

server > models > postMessage.js > default

```
import mongoose from 'mongoose';

const postSchema = mongoose.Schema({
    title: String,
    message: String,
    name: String,
    creator: String,
    tags: [String],
    selectedFile: String,
    likes: { type: [String], default: [] },
    createdAt: {
        type: Date,
        default: new Date(),
    },
})

var PostMessage = mongoose.model('PostMessage', postSchema);

export default PostMessage;
```

user.js

server > controllers > user.js > ...

```
export const signin = async (req, res) => {
  const { email, password } = req.body;

  try {
    const oldUser = await UserModal.findOne({ email });

    if (!oldUser) return res.status(404).json({ message: "User doesn't exist" });

    const isPasswordCorrect = await bcrypt.compare(password, oldUser.password);

    if (!isPasswordCorrect) return res.status(400).json({ message: "Invalid credentials" });

    const token = jwt.sign({ email: oldUser.email, id: oldUser._id }, secret, { expiresIn: "1h" });

    res.status(200).json({ result: oldUser, token });
  } catch (err) {
    res.status(500).json({ message: "Something went wrong" });
  }
};

export const signup = async (req, res) => {
  const { email, password, firstName, lastName } = req.body;

  try {
    const oldUser = await UserModal.findOne({ email });

    if (oldUser) return res.status(400).json({ message: "User already exists" });

    const hashedPassword = await bcrypt.hash(password, 12);

    const result = await UserModal.create({ email, password: hashedPassword, name: `${firstName} $
    {lastName}` });
```

posts.js

server > controllers > posts.js > ...

```
const router = express.Router();

export const getPosts = async (req, res) => {
    try {
        const postMessages = await PostMessage.find();

        res.status(200).json(postMessages);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
}

export const getPost = async (req, res) => {
    const { id } = req.params;

    try {
        const post = await PostMessage.findById(id);

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

auth.js

server > middleware > auth.js > ...

```
import jwt from "jsonwebtoken";

const secret = 'test';

const auth = async (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    const isCustomAuth = token.length < 500;

    let decodedData;

    if (token && isCustomAuth) {
      decodedData = jwt.verify(token, secret);

      req.userId = decodedData?.id;
    } else {
      decodedData = jwt.decode(token);

      req.userId = decodedData?.sub;
    }

    next();
  } catch (error) {
    console.log(error);
  }
};

export default auth;

```

11.1.2 Client Side: -

posts.js

client > src > reducers > posts.js > ...

```
import { FETCH_ALL, CREATE, UPDATE, DELETE, LIKE } from '../constants/actionTypes';

export default (posts = [], action) => {
  switch (action.type) {
    case FETCH_ALL:
      return action.payload;
    case LIKE:
      return posts.map((post) => (post._id === action.payload._id ? action.payload : post));
    case CREATE:
      return [...posts, action.payload];
    case UPDATE:
      return posts.map((post) => (post._id === action.payload._id ? action.payload : post));
    case DELETE:
      return posts.filter((post) => post._id !== action.payload);
    default:
      return posts;
  }
};

```

index.js

client > src > api > index.js > ...

```
import axios from "axios";

const API = axios.create({ baseURL: "http://localhost:5000" });

API.interceptors.request.use((req) => {
  if (localStorage.getItem("profile")) {
    req.headers.Authorization = `Bearer ${
      JSON.parse(localStorage.getItem("profile")).token
    }`;
  }

  return req;
});

export const fetchPosts = () => API.get("/posts");
export const createPost = (newPost) => API.post("/posts", newPost);
export const likePost = (id) => API.patch(`/posts/${id}/likePost`);
export const updatePost = (id, updatedPost) =>
  API.patch(`/posts/${id}`, updatedPost);
export const deletePost = (id) => API.delete(`/posts/${id}`);

export const signIn = (formData) => API.post("/user/signin", formData);
export const signUp = (formData) => API.post("/user/signup", formData);

```

auth.js

client > src > actions > auth.js > ...

```
import { AUTH } from '../constants/actionTypes';
import * as api from '../api/index.js';

export const signin = (formData, router) => async (dispatch) => {
  try {
    const { data } = await api.signIn(formData);

    dispatch({ type: AUTH, data });

    router.push('/');
  } catch (error) {
    console.log(error);
  }
};

export const signup = (formData, router) => async (dispatch) => {
  try {
    const { data } = await api.signUp(formData);

    dispatch({ type: AUTH, data });

    router.push('/');
  } catch (error) {
    console.log(error);
  }
};

```

Auth.js

client > src > components > Auth > Auth.js > ...

```
const initialState = { firstName: '', lastName: '', email: '', password: '', confirmPassword: '' };

const SignUp = () => {
  const [form, setForm] = useState(initialState);
  const [isSignup, setIsSignup] = useState(false);
  const dispatch = useDispatch();
  const history = useHistory();
  const classes = useStyles();

  const [showPassword, setShowPassword] = useState(false);
  const handleShowPassword = () => setShowPassword(!showPassword);

  const switchMode = () => {
    setForm(initialState);
    setIsSignup((prevIsSignup) => !prevIsSignup);
    setShowPassword(false);
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    if (isSignup) {
      dispatch(signup(form, history));
    } else {
      dispatch(signin(form, history));
    }
  };

  const googleSuccess = async (res) => {
    const result = res?.profileObj;
    const token = res?.tokenId;
```

Form.js

client > src > components > Form > Form.js > Form > clear

```
8
9   const Form = ({ currentId, setCurrentId }) => {
10    const [postData, setPostData] = useState({ title: '', message: '', tags: '', selectedFile: '' });
11    const post = useSelector((state) => (currentId ? state.posts.find((message) => message._id ===
      currentId) : null));
12    const dispatch = useDispatch();
13    const classes = useStyles();
14    const user = JSON.parse(localStorage.getItem('profile'));
15
16    useEffect(() => {
17      if (post) setPostData(post);
18    }, [post]);
19
20    const clear = () => {
21      setCurrentId(0);
22      setPostData({ title: '', message: '', tags: '', selectedFile: '' });
23    };
24
25    const handleSubmit = async (e) => {
26      e.preventDefault();
27
28      if (currentId === 0) {
29        dispatch(createPost({ ...postData, name: user?.result?.name }));
30        clear();
31      } else {
32        dispatch(updatePost(currentId, { ...postData, name: user?.result?.name }));
33        clear();
34      }
35    };
36
```

Home.js

client > src > components > Home > Home.js > ...

```
7   import Form from '../Form/Form';
8
9   const Home = () => {
10    const [currentId, setCurrentId] = useState(0);
11    const dispatch = useDispatch();
12
13    useEffect(() => {
14      dispatch(getPosts());
15    }, [currentId, dispatch]);
16
17    return (
18      <Grow in>
19        <Container>
20          <Grid container justify="space-between" alignItems="stretch" spacing={3}>
21            <Grid item xs={12} sm={7}>
22              <Posts setCurrentId={setCurrentId} />
23            </Grid>
24            <Grid item xs={12} sm={4}>
25              <Form currentId={currentId} setCurrentId={setCurrentId} />
26            </Grid>
27          </Grid>
28        </Container>
29      </Grow>
30    );
31  };
32
33  export default Home;
34
```

Navbar.js

client > src > components > Navbar > Navbar.js > ...

```
10
11  const Navbar = () => {
12    const [user, setUser] = useState(JSON.parse(localStorage.getItem("profile")));
13    const dispatch = useDispatch();
14    const location = useLocation();
15    const history = useHistory();
16    const classes = useStyles();
17
18    const logout = () => {
19      dispatch({ type: actionType.LOGOUT });
20
21      history.push("/auth");
22
23      setUser(null);
24    };
25
26    useEffect(() => {
27      const token = user?.token;
28
29      if (token) {
30        const decodedToken = decode(token);
31
32        if (decodedToken.exp * 1000 < new Date().getTime()) logout();
33      }
34
35      setUser(JSON.parse(localStorage.getItem("profile")));
36    }, [location]);
37
```

posts.js

client > src > actions > posts.js > ...

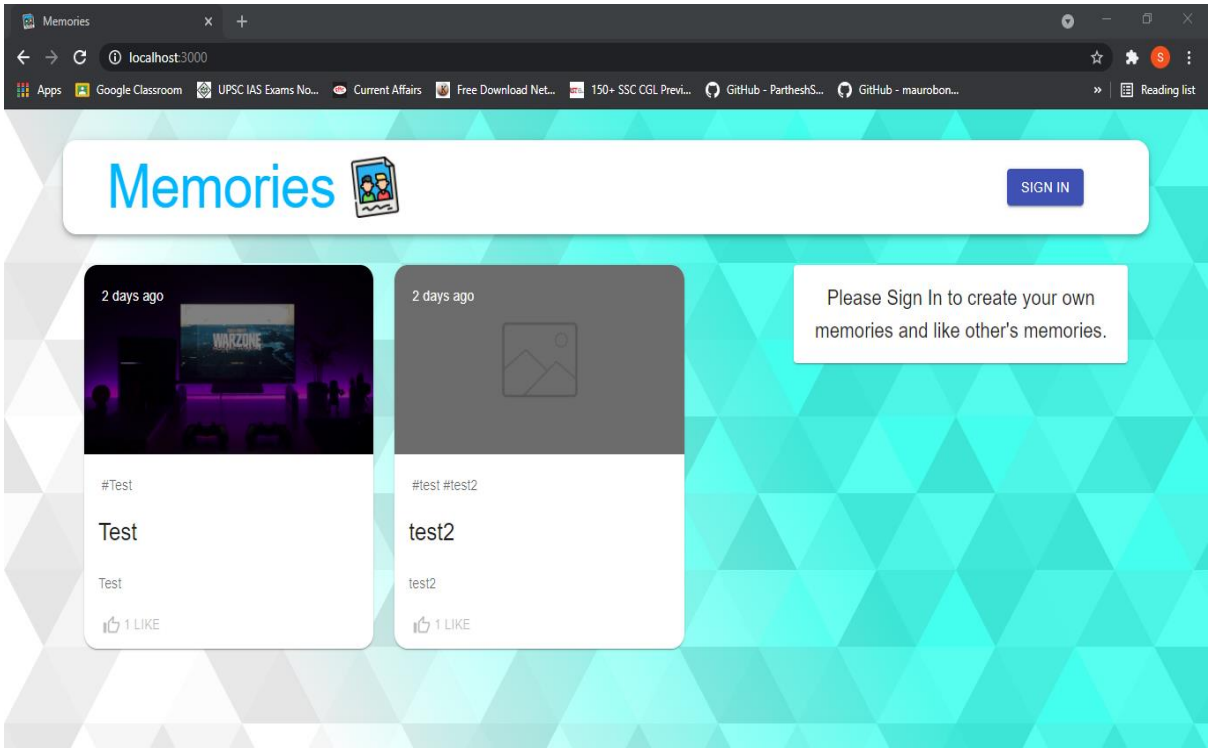
```
3
4   export const getPosts = () => async (dispatch) => {
5     try {
6       const { data } = await api.fetchPosts();
7
8       dispatch({ type: FETCH_ALL, payload: data });
9     } catch (error) {
10      console.log(error);
11    }
12  };
13
14  export const createPost = (post) => async (dispatch) => {
15    try {
16      const { data } = await api.createPost(post);
17
18      dispatch({ type: CREATE, payload: data });
19    } catch (error) {
20      console.log(error);
21    }
22  };
23
24  export const updatePost = (id, post) => async (dispatch) => {
25    try {
26      const { data } = await api.updatePost(id, post);
27
28      dispatch({ type: UPDATE, payload: data });
29    } catch (error) {
30      console.log(error);
31    }
32  };
```

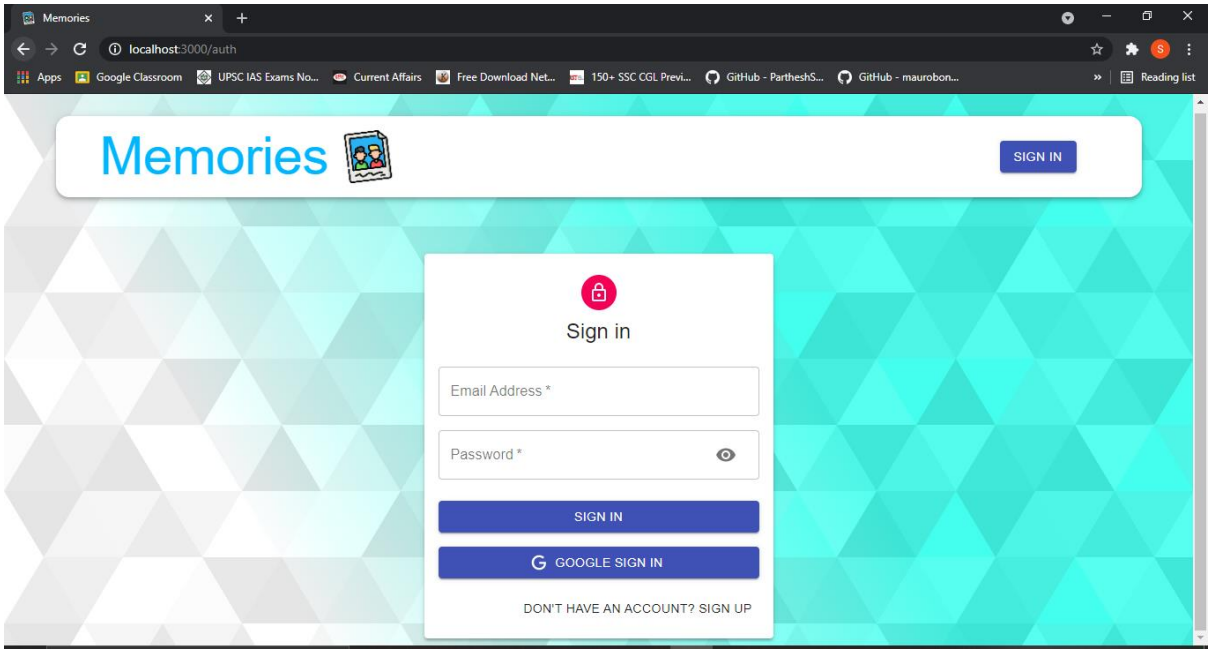
Posts.js

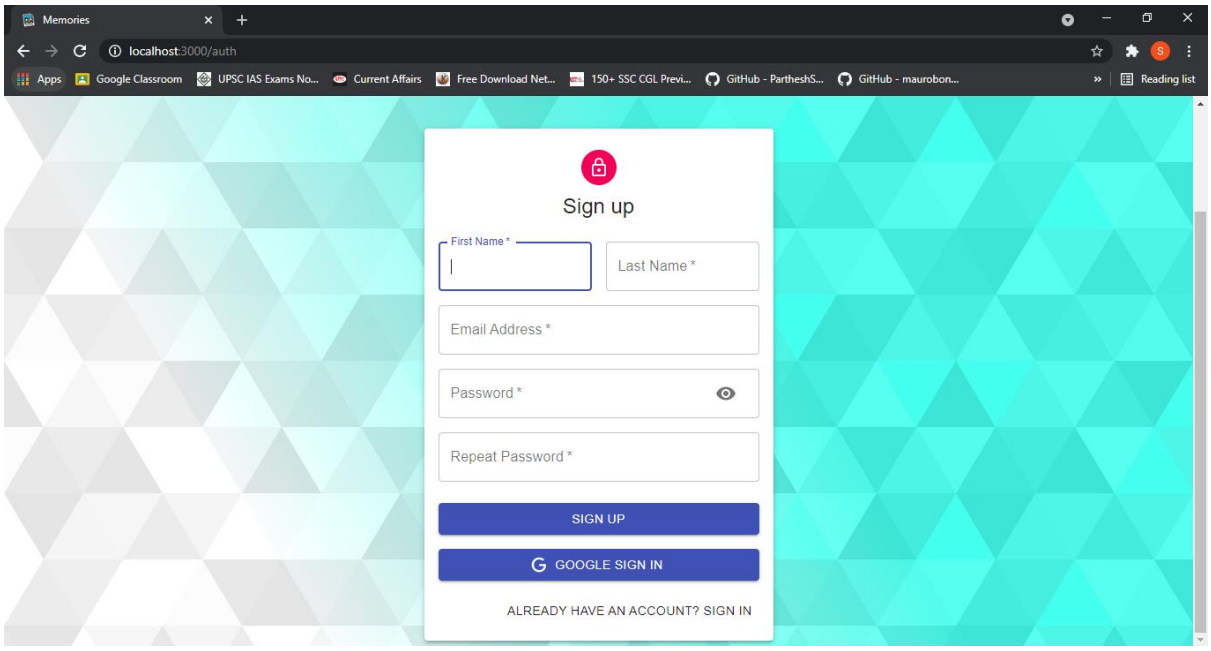
client > src > components > Posts > Posts.js > Posts

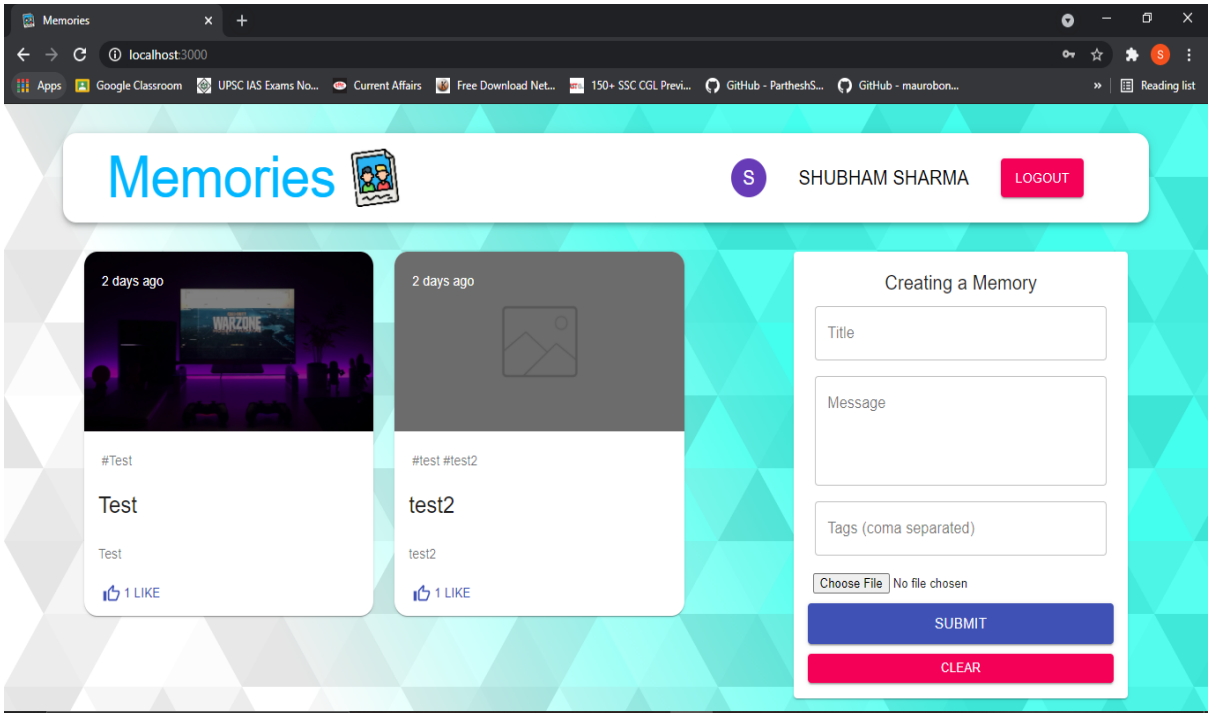
```
4
5   import Post from './Post/Post';
6   import useStyles from './styles';
7
8   const Posts = ({ setCurrentId }) => {
9     const posts = useSelector((state) => state.posts);
10    const classes = useStyles();
11
12    return (
13      !posts.length ? <CircularProgress /> : (
14        <Grid className={classes.container} container alignItems="stretch" spacing={3}>
15          {posts.map((post) => (
16            <Grid key={post._id} item xs={12} sm={6} md={6}>
17              <Post post={post} setCurrentId={setCurrentId} />
18            </Grid>
19          ))}
20        </Grid>
21      )
22    );
23  };
24
25  export default Posts;
26
```

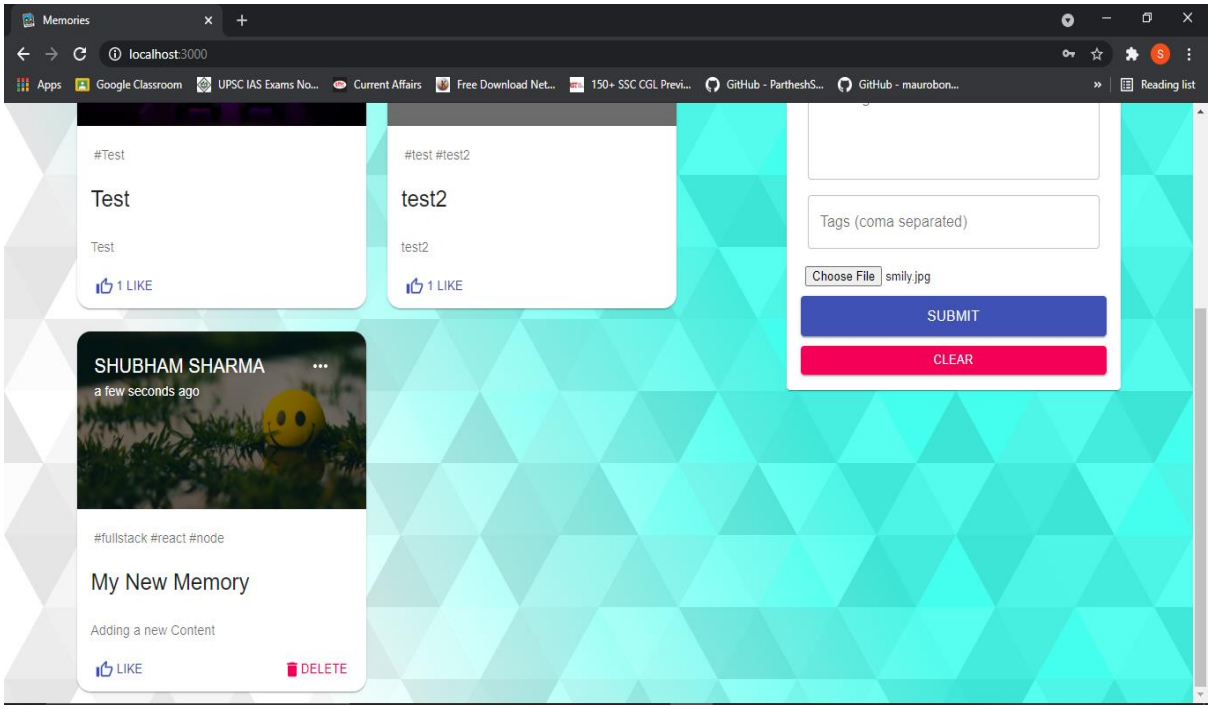
11.2. UI: -




Memories
localhost:3000/auth
Apps
Google Classroom
UPSC IAS Exams No...
Current Affairs
Free Download Net...
150+ SSC CGL Previ...
GitHub - PartheshS...
GitHub - maurobon...
Reading list
Memories
SIGN IN
Sign in
Email Address *
Password *
SIGN IN
GOOGLE SIGN IN
DON'T HAVE AN ACCOUNT? SIGN UP


Memories
localhost:3000/auth
Apps
Google Classroom
UPSC IAS Exams No...
Current Affairs
Free Download Net...
150+ SSC CGL Previ...
GitHub - PartheshS...
GitHub - maurobon...
Reading list
Sign up
First Name *
Last Name *
Email Address *
Password *
Repeat Password *
SIGN UP
GOOGLE SIGN IN
ALREADY HAVE AN ACCOUNT? SIGN IN


Memories
localhost:3000
Apps
Google Classroom
UPSC IAS Exams No...
Current Affairs
Free Download Net...
150+ SSC CGL Previ...
GitHub - PartheshS...
GitHub - maurobon...
Reading list
Memories
S
SHUBHAM SHARMA
LOGOUT
2 days ago
#Test
Test
Test
1 LIKE
2 days ago
#test #test2
test2
test2
1 LIKE
Creating a Memory
Title
Message
Tags (coma separated)
Choose File
No file chosen
SUBMIT
CLEAR


Memories
localhost:3000
Apps
Google Classroom
UPSC IAS Exams No...
Current Affairs
Free Download Net...
150+ SSC CGL Previ...
GitHub - PartheshS...
GitHub - maurobon...
Reading list
#Test
Test
Test
1 LIKE
#test #test2
test2
test2
1 LIKE
Tags (coma separated)
Choose File
smily.jpg
SUBMIT
CLEAR
SHUBHAM SHARMA
a few seconds ago
#fullstack #react #node
My New Memory
Adding a new Content
LIKE
DELETE

11.4 Existing system: -

In the existing system, the test can only be done manually. In the proposed system, we must use this application to computerize the test.

• Insufficient data security.

• More labour.

• It takes a long time.

• Engaged in a lot of educational work.

• You need to calculate manually.

• Senior officials have no direct role.

11.5 Proposed system of Memories Social Media Application: -

The proposed system allows you to overcome all the limitations of the existing system. The system provides sufficient safety and reduces manual work.

• Data Security.

• Make sure the information is correct.

• Minimize manual data entry.

• The shortest time for different processes.

• Improve efficiency.

11.6 Features of application: -

Features of the mission Memories Social Media Application:

• Product and Component based

• Creating & Changing Issues at ease

• Query Issue List to any depth

• User Accounts to manipulate the get admission to and preserve security

• Simple Status & Resolutions

• Multi-stage Priorities & Severities.

• Targets & Milestones for directing the programmers

• Attachments & Additional Comments for extra information

• Robust database back-end

• Various stage of stories to be had with quite a few clear out out criterias

• Accuracy in work.

• Easy & speedy retrieval of information.

12. Conclusion

This project is only a major goal is to venture to satisfy the need to manage their project work. With the help of ***MERN*** stack that I have used to make it possible for the user. This will prove to be a powerful web application satisfying all the requirements of the users. The objective of software is to show that the framework is capable of making the user satisfying their needs, fast and reliable.

Few concluding points

- A description of the background and context of the project and its relation to work already done in the area.
- Made statement of the aims and objectives of the project.
- The description of Purpose, Scope, and applicability.
- We define the problem on which we are working in the project
- We Describe the requirement specification of the system and the actions that can be done on these things
- We understand the problem domain and produce a model of the system which describes operations that can be performed on the system.
- We included features and operation in detail, including screen layout.
- We designed user interface and security issue to system

13. References: -

• Beta-labs: - https://www.beta-labs.in/p/reactjs.html

• NPM Documentation: https://www.npmjs.com/

• GeeksForGeeks: - https://www.geeksforgeeks.org/mern-stack/

• Faculty Guidelines: Mr. Pankaj Kapoor

- Getting MERN with Mongo, Express, React and Node
- Pro MERN Stack Development