



Dharmsinh Desai University, Nadiad

Faculty of Technology

Department of Computer Engineering

B. Tech. CE Semester – V

Subject : (CE-515) Advanced Technologies

Project Title : Social Media Website (SnapBook)

By :-

- 1] Dhruval Shah, Roll No : CE-137, Id: 19CEUON071
- 2] Vrundan Shah, Roll No : CE-140, Id : 19CEUOS141
- 3] Jainil Trivedi , Roll No : CE-163, Id : 19CEUON061
- 4] Harpritsinh Yadav , Roll No : CE-176, Id : 19CEUOS129

Guided By :-

- 1] Prof. Prashant M Jadhav
- 2] Prof. Siddharth P Shah



Certificate

This is to certify that the project entitled as “Social Media Website” is a bonafide report of the work carried out by :

- 1) Mr. Dhruval Shah, Student ID No: 19CEUON071
- 2) Mr. Vrundan Shah, Student ID No: 19CEUOS141
- 3) Mr. Jainil Trivedi , Student ID No : 19CEUON061
- 4) Mr. Harpritsinh Yadav , Student ID No : 19CEUOS129

of Department of Computer Engineering, semester V, under the guidance and supervision of Prof. Prashant M Jadhav and Prof. Siddharth P Shah for the subject Advanced Technologies during the academic year 2021-2022.

Project Guide :

Prof. Prashant M Jadhav,

Prof. Siddharth P Shah

Head of the department :

Dr. C.K Bhensdadia

Department of Computer Engineering,

Faculty Of Technology,

Dharmsinh Desai University,

Nadiad

Contents :

1.Front Page.....	1
2. Certificate	2
3. Contents	3
4.Abstract	4
5.Introduction	5
5.1 Technologies/Tools/Frameworks	6
6.Software Requirement specification	7
7.Design	
I) Data Flow Diagram	13
II) Entity Relationship.....	18
III)Data Dictionary	19
8.Implementation Details	
I) Modules(Brief Description)	22
II)Major functionality	23
III)Algorithm/Flowchart	34
9.Testing	35
10.Screenshots	38
11.Conclusion	42
12.Limitations and Future Extensions	43
13.Bibliography	44

4. Abstract

Social media application refers to online platform that is designed to allow people to share their life via photos to their friends. Social media acts a break from the real life for most of the people.

Anyone with internet access can sign up for a social media account. They can use that account to share whatever content they choose to, and the content they share reaches anyone who visits their page or profile.

People of this new era can be miles away from their friends and family but can be still be in touch with them via chat and calling facility that socail media provides us.

Many people access social media through smartphone apps, this communication tool started with computers, and social media can refer to any internet communication tool that allows users to broadly share content and engage with the public with absolute ease.

5. Introduction

Social media application refers to online platform that is designed to allow people to share their photos to their friends

To customize user experience we have implemented number of features:-

Features:-

- 1] First and foremost it's a completely dynamic website.
- 2] Signup/login.
- 3] User has to verify his/her Email before logging-in.
- 4] User can view as well as edit his/her profile.
- 5] User can post photo with caption.
- 6] User can reply on his/her post's comments and can comment on friend's post as well.
- 7] User can search and view other user's profile.
- 8] User can add/remove friend.
- 9] User can delete his/her posts and comments.
- 10] User can like a post as well as save posts for his/her future reference.
- 11] User can see all his/her online friends and can chat with them also they can chat with the friends he/she has previously messaged.
- 12] Logout

5.1 Technologies/languages/Frameworks used:-

1. Javascript
2. Html
3. Css
4. React
5. Node,Express
6. Socket
7. MongoDB

Libraries :-

1. Material-UI
2. Multer
3. dotenv
4. Axios
5. Nodemailer

Tools :-

1. Visual Studio Code
2. Github/git
3. Atlas-Mongodb server
4. Postman

Platform :-

local server

6. Software Requirement Specification

End User :-

- User

Functional Requirements :-

R1] Registration

R.1.1] Sign up a User :

- A user that has no existing account can sign in to our website and will be added to the database.

Input:

- Name, Email, Address, Phone no, address, username, password

Output:

- User is redirected to the login page and the user info is added to the database.

R.1.2] Login by user :

- Existing user can login with his/her credentials.

Input:

- Username, Password

Output:

- If the credentials are correct user is redirected to the home page else the error message is displayed.

R2] Manage Post

R.2.1] Add Post :

- To share an image user can upload the image from the saved photos he/she would like to post.

Input:

- Upload a photo, write a caption(Optional)

Output:

- The post will be added in the database.

R.2.2] View Post :

- The user can view all the post of the his/her friends on the home page as well as he/she can view a post of a user.

Input:

- Adding friend , clicking on the post to view

Output:

- All the post of the friends are shown in the home page and a single post can be viewed by clicking on the post.

R.2.3] Like Post :

- User can like a post.

Input:

- Click on the like button

Output:

- The color of the like button would toggle on like and unlike post and database would be updated

R.2.4] Saved Post :

- User can save the post for the further reference.

Input:

- Click on the save button

Output:

- The color of the save button would toggle on save and unsave post and the database would be updated.

R.2.5] Delete Post :

- User can delete the post if he/she likes.

Input:

- Click on the delete button in on the post

Output:

- Redirected in the user profile on successful delete or error message is displayed.

R3] Search and View Profile

R.3.1] Search Profile:

- User can search a profile using some keyword.

Input:

- Enter the keyword to be searched

Output:

- Shows all the user profiles matching the entered keyword.

R.3.2] View Profile:

- User can view a profile from the search page, from the home page.

Input:

- Click on the profile that has to be viewed

Output

- The profile of the user clicked

R4] Manage Comment

R.4.1] Add Comment:

- User can add a comment on the post selected.

Input:

- Write Comment in the text box and click on add comment.

Output:

- Comment added would be displayed.

R.4.2] View Comment:

- User can view all comments of the post selected.

Input:

- Click on the post of which the comments are to be viewed.

Output:

- Selected post would be shown with all it's comments

R.4.3] Delete Comment:

- User can delete his/her own comment.

Input:

- Click on the delete button shown

Output:

- Comment would be deleted

R5] Manage Friend

R.5.1] Add Friend:

- User can add Friend by visiting on the friends profile.

Input:

- Click on the add friend icon

Output:

- Icon would toggle to remove friend

R.5.2] Remove Friend:

- A user can remove existing friend by visiting his/her profile.

Input:

- Click on the remove friend icon

Output:

- Icon would toggle to add friend

R6] Chat Friend

R.6.1] View Conversation:

- A user can view his/her existing conversation.

Input:

- Click on the chat option in the navbar

Output:

- Chat page would open and conversation would be available on the left.

R.6.2] Messages of an Conversation:

- A user can view his/her messages with one of the friend.

Input:

- Click on the user profile shown in the conversation column

Output:

- User would be able to view all the existing and new messages with their sent or received time.

R.6.3] View Online Friends:

- A user can view his/her friends online .

Input:

- Click on the chat option in the navbar

Output:

- Chat page would open and profile of the friends online would be available on the right.

R.6.4] Message friends :

- A user can message his/her friends.

Input:

- Type the message in the textbox given and click on the post to send message.

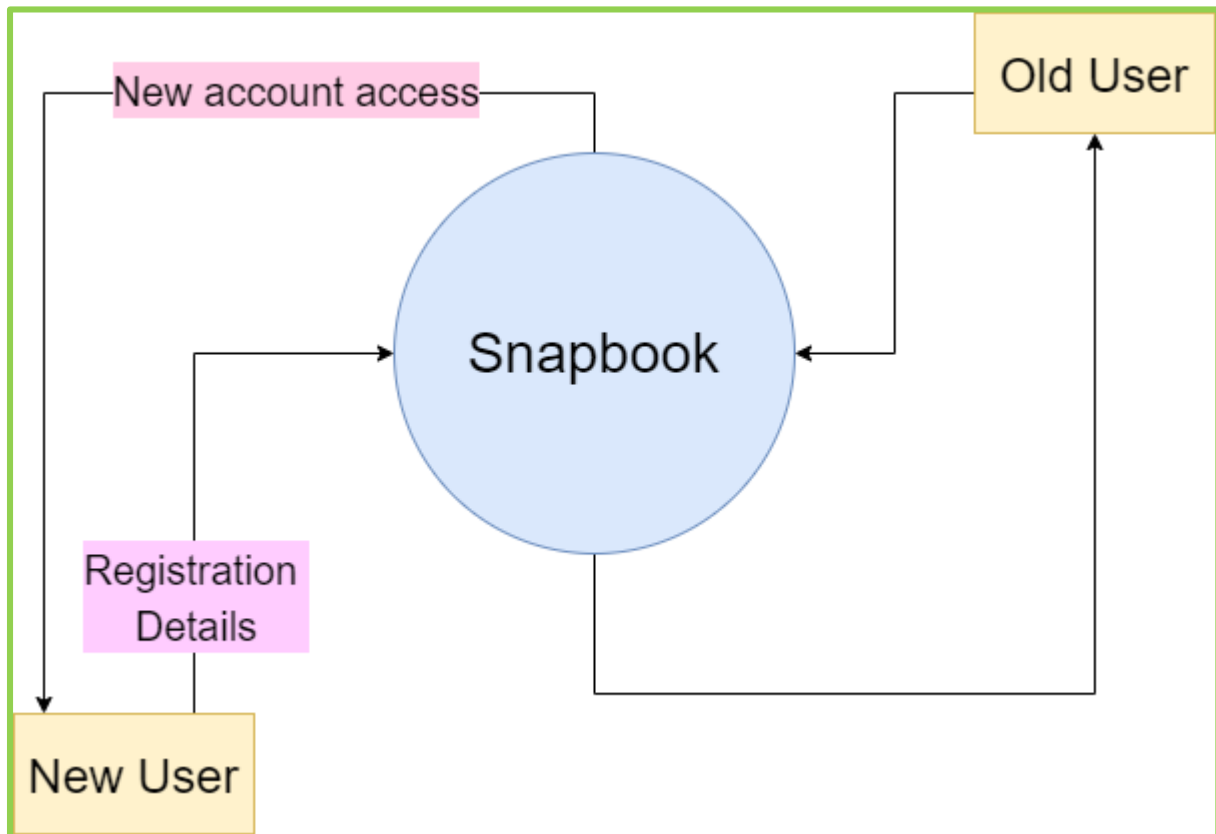
Output:

- Message would be send to user(friend) selected and would be shown in chatbox.

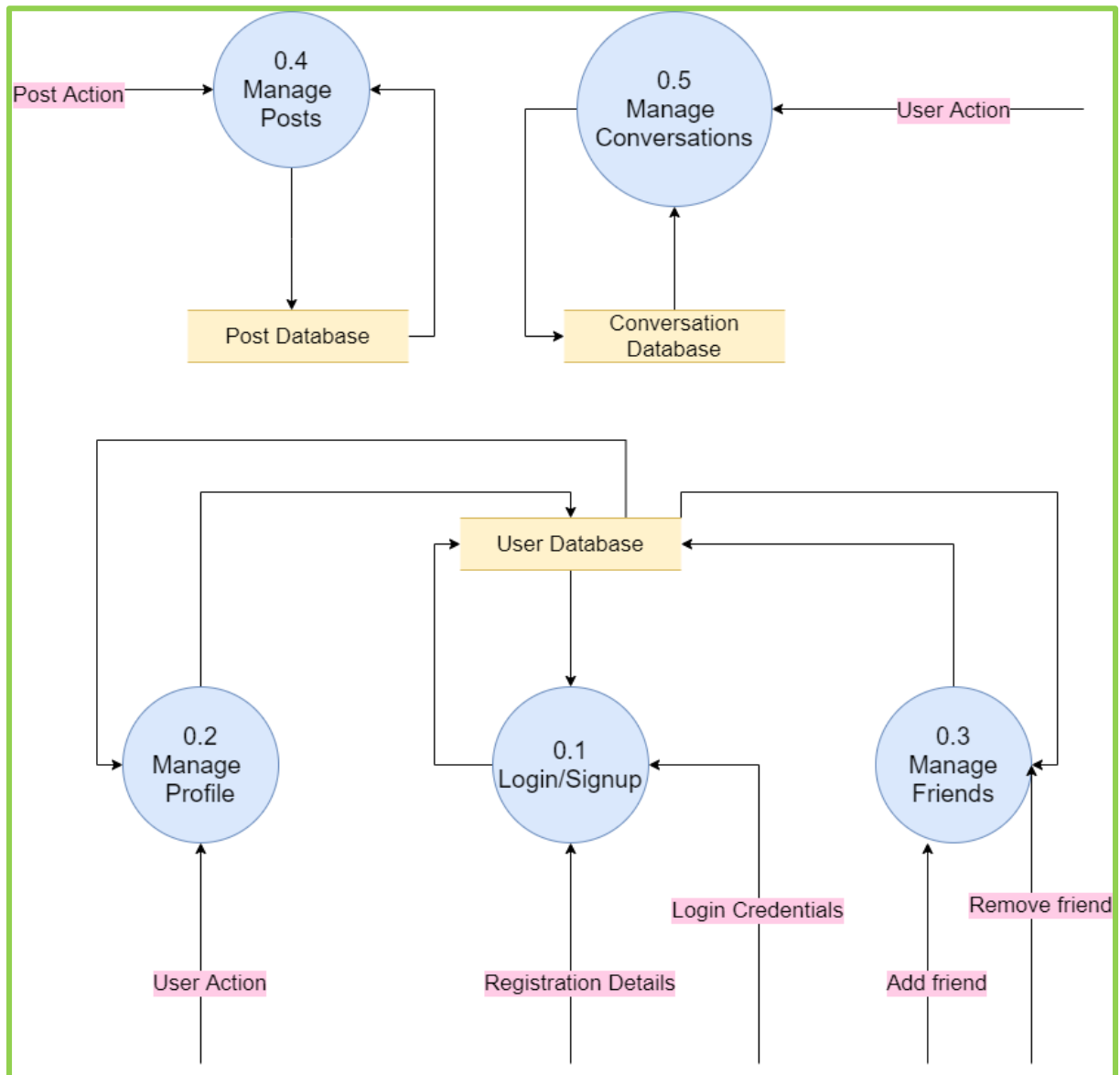
7. Design

I . Data Flow Diagram (DFD)

Level 0 :

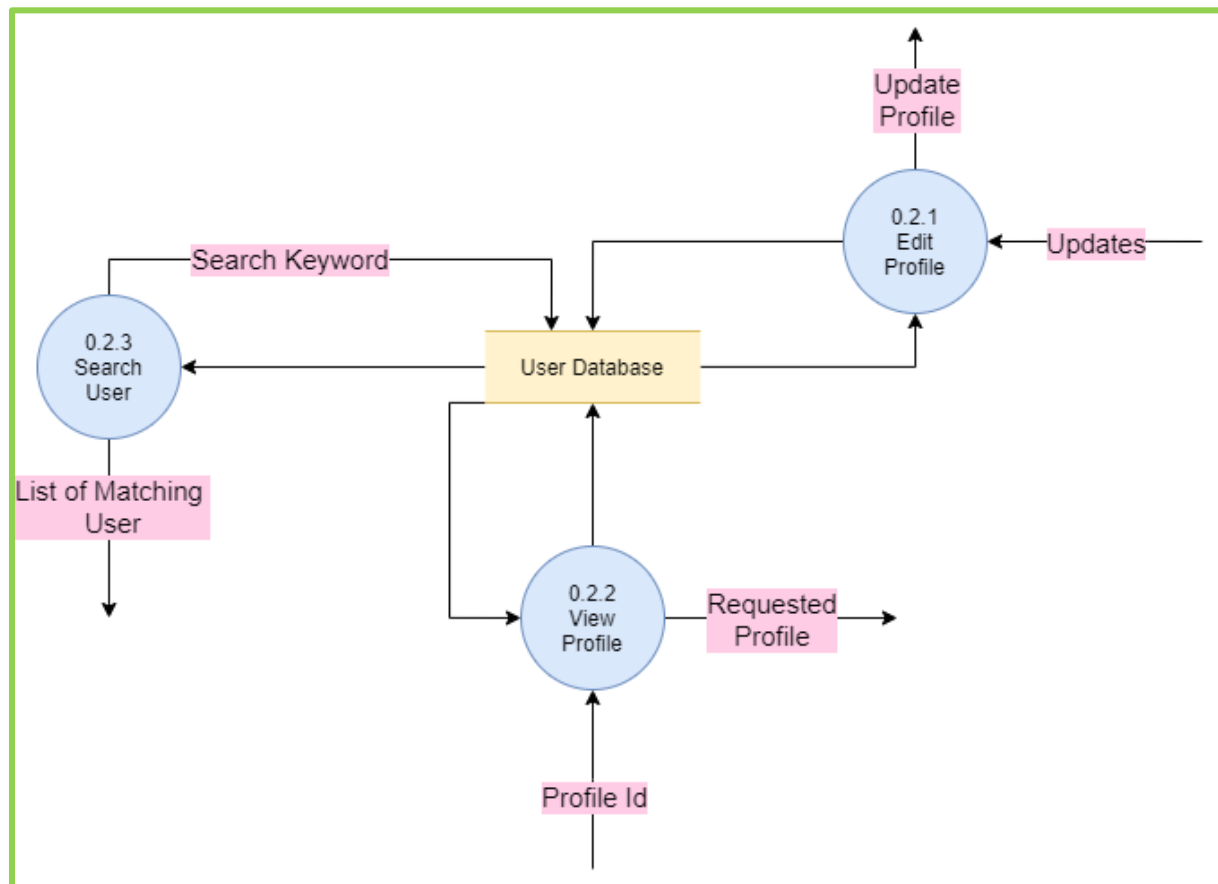


Level 1 :



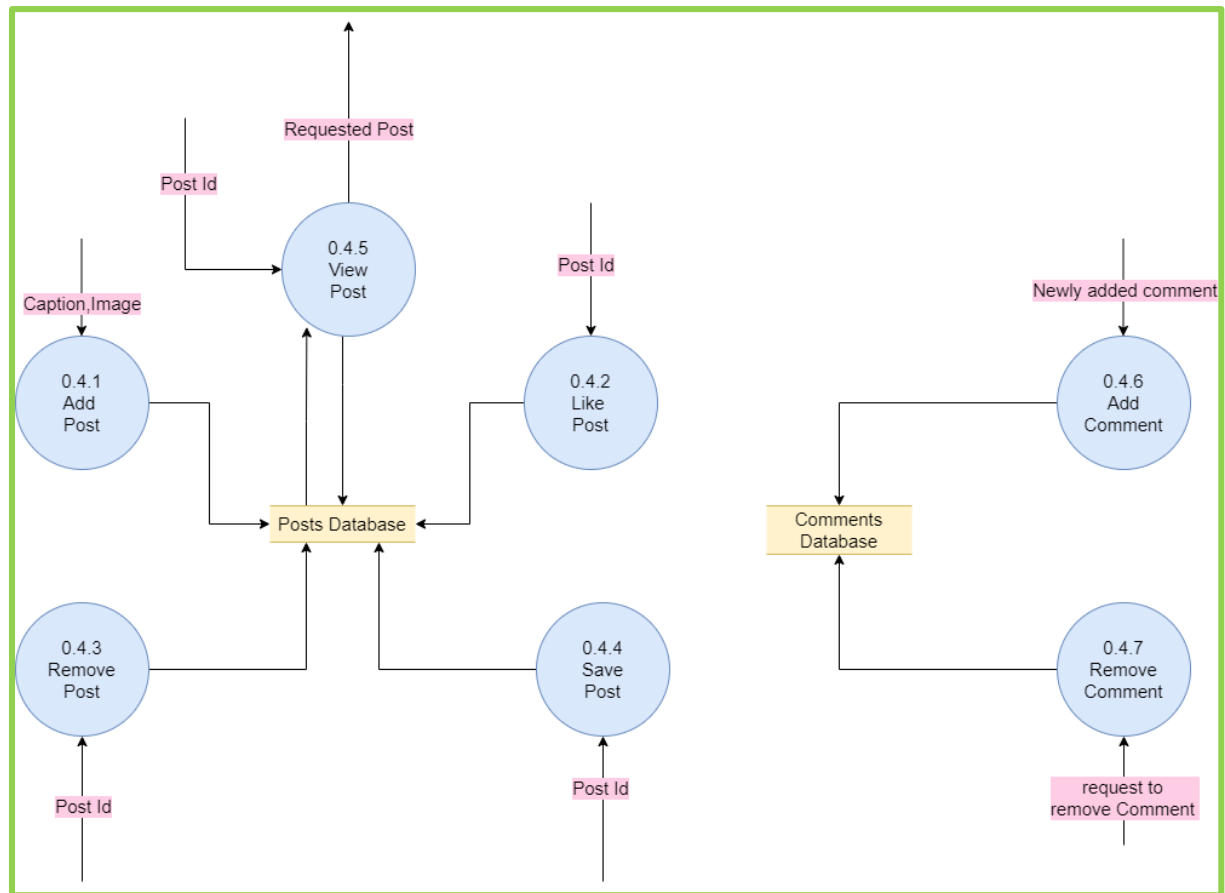
Level - 2

Manage Profile



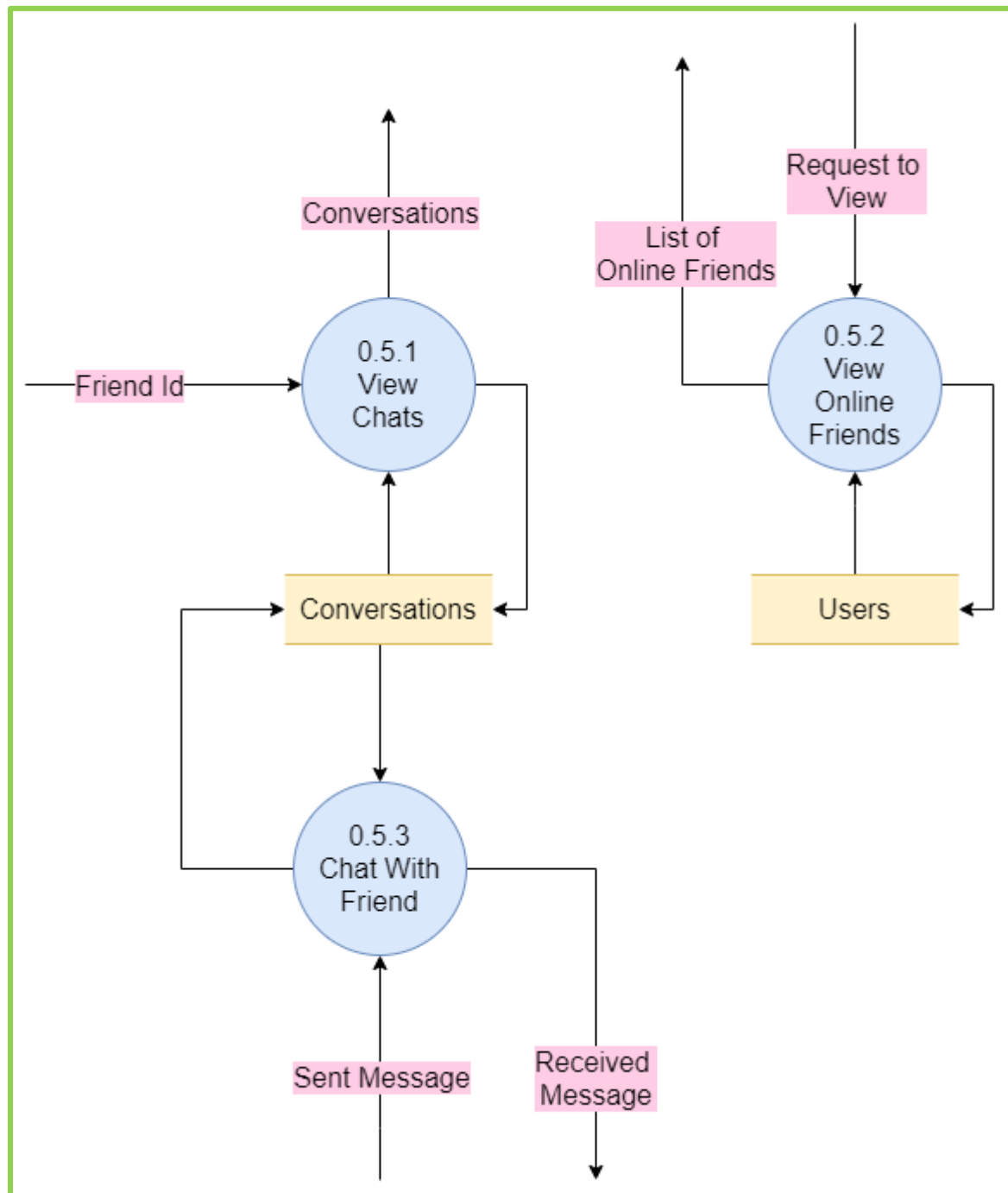
Level - 2

Manage Post



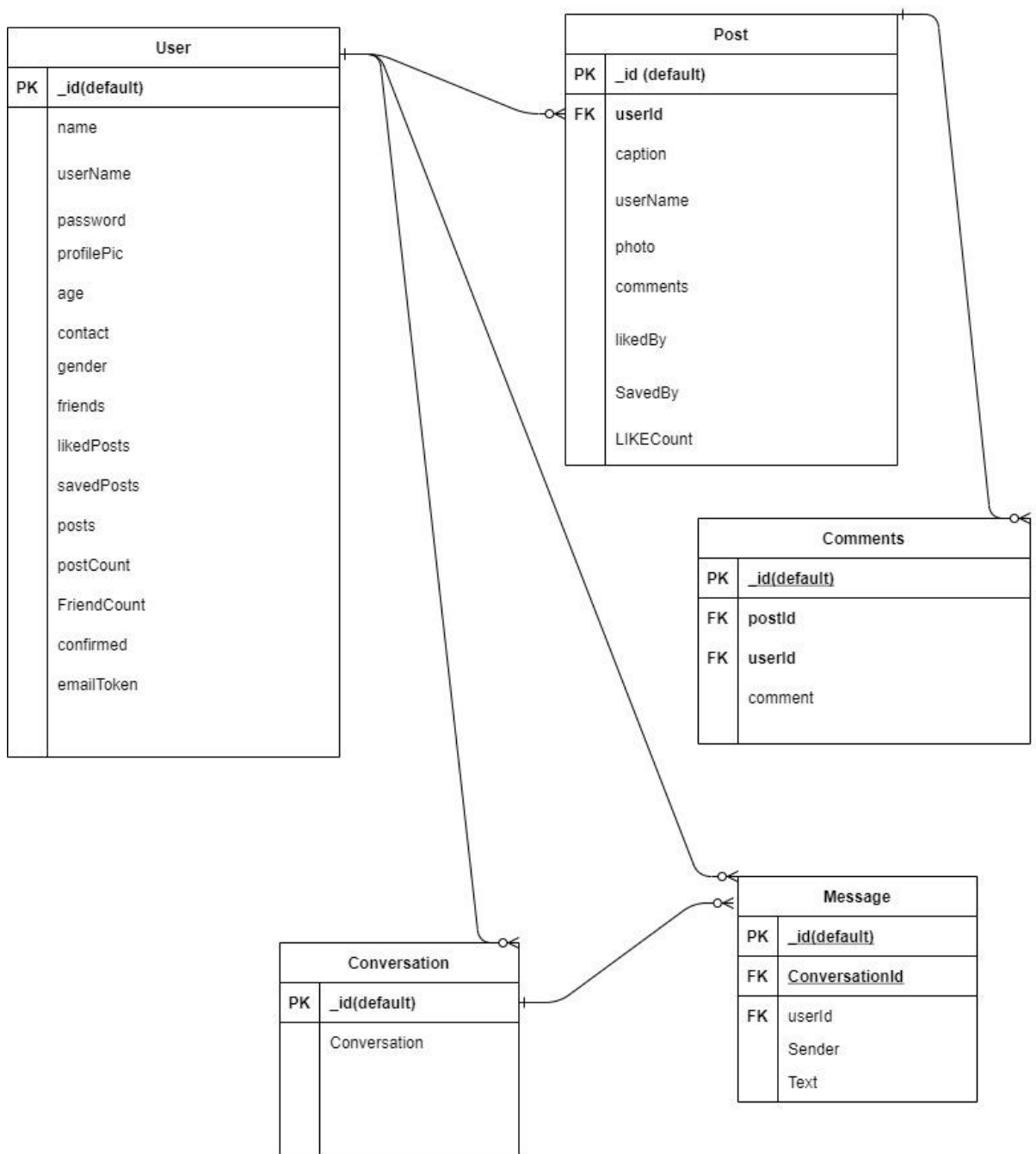
Level - 2

Manage Conversations



II. Entity Relationship

ER-Diagram



III. Data Dictionary

USER	
Attribute	DataType
userId	int
name	String
email	String
userName	String
password	String
Profilepic (path)	String
Age	Int
Contact	String
Gender	String
Friends	Array(int)
Likedposts	Array(int)
Savedposts	Array(int)
Posts	Array(int)
postCount	Int
friendCount	Int
Confirmed	Boolean
emailToken	String

POST

Attribute	DataType
PostId	Int
Photo	String
Caption	String
userName	String
userId	Int
Comments	Array(int)
likedBy	Array(int)
SavedBy	Array(int)
likeCount	Int

COMMENT

Attribute	DataType
CommentId	Int
Comment	String
userId	Int
postId	Int

CONVERSATION

Attribute	DataType
ConversationId	Int
Members	Array(ReceiverId,SenderId)

MESSAGE

Attribute	DataType
MessageId	Int
Text	String
Sender	String

8. Implementation Details

I. Modules

1. User module:

User module saves all the information during the registration of the user if the information is valid. During login credentials of the user would be verified if valid would be logged in successfully. User module would also be used to save user friend IDs, post IDs, likedPost IDs, savedPost IDs.

2. Post module:

User would be able to add, view and delete his/her own post. During adding a post the photo would be uploaded in the folder and the path to that folder would be saved in the module. Post module would also save likes of the post.

3. Comment module:

User can add comment to the any post and a user can also delete his/her own comment.

4. Conversation module:

When a user starts a new conversation with his/her friend new Entry would be added to the database for the conversation saving userId of both the users.

5. Message module:

When a user sends a message to a friend message would be saved in this module for a particular conversation.

II. Function Prototype that implement major functionalities

1. Register

```
const Signup = () => {
  const [name, setName] = useState("")
  const [uname, setUsername] = useState("")
  const [age, setAge] = useState(0)
  const [phone, setPhone] = useState("")
  const [email, setEmail] = useState("")
  const [pass, setPass] = useState("")
  const [Repass, setRepass] = useState("")
  const [gender, setGender] = useState("")
  const publicfolder = "http://localhost:5000/Images/"
  const handleSubmit = async (e) => {
    e.preventDefault()
    try {
      if (pass !== Repass) {
        toast.error('Passwords do not match', {
          position: "top-center",
          autoClose: 5000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      } else {
        const newUser = {
          userName: uname,
          contact: phone,
          age: age,
          name: name,
          gender: gender,
          password: pass,
          email: email,
          gender: gender,
          profilepic: "AVATAR.png",
        }
        const user = await axios.post('/auth/register', newUser)
        if (user.status === 200) {
          if (user.data === "signUp failed") {
            toast.error('Username or email already exists', {
              position: "top-center",
              autoClose: 5000,
              hideProgressBar: false,
              closeOnClick: true,
              pauseOnHover: true,
              draggable: true,
              progress: undefined,
            });
          }
        }
      }
    }
  }
}
```

```
router.post("/register", async (req, res) => {
  try {
    const salt = await bcrypt.genSalt(13);
    const hashedpass = await bcrypt.hash(req.body.password, salt);
    const newUser = new User({
      userName: req.body.userName,
      password: hashedpass,
      email: req.body.email,
      age: req.body.age,
      name: req.body.name,
      contact: req.body.contact,
      gender: req.body.gender,
      profilepic: req.body.profilepic,
      emailToken: crypto.randomBytes(64).toString('hex')
    });
    const user = await newUser.save();
    console.log(process.env.GMAIL_USER);
    console.log(process.env.GMAIL_PASS);

    const mailOptions = {
      from: process.env.GMAIL_USER,
      to: user.email,
      subject: 'Verify your mail',
      html: `<p> Thanks for Registering </p>
        <p> Please verify yourself to continue </p>
        <a href="http://${req.headers.host}/api/auth/verify-email?token=${user.emailToken}">click here</a>`
    }
    transporter.sendMail(mailOptions, (err, info) => {
      if (err) {
        console.log(err);
      } else {
        console.log(info);
      }
    })
    res.status(200).json(user);
  } catch (err) {
    //
  }
})
```

2. Login

```

const Login = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [fetchUser, setFetchUser] = useState({})

  const { dispatch } = useContext(Context)

  const handleSubmit = (e) => {
    e.preventDefault();
    let status, fetched;
    const fetchUser = async () => {
      try {
        fetched = await axios.post("auth/login", { username: username, password: password });
        setFetchUser(fetched.data)
        status = fetched.status

        console.log(fetched+"\n"+status);

        if (status == 200) {
          if (fetched.data == "User not found") {
            toast.error('Incorrect Username', {
              position: "top-center",
              autoClose: 5000,
              hideProgressBar: false,
              closeOnClick: true,
              pauseOnHover: true,
              draggable: true,
              progress: undefined,
            });
          }
          else if (fetched.data == "Wrong password") {
            toast.error('Please enter correct password', {
              position: "top-center",
              autoClose: 5000,
              hideProgressBar: false,
              closeOnClick: true,
              pauseOnHover: true,
              draggable: true,
              progress: undefined,
            });
          }
          else if (!fetched.data.confirmed) {
            toast.error('Please verify email to login', {
              position: "top-center",
              autoClose: 5000,
              hideProgressBar: false,
              closeOnClick: true,
              pauseOnHover: true,
              draggable: true,
              progress: undefined,
            });
          }
          else {
            dispatch({ type: "LOGIN", payload: fetched.data })
            window.location.replace("/home")
          }
        }
      }
    }
  }
}

```

```

router.post("/login", async (req, res) => {
  try {
    {
      const user = await User.findOne({ username: req.body.username });
      const comparepass = await bcrypt.compare(req.body.password, user.password);
      !comparepass && res.status(200).json("Wrong password");
      console.log("below user");
      res.status(200).json(user);
    }
  } catch (err) {
    {
      console.log("inside catch");
      res.status(200).json("User not found")
    }
  }
})

```


3. Add Post

```
const AddPost = () => {

  const [Des, setDes] = useState("")
  const [photo, setphoto] = useState(null)

  const { user } = useContext(Context)

  const handlePostUpload = async (e) => {
    e.preventDefault()
    const data = new FormData()
    const fname = user._id + Date.now()

    const newPost = {
      photo: fname,
      caption: Des,
      userName: user.userName,
      userId: user._id
    }

    data.append('fname', fname)
    data.append("file", photo)

    try {
      const res = await axios.post('/uploads', data)
      console.log("In upload post")
      console.log(res)
    } catch (err) {
      console.log(err)
    }

    try {
      const finalPost = await axios.post('/post/addPost', newPost)
    } catch (err) {
      console.log(err)
    }

    window.location.replace("/home")
  }
}
```

```
router.post("/addPost", async (req, res) => {
  try {
    const newPost = new Post({
      photo: req.body.photo,
      caption: req.body.caption,
      userName: req.body.userName,
      userId: req.body.userId
    })
    const post = await newPost.save()
    const user = await User.findOne({ userName : req.body.userName })
    const currentpostCount = user.postCount
    await user.updateOne({
      $push: { posts: post._id },
      postCount : currentpostCount + 1
    })
    res.status(200).json(post)
  } catch (err) {
    res.status(403).send("Error updating your Post")
  }
})
```

4. View Post

```
const ViewPost = () => {
  const location = useLocation()
  const postId = location.pathname.split('/')[2]
  const [post, setpost] = useState({})
  const [comment, setcomment] = useState([])
  const [Newcomment, setNewcomment] = useState("")
  const [isLiked, setIsLiked] = useState(false)
  const [isSaved, setIsSaved] = useState(false)
  const publicFolder = "http://localhost:5000/Images/"
  const { user, dispatch } = useContext(Context);
  const [curruser, setcurruser] = useState({})

  useEffect(() => {
    const fetchPost = async () => {
      try {
        const currPost = await axios.get(`/post/${postId}`)
        setpost(currPost.data)
        const currId = currPost.data.userId
        const usertmp = await axios.get(`/user/${currPost.data.userId}`)
        setcurruser(usertmp.data)
        const comments = await axios.get(`/comment/${postId}`)
        setcomment(comments.data)

        currPost.data.likedBy.includes(user._id) && setIsLiked(true)
        user.savedposts.includes(currPost.data._id) && setIsSaved(true)
      }
      catch (err) {
      }
    }
    fetchPost()
  }, [1])
}
```

```
router.get("/:id", async (req, res) => {
  try {
    const post = await Post.findById(req.params.id)
    res.status(200).json(post);
  } catch (err) {
    res.status(403).send("Error reading your Post")
  }
})
```

5. Delete Post

```
const handleDelete = async () => {
  console.log(postId)
  try {
    console.log("above")
    const comments = await axios.delete(`/comment/${postId}/deleteAll`)
    const post123 = await axios.delete(`/post/${postId}/${user._id}`)
    const fetchedUser = await axios.get(`/user/${user._id}`);
    dispatch({type: 'UPDATE', payload: fetchedUser.data});
    window.location.replace('/home')
  } catch (err) {
  }
}
```

```

router.delete('/:id/:uid', async(req, res) => {
  try {
    const post1 = await Post.findById(req.params.id)
    const post = await Post.findByIdAndDelete(req.params.id)
    console.log(req.params.uid)
    const user = await User.findById(req.params.uid)
    const currentpostCount = user.postCount
    console.log("lower"+post1._id)

    if(user.posts.includes(post1._id))
    {
      console.log("in first if")
      await user.updateOne({
        $pull: {posts: post1._id},
        postCount : currentpostCount - 1
      })
      if(user.posts.includes(post1._id))
      {
        console.log("in second if")

        let ind = user.posts.indexOf(post1._id)
        user.posts.splice(ind,1)
        user.postCount = user.postCount-1;
        await user.save()
      }
    }
    let likedBy = post1.likedBy
    likedBy.forEach(async(e)->
    {
      const curruser = await User.findById(e)

      if(curruser.likedposts.includes(post1._id))
      {
        let ind = curruser.likedposts.indexOf(post1._id)
        curruser.likedposts.splice(ind,1)
        await curruser.save()
      }
    })
    let savedBy = post1.savedBy
    savedBy.forEach(async(e)->
    {
      const curruser = await User.findById(e)
      if(curruser.savedposts.includes(post1._id))
      {
        let ind = curruser.savedposts.indexOf(post1._id)
        curruser.savedposts.splice(ind,1)
        await curruser.save()
      }
    })
    res.status(200).json("Post Deleted successfully");
  } catch(err){
    res.status(200).json(err.message)
  }
})

```

6. Like post and save post

```
const handleLike = async () => {
  try {
    const tmpUser = {
      userId: user._id,
    }
    const likedPost = await axios.put(`/post/${post._id}/likepost`, tmpUser)
    const fetchedUser = await axios.get(`/user/${user._id}`);
    dispatch({type: 'UPDATE', payload: fetchedUser.data});
    setIsLiked(!isLiked)
  } catch (err) {
  }
}

const handleSave = async () => {
  try {
    const tmpUser = {
      userId: user._id,
    }
    const savedPost = await axios.put(`/user/${post._id}/savedPost`, tmpUser);
    setIsSaved(!isSaved);
    const fetchedUser = await axios.get(`/user/${user._id}`);
    dispatch({type: 'UPDATE', payload: fetchedUser.data});
  } catch (err) {
  }
}
```

```
router.put('/:id/likepost', async (req, res) => {
  try {
    const post = await Post.findById(req.params.id)
    let likes = post.likeCount
    const user = await User.findById(req.body.userId)
    if (post.likedBy.includes(req.body.userId)) {
      await post.updateOne({
        $pull: {likedBy: req.body.userId}, likeCount: likes - 1
      })
      await user.updateOne({
        $pull: {likedposts: req.params.id}
      })
    } else {
      await post.updateOne({
        $push: {likedBy: req.body.userId}, likeCount: likes + 1
      })
      await user.updateOne({
        $push: {likedposts: req.params.id}
      })
    }
    const updatedPost = await Post.findById(req.params.id)

    res.status(200).json(updatedPost)
  } catch (err) {
    res.status(403).send(err.message)
  }
})
```

```

router.put("/:id/savedPost", async (req, res) => {
  try {
    const user = await User.findById(req.body.userId)
    const post = await Post.findById(req.params.id)
    if (user.savedposts.includes(req.params.id)) {
      await user.updateOne({
        $pull: { savedposts: req.params.id }
      })
      await post.updateOne({
        $pull: { savedBy: req.body.userId }
      })
    } else {
      await user.updateOne({
        $push: { savedposts: req.params.id }
      })
      await post.updateOne({
        $push: { savedBy: req.body.userId }
      })
    }

    const updatedUser = await User.findById(req.body.userId)
    res.status(200).json(updatedUser)
  } catch (err) {
    res.status(200).json(err.message)
  }
})

```

7. Add Comment , Delete Comment

```

const handleComment = async () => {
  try {
    const PushComment = {
      comment: Newcomment,
      postId: postId,
      userId: user._id,
    }
    const response = await axios.post('/comment/', PushComment)
    setNewcomment("")
    const comments = await axios.get(`/comment/${postId}`)
    setcomment(comments.data)
  } catch (err) {
  }
}

const DeleteComment = async (com) => {
  try {
    const delete_comment = await axios.delete(`/comment/${com._id}`)

    const comments = await axios.get(`/comment/${postId}`)
    setcomment(comments.data)
  } catch (err) {
  }
}

```

```

// Add comment
router.post("/", async(req, res) =>
{
  try
  {
    const comment = new Comment({
      comment: req.body.comment,
      userId : req.body.userId,
      postId : req.body.postId
    })
    const newcomment = await comment.save();
    res.status(200).json(newcomment);
  } catch(err)
  {
    res.status(403).json(err);
  }
})

// Delete comment
router.delete("/:id", async(req, res) => {
  try
  {
    await Comment.findByIdAndDelete(req.params.id);
    res.status(200).send("Deleted");
  }
  catch(err)
  {
    res.status(403).send("Unable to delete");
  }
}
)

```

8. Add friend

```

const handleAddfriend = async() =>
{
  try
  {
    const friendAdded = await axios.put('/user/'+user._id+'/addFriend/', {userId: topProfile.userId});
    dispatch({type: 'UPDATE', payload: friendAdded.data});
    // setUserstate(friendAdded);
    window.location.href = "/userProfile/" + topProfile.userId;
  }
  catch(err)
  {
    console.log(err)
  }
}

```

```

router.put("/:id/addFriend", async(req, res) => {
  try {
    const user = await User.findById(req.params.id);
    const currentFriendcount = user.friendCount;
    const otheruser = await User.findById(req.body.userId);
    const otheruserFriends = otheruser.friendCount
    if(!user.friends.includes(req.body.userId))
    {
      await user.updateOne({$push: {friends: req.body.userId}, friendCount : currentFriendcount + 1})
      await otheruser.updateOne({
        $push: {friends: req.params.id},
        friendCount : otheruserFriends + 1
      })
    }
    else
    {
      await user.updateOne({$pull: {friends: req.body.userId}, friendCount : currentFriendcount - 1})
      await otheruser.updateOne({
        $pull: {friends: req.params.id},
        friendCount : otheruserFriends - 1
      })
    }
    res.status(200).json(user)
  }
  catch{
    res.status(200).send(err.message);
  }
})

```

9. Search User

```
const Search = () => {

  const [searchResult, setSearchResult] = useState([])
  const location = useLocation()
  const searchTerm = location.pathname.split('/')[2];

  console.log(searchTerm)

  useEffect(() => {
    const getSearchProfile = async () => {
      const profile = await axios.post(`/user/searchProfile`, { searchTerm:searchTerm })
      setSearchResult(profile.data)
    }
    getSearchProfile()
  }, [])

  router.post("/searchProfile", async(req,res)=>{
    try{
      const regexterm = new RegExp( req.body.searchTerm,'i')
      const users = await User.find({name: {$regex:regexterm}})
      res.status(200).json(users)
    }catch(err){

      console.log(err.message)
    }
  })
}
```

10.Chat Functionality

```
useEffect(() => {
  socket.current.emit("addUser",user._id);
  socket.current.on("getUser",(users)=>{
    setOnlineUsers(users)
    users.forEach((e)=>{
      user.friends.forEach((e1)=>
      {
        console.log("e1 : "+e1);
        if(e.userId==e1)
        {
          console.log("INSIDE e==e1")
          onlinefriends_t.push(e1)
          setOnlinefriend(onlinefriends_t)
        }
      })
    })
  })
}, [user])
```

```

useEffect(() => {
  socket.current = io("ws://localhost:8900");
  socket.current.on("getMessage", (data) => {
    setArrivalMessages({
      sender: data.senderId,
      text: data.text,
      createdAt: Date.now(),
    })
  })

  if(urlarray.length==3)
  {
    let convoId = urlarray[2];
    const fetchConvo = async() => {
      const convo = await axios.get(`/conversation/convo/${convoId}`);
      setCurrChat(convo.data);
    }
    fetchConvo();
  }
}, [])

```

```

const handleSubmit = async(e) =>{
  e.preventDefault()
  const currmessage={
    sender : user._id,
    text:newmessages,
    conversationId:currChat._id
  }
  const receiverId = currChat.members.find(member=>member !== user._id)
  socket.current.emit("sendMessage",{
    senderId:user._id,
    receiverId,
    text:newmessages,
  })

  try {
    const res = await axios.post("/message",currmessage)
    setMessages([...messages,res.data])
    setNewMessages("")
  } catch (err) {
    console.log(err.Message)
  }
}

```



```

const addUser = (userId,socketId) =>{
  !users.some((user)=> user.userId === userId) && users.push({userId,socketId})
}

const removeUser = (socketId) =>{
  users = users.filter((user)=> user.socketId !== socketId)
}

const getUser =(userId)=>{
  // console.log("User id : "+userId)
  console.log("Users"+users)
  return users.find((user)=>user.userId === userId)
}

io.on('connection', (socket) => {
  // connection establish
  console.log('a user connected');

  // messenger opened
  socket.on("addUser",userId=>{
    console.log("INSIDE ADDUSER")
    addUser(userId,socket.id)
    console.log(users)
    io.emit("getUser",users)
  })

  //send message
  socket.on("sendMessage",({senderId,receiverId,text})=>{
    const user = getUser(receiverId);
    console.log("user : "+user)
    console.log("receiver id : "+receiverId)
    io.to(user.socketId).emit("getMessage",{
      senderId,
      text,
    })
  })

  socket.on("disconnect",()=>{
    console.log("a user Disconnected");
    removeUser(socket.id)
    io.emit("getUser",users)
  })
});

router.post("/",async(req,res)=>{
  const newConversation = new Conversation({
    members : [req.body.senderId,req.body.receiverId],
  });
  try {
    const savedConversation = await newConversation.save()
    res.status(200).json(savedConversation)
  } catch (err) {
    res.status(200).json(err)
  }
});

router.get("/convo/:id",async(req,res)=>{
  try{
    const convo = await Conversation.findById(req.params.id);
    return res.json(convo);
  }catch(err){
    res.status(200).json(err)
  }
});

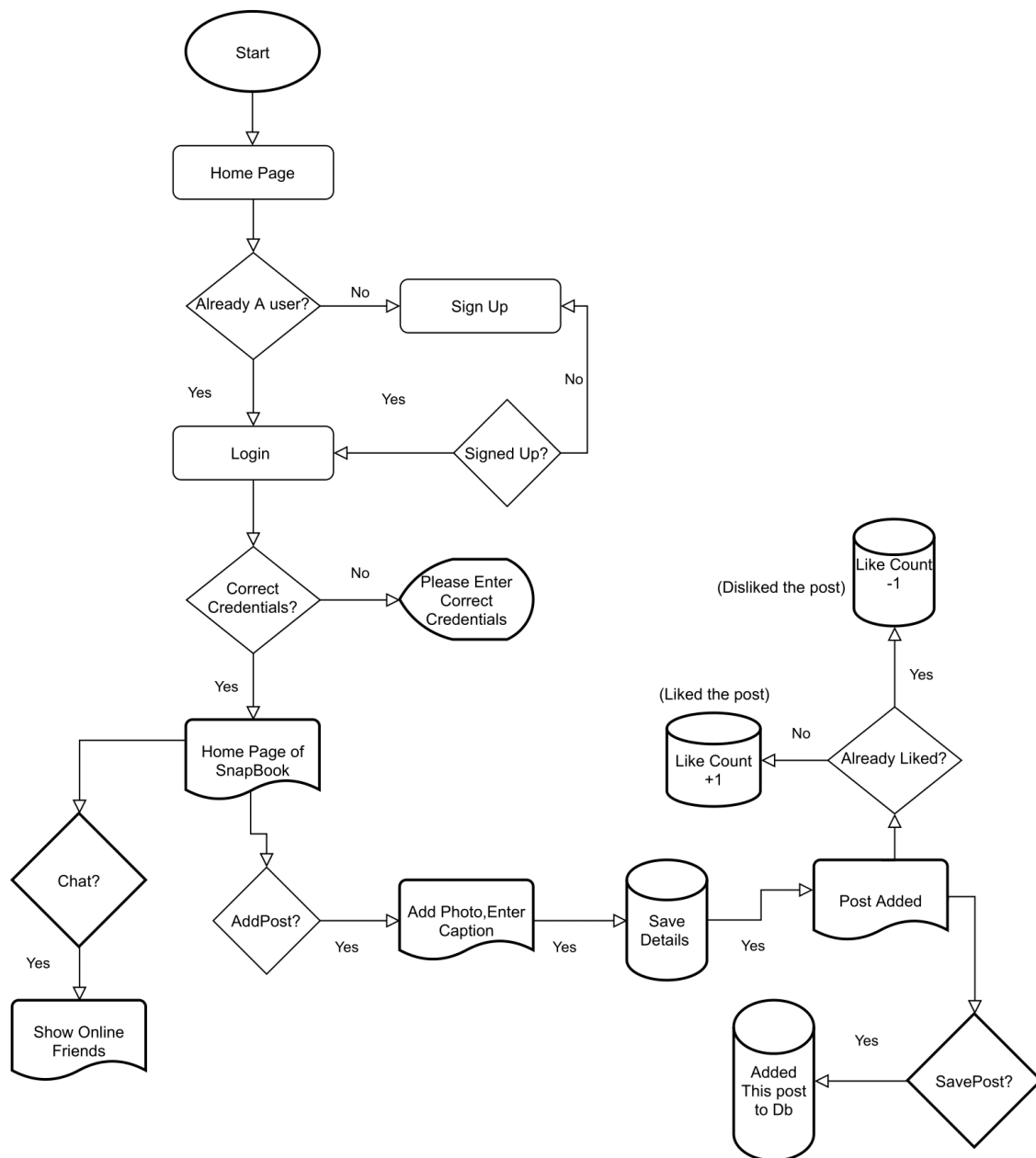
router.get("/:rId/:sId",async(req,res)=>{
  try{
    const conversations = await Conversation.find({
      members : {$in : [req.params.rId]},
    })
    let conversation1 = [];
    conversations.forEach((e)=>{
      if(e.members[0]==req.params.sId || e.members[1]==req.params.sId)
      {
        conversation1.push(e)
        return res.json(conversation1);
      }
    })
    return res.json(conversation1);
  }catch(er){
    res.status(200).json(err)
  }
});

router.get("/:id",async(req,res)=>{
  try {
    const conversations = await Conversation.find({
      members : {$in : [req.params.id]},
    })
    res.status(200).json(conversations)
  } catch (err) {

```

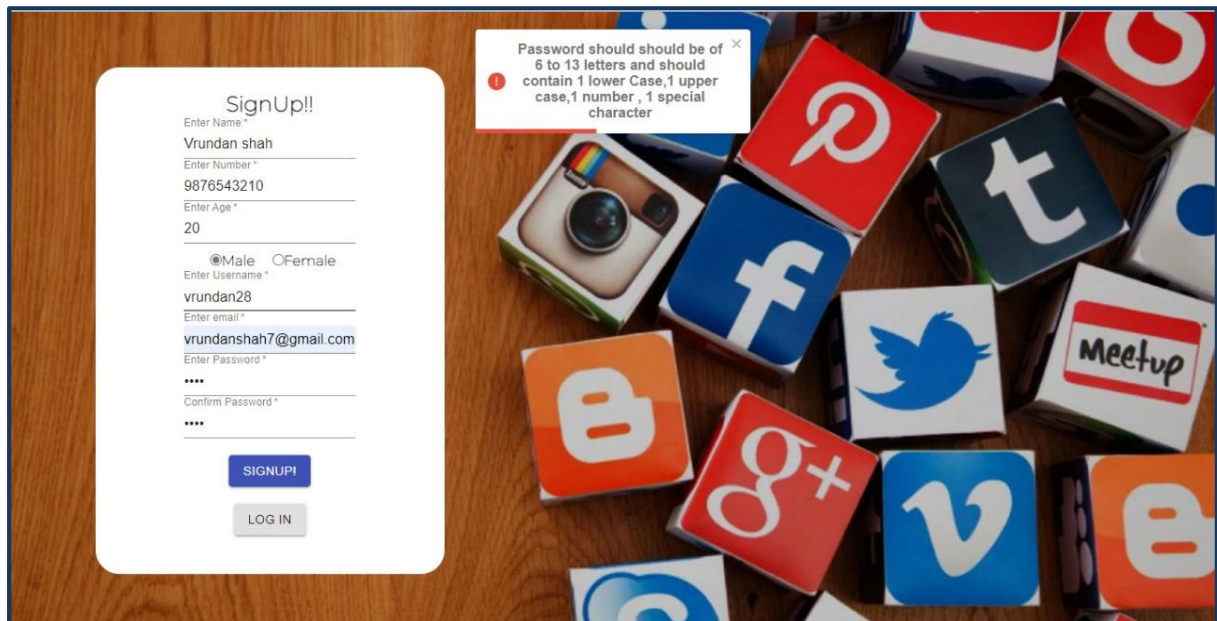
III) Algorithm/FlowChart

FlowChart

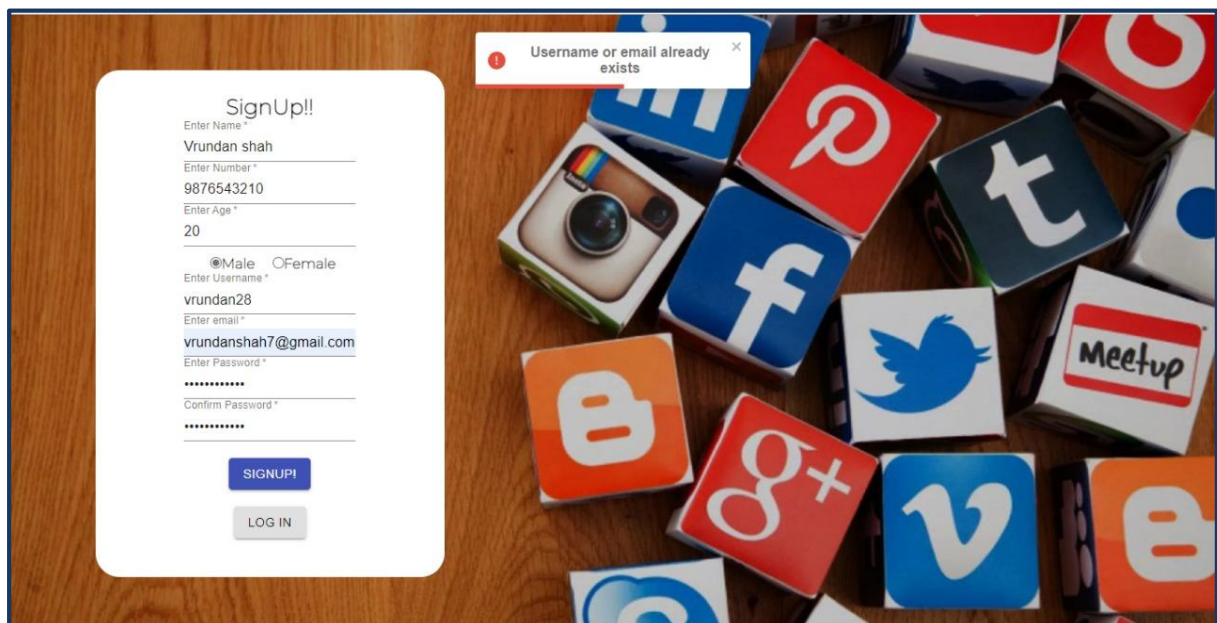


9. Testing - Manual Testing.

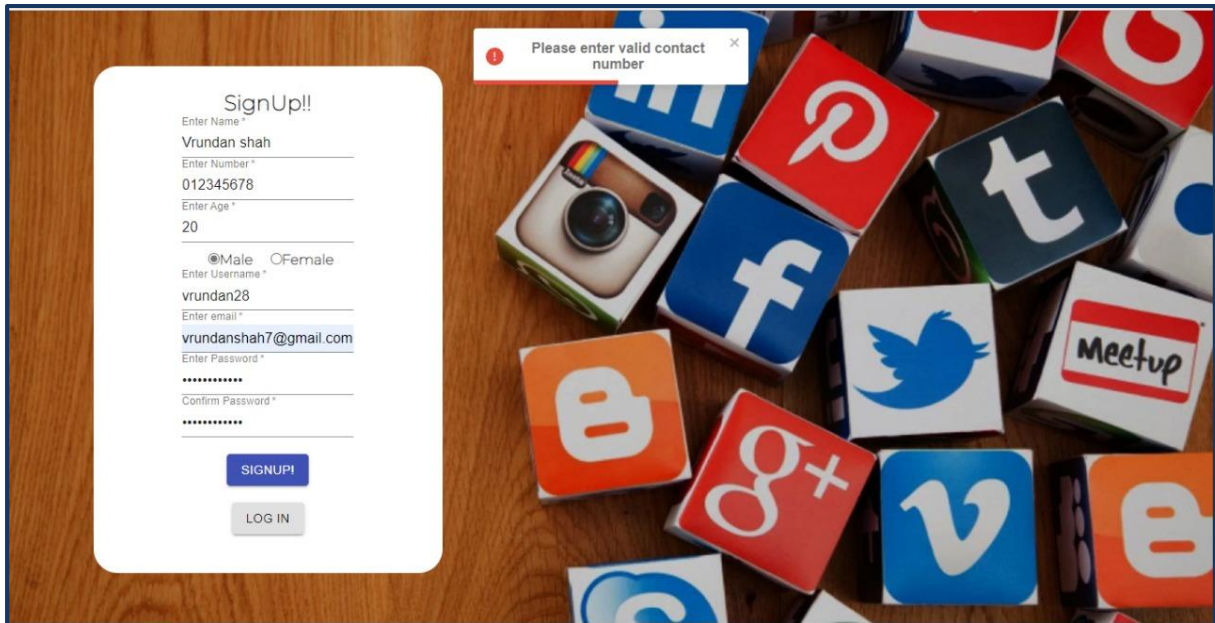
1. Password Validations



2. Already Existing Username or Email



3. Contact number validation



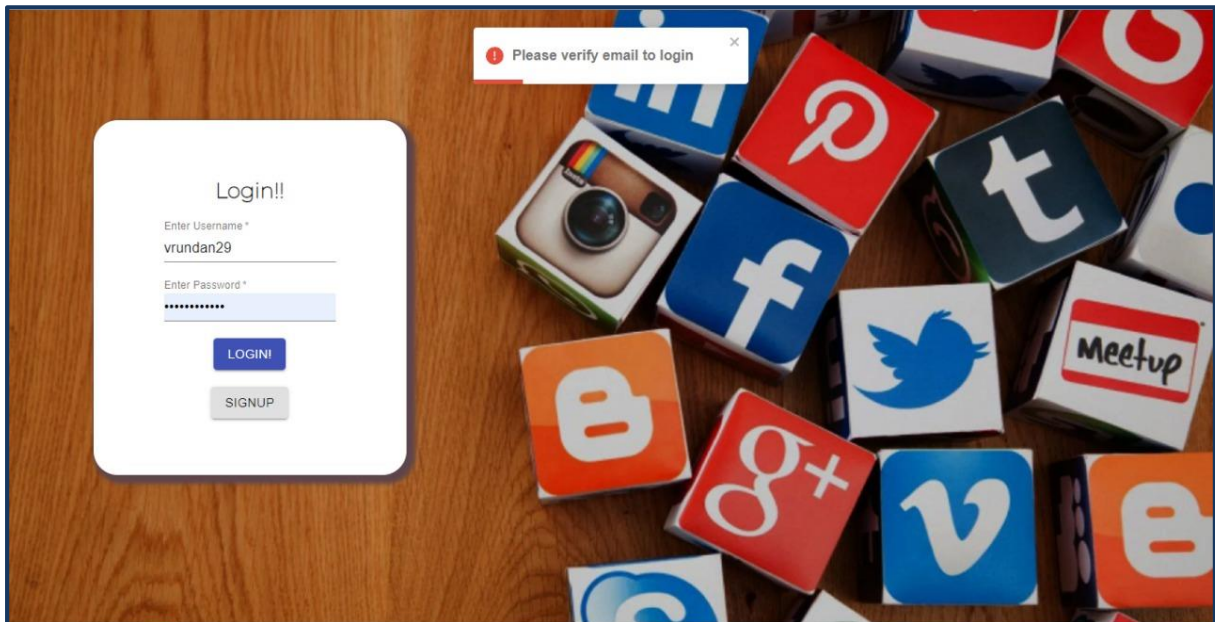
The screenshot shows a 'SignUp!!' form on a wooden background with various social media icons. The form fields are as follows:

- Enter Name*: Vrundan shah
- Enter Number*: 012345678
- Enter Age*: 20
- Gender: ☒ Male ☐ Female
- Enter Username*: vrundan28
- Enter email*: vrundanshah7@gmail.com
- Enter Password*: [masked]
- Confirm Password*: [masked]

Buttons: SIGNUP (blue), LOG IN (grey).

An error message box at the top right states: "Please enter valid contact number".

4. Try to login without verifying email



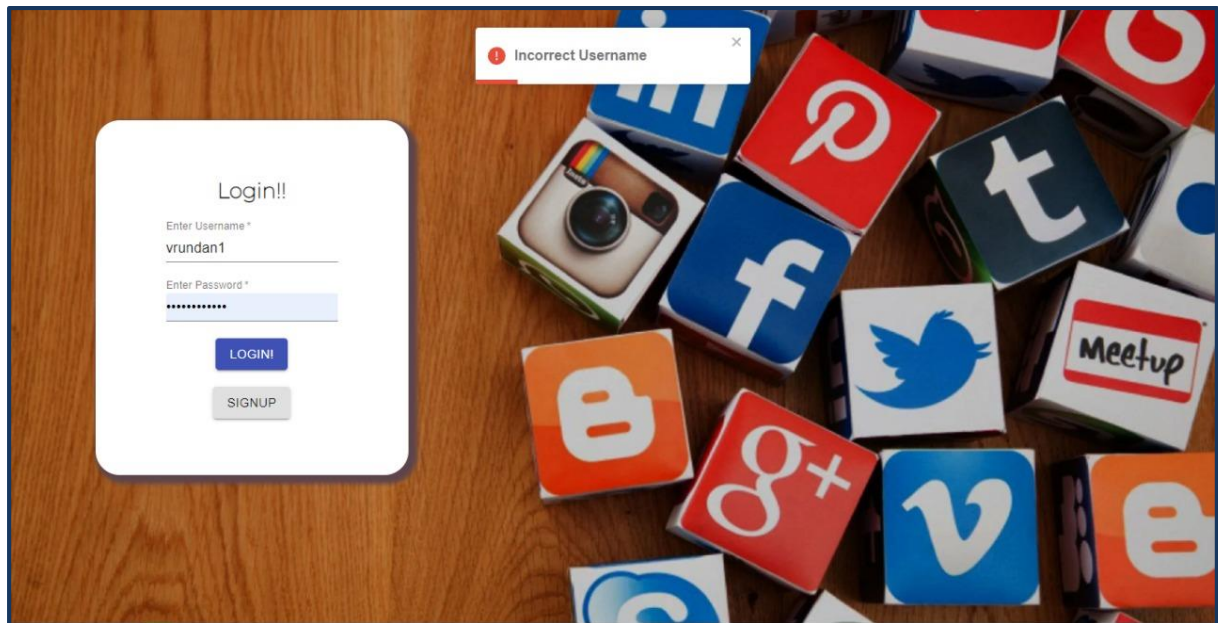
The screenshot shows a 'Login!!' form on a wooden background with various social media icons. The form fields are as follows:

- Enter Username*: vrundan29
- Enter Password*: [masked]

Buttons: LOGIN! (blue), SIGNUP (grey).

An error message box at the top right states: "Please verify email to login".

5. Trying to login with incorrect username

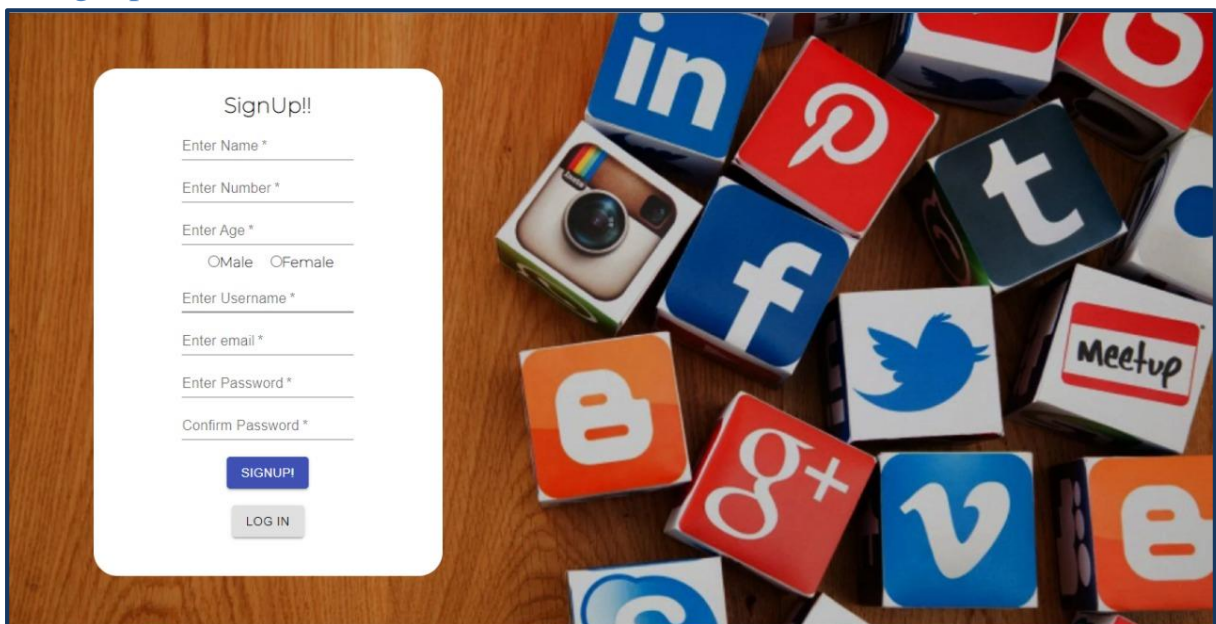


10. Screenshots

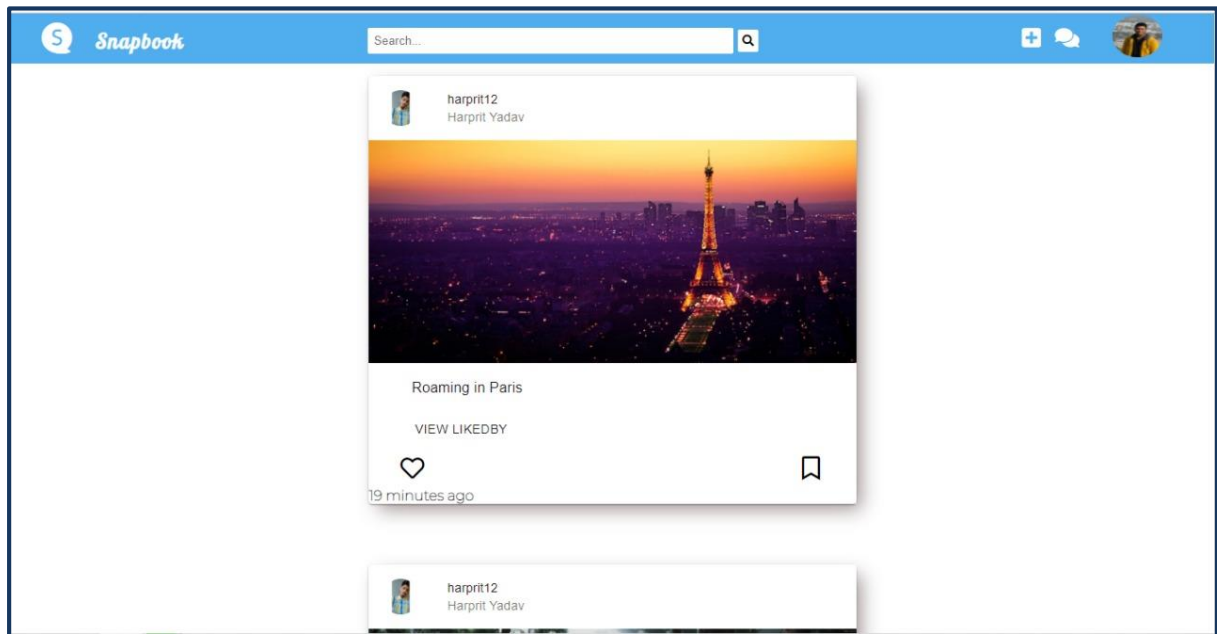
1. Login



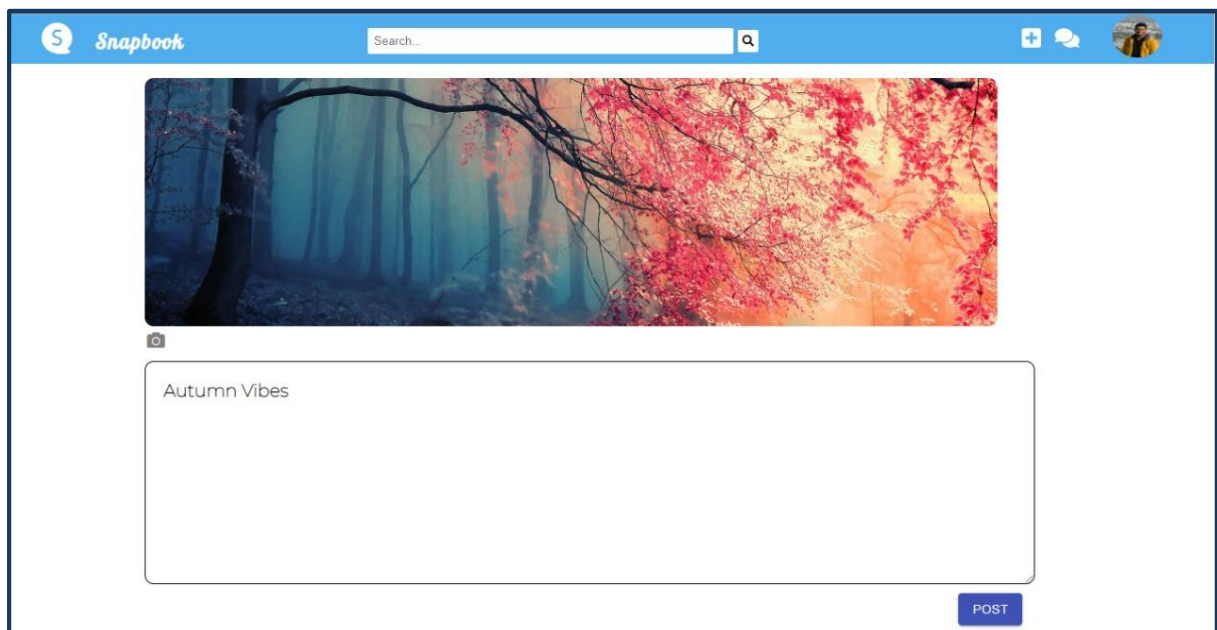
2. Signup



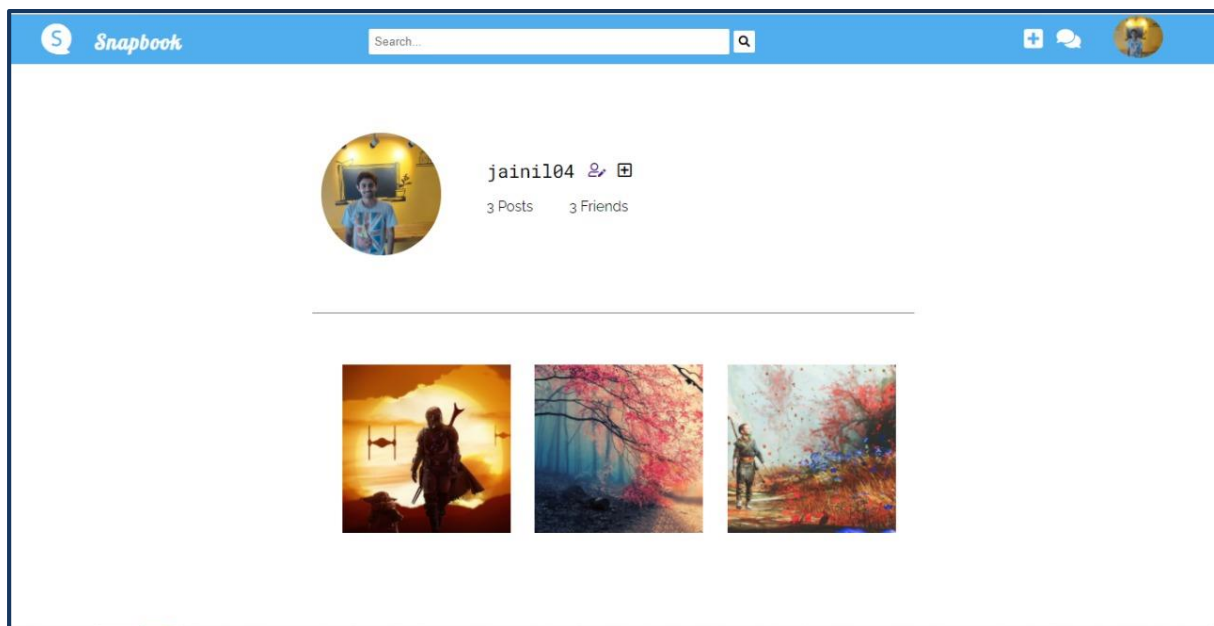
3. Home



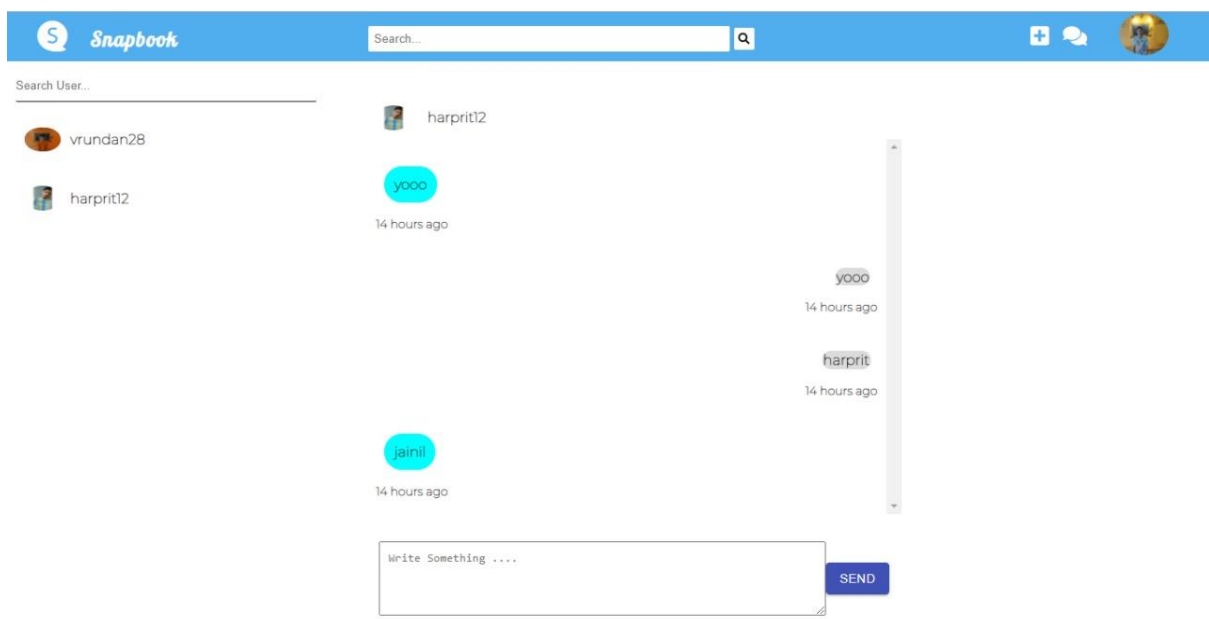
4. Add Post




5. User Profile





6. Chat Functionality



7. Edit Profile





Name

Vrundan Shah

Username

vrundan28

Phone


9687642782

Age


18

Update!

8. Single Post





harprit12
Harprit Yadav




Roaming in Paris


VIEW LIKEDBY

 34 minutes ago



Comments

 jainil04
nice pic
3 minutes ago

 dhruval11
good
2 minutes ago

Comment

COMMENT

11. Conclusion

Hereby, we conclude that we have successfully implemented the user module, post module and chat module.

User will be able to create an account, share his/her posts, view and edit profile, view friends profile and add them as friend, see and like friend's posts in feed as well as save them, see all their friends who are online and chat with them, comment on posts, view their saved and liked posts and search profile of the user they want.

After successful implementation of the system, testing was done by the team; also we asked different end-users what they expect out of a social media application. So by this technique we can find if there is any scope of improvement and if there were any flaws in the system.

12. Limitations and Future Extensions

Limitations :

1. Cannot upload post directly with camera.
2. Cannot Add Video as post.
3. Chat with online friends only
4. Everyone can see your profile.

Future extensions :

1. Deployment
2. Private/Public Profile
3. Can only send message to online
4. Location
5. Status (Like travelling somewhere , eating at some place, posting thoughts)
6. Story (24 hour)

13. Bibliography

- 1) <https://www.w3schools.com/>
- 2) <https://reactjs.org/>
- 3) <https://docs.mongodb.com/>
- 4) Technical Thapa (Youtube Channel)
- 5) <https://mui.com/>