DHARMSINH DESAI UNIVERSITY, Nadiad

Faculty of Technology

Department of Computer Engineering

B. Tech. CE Semester – V

Subject: (CE – 5 (2023 - 2024)) Advanced Technologies

Project Title: **Instagram**

By: Bhandari Nishant, CE008, 21CEUBG051
Chhipani Devanshu, CE018, 21ECUOS102

Guided by: prof. Siddharth Shah

DHARMSINH DESAI UNIVERSITY, Nadiad

Faculty of Technology

Department of Computer Engineering

CERTIFICATE

This is to certify that the practical/term work carried out in the subject of Advanced Technology and recorded in this journal is bonafide work of Mr **CE008 Bhandari Nishant (21CEUBG051) , CE018 Chhipani Devanshu (21ECUOS102)** of B. Tech. semester V in the branch of **Computer Engineering**, during the academic year 2023-2024.

Prof. Siddharth Shah	Dr. C.K. Bhensdadia
Dept of computer engg.	Head of Dept. of Computer engg.
Faculty of technology	Faculty of technology
Date	Date

Contents

Abstract

Our **Instagram Clone Website** is a modern web-based application designed to connect peoples, enable communication and content sharing. They can create accounts, share photos, follow other users, like and comment on posts, and explore content from other users. With the growing influence of social media in our daily lives, this website aims to provide users with a platform to express themselves, stay connected with friends and family, discover new interests, and engage with a wide range of content.

Introduction

The purpose of developing this project is to create a web-based application that will implement all major functionalities of Instagram. By developing this clone, we aim to provide users with an alternative platform that offers similar features like uploading and sharing photos, following other users, liking and commenting on posts, and exploring content in a visually appealing manner and to connect peoples, enable communication and content sharing using social media platforms. Users can efficiently connect with new peoples from around the world.

We have developed a Web-Application using MERN(MongoDb Express React Node) Stack Technology which is based on JavaScript. We have used Vs-Code(Visual Studio Code) for developing and programming. For development planning we have used Notion in which we assign tasks and for maintaining Code backup and sharing with each other for at the same time development we are using GitHub.

Key Features :

- Authentication
- User profile page
- Search user profile
- Users can share posts.
- Users can Like , share and comment on other user's content.
- Get all followers and following user post
- Get all followers and following user status
- Chat with all followers and following users

Software Requirement Specification (SRS)

1. Introduction

1.1 Purpose

The purpose of developing this project is to create a web-based application that will implement all major functionalities of Instagram. By developing this clone, we aim to provide users with an alternative platform that offers similar features like uploading and sharing photos, following other users, liking and commenting on posts, and exploring content in a visually appealing manner. and to connect peoples, enable communication and content sharing using social media platform. Users can efficiently connect with new peoples from around the world.

1.2 Document Conventions

In this we have use two(2) font sizes and three(3) types of fonts. For heading & label we have used font size of 15 and for main heading font style is Lexend and for semi Heading we have used Lore font style. For normal text Arial font style is used and 13 sized is used.

1.3 Intended Audience and Reading Suggestions

The intended audience for this website is primarily users who are looking for
a convenient and efficient way to connect with new people. This website is for all user ages who have access to the internet and are comfortable to connect with people on social media platform.

1.4 Product Scope

In social media website user can connect peoples, enable communication and content sharing. They can create accounts, share photos, follow other users, like and comment on posts, and explore content from other users.This website aims to provide users with a platform to express themselves, stay connected with friends and family, discover new interests, and engage with a wide range of content.

1.5 References

Fundamentals of Software Engineering by Rajib Mall, PHI Learning
Instagram - Official website and mobile application.

2. Overall Description

2.1 Product Perspective

The Instagram Clone is an independent web-based application developed as a replica of Instagram. While it aims to emulate the core features of Instagram, it is not affiliated with or endorsed by Instagram in any way. The application will be designed to operate as a standalone platform, allowing users to register, create profiles, post pictures, follow other users, and do interaction with people.

2.2 Product Functions

- Search User Profile by Name
- Display Followers and Following Posts on Home Page
- Display Followers and Following Status on Home Page
- Open Reels Page (Optional)
- Open Message Panel
- Redirect to User Profile Page
- Track User Notifications
- User Login and Signup
- Share Post (Text, Image)
- Interact with Posts

2.3 Operating Environment

The Instagram Clone will be developed as a web application, and users will access it using standard web browsers like web browsers, including Chrome, Firefox, Safari, Edge, and Internet Explorer. The machine should have sufficient ram and internet connection.

2.4 Design and Implementation Constraints

Instagram clone is a modern web based application which will be made of mern- stack technology. In this we are gonna use React, Redux and tailwind for frontend development, for the database we are going to use MangoDb and for the backend we will use Express and Node Js.

2.5 User Documentation

The project will include comprehensive user documentation to help users navigate and understand the application's features effectively. The documentation will cover user registration, profile management, posting content, interacting with other users, and other essential functionalities.

2.6 Assumptions and Dependencies

➢ Assumptions:
- Users have basic knowledge of using web browsers and are familiar with common internet browsing practices.
- Users will abide by the website's terms of service and community guidelines when creating and sharing content.

➢ Dependencies:
- Reliable and secure internet infrastructure to ensure seamless access to the website.

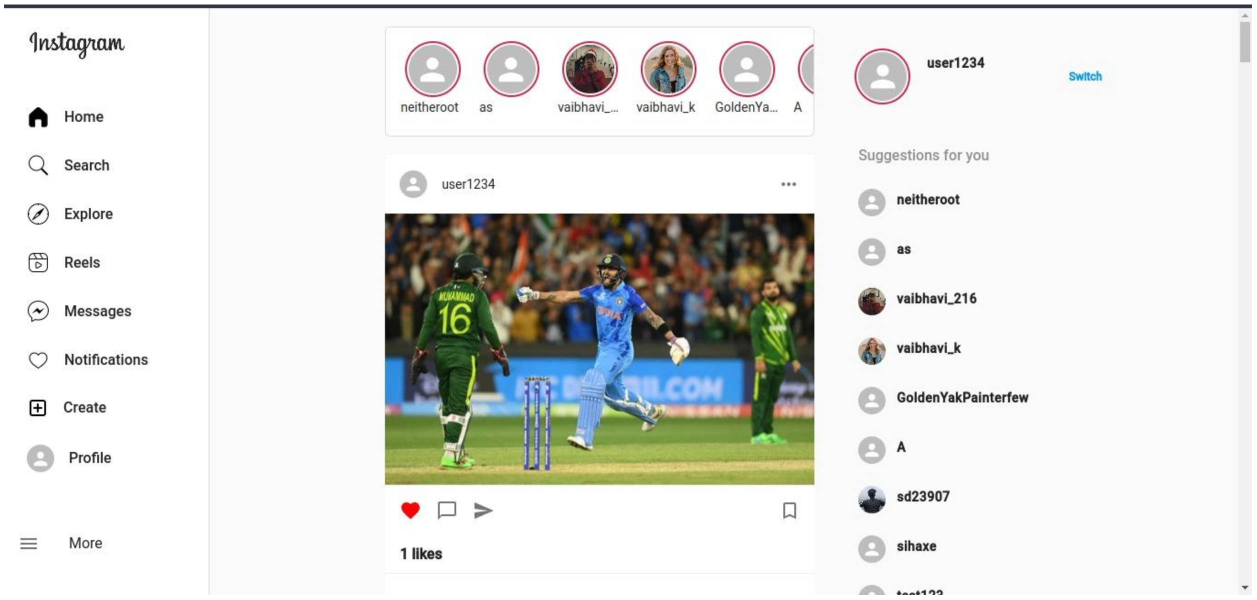
- Database management system to store user profiles, posts, and other data securely.
- Web hosting services to host the Instagram website and handle user requests.
- Development and maintenance teams to continuously update and enhance the website's features and security.

3. External Interface Requirements

3.1 User Interface Description for Instagram Website

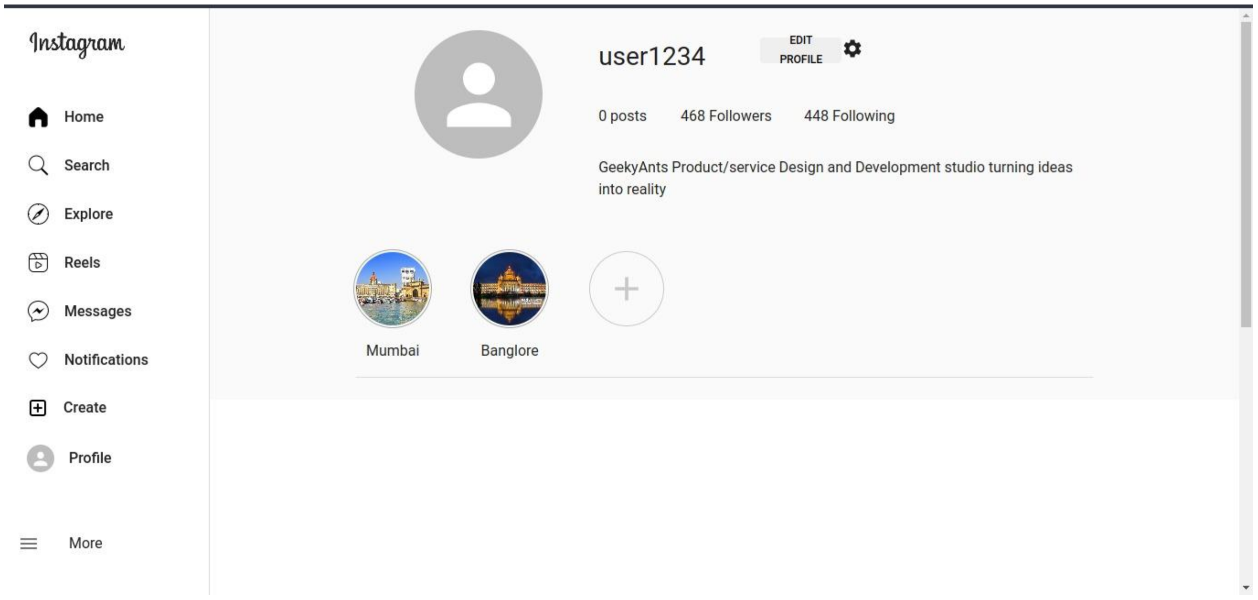
1. Home Page:

- The home page features a search bar at the top, allowing users to search for other users by their names.
- Below the search bar, a scrolling feed displays posts from users whom the current user follows and who follow the current user.
- Each post includes the user's profile picture, username, post content (text or image), like, share, and comment buttons.
- Users can click on the profile picture or username to visit the user's profile.
- The home page also includes buttons for accessing the reels page, message panel, and notifications.



2. User Profile Page:

- The user profile page displays the user's profile picture, username, total number of posts, total followers, total following, status, and bio.
- Users can edit their profile details, including profile picture, username, status, and bio.
- On the profile page, users can see three types of posts:
- Posts shared by the user.
- Saved posts from all post selections made by the user.
- Tagged posts where the user has been mentioned by other users.



GeekyAnts Product/service Design and Development studio turning ideas into reality

3. Reels Page:

- The reels page features a grid of short videos shared by other users.
- Users can scroll through the videos and watch them by clicking on a specific video.
- Like, comment, and share buttons are available for each video.

4. Message Panel:

- The message panel allows users to communicate with other users via text messages.
- Users can see a list of their followers and following users and click on a user's name to start a conversation.
- The message panel displays the conversation history with a specific user.

5. Notifications:

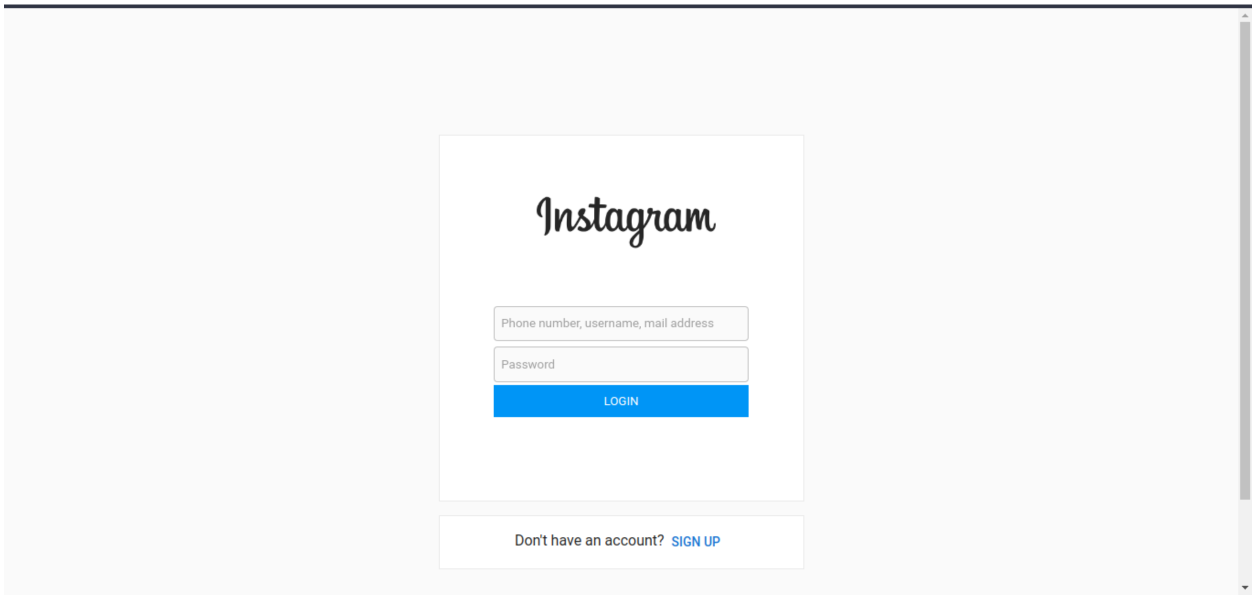
- The notifications section displays a list of notifications related to the user's activity and interactions.
- Notifications may include likes, comments, followers, and mentions.

6. Settings Page:

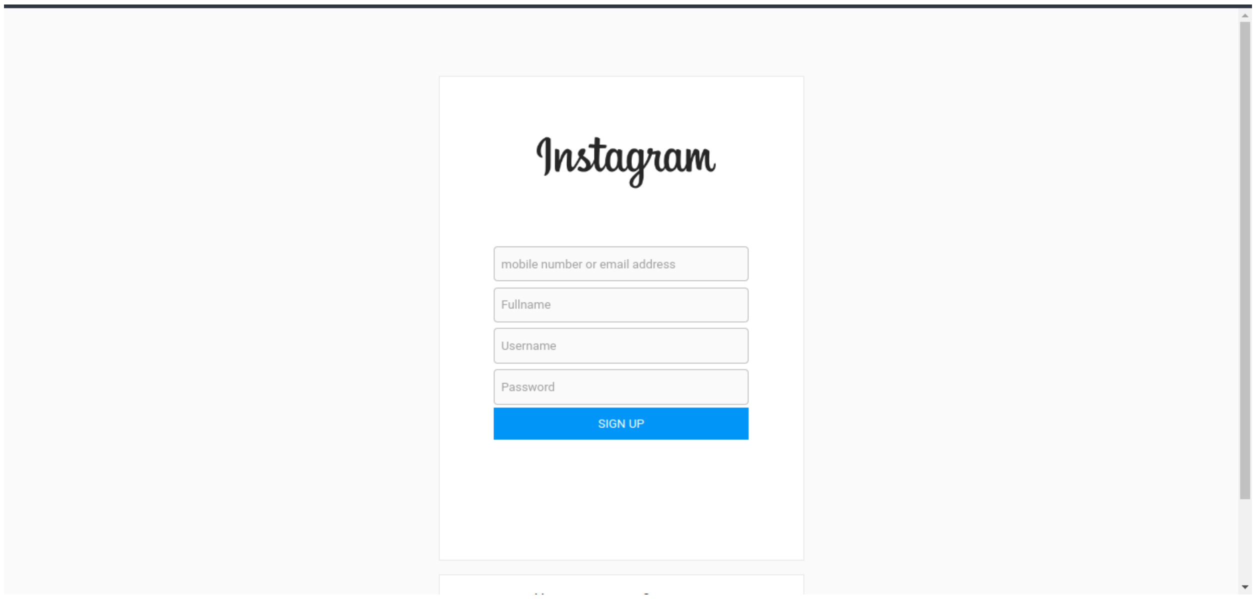
- The settings page allows users to update their credentials, such as password and email.
- Users can also manage notification preferences and other account-related settings.

7. User Login and Signup:

- Users can log in or sign up using their email and password.
- In case of new users, a registration form collects necessary information for account creation.


Instagram
Phone number, username, mail address
Password
LOGIN
Don't have an account? SIGN UP

Login page



Signup page

3.2 Hardware Interfaces

- The system is compatible with a wide range of hardware, including desktop computers, laptops, and mobile devices. The website will be able to run on popular operating systems such as Windows, MacOS, and Linux.
- RAM: 4GB
- Hard Drive Storage Needed: 2GB
- Other Hardware Requirement: None

3.3 Software Interfaces

1. <u>Front-End Interfaces</u>:

React Components:

- Various React components for different parts of the Instagram website, such as Home Page, User Profile Page, Reels Page, Chat Page, etc.
- Each component communicates with the back-end to fetch data and update the user interface accordingly.

Redux Store:
- The Redux store holds the application's state, including user authentication status, user data, posts, and other relevant information.
- Components interact with the Redux store to access and update application state using Redux actions and reducers.

API Endpoints:
- Front-end components interact with the back-end API endpoints to perform CRUD (Create, Read, Update, Delete) operations for user profiles, posts, and other data.
- These API endpoints are defined using Express.js on the server-side.

HTTP Requests:
- The front-end makes HTTP requests to the back-end API using libraries like Axios or Fetch to send and receive data.

User Authentication:
- Front-end components handle user authentication processes such as login, signup, and user session management.
- JWT (JSON Web Tokens) might be used to manage user sessions and secure API endpoints.

2. Back-End Interfaces:

Express.js Middleware:
- Middleware functions in Express.js handle tasks like parsing incoming requests, logging, error handling, etc.
- Authentication middleware may be used to verify user tokens for protected routes.

MongoDB Database:
- The back-end interacts with the MongoDB database to store and retrieve data related to user profiles, posts, notifications, etc.
- Mongoose may be used as an Object Data Modeling (ODM) library to interact with MongoDB.

API Routes:

- Express.js defines API routes for various functionalities, like fetching user data, creating posts, managing followers, etc.
- These routes handle HTTP requests from the front-end and interact with the database accordingly.

Authentication Endpoints:
- Express.js handles authentication endpoints, such as login and signup, to verify user credentials and provide access tokens.

File Upload/Storage:
- If the website supports image/video uploads for posts and profile pictures, the back-end might interface with a cloud storage service like cloudnary for file storage.

3.4 Communications Interfaces

This uses standard network protocols, such as HTTPS.The system will comply with industry standards for data encryption and secure communication to ensure that customer information and orders are protected from unauthorized access or attack.

4. System Features

R.1 Login and Register

Description : Users can login and register on our website by itself.

R.1.1 Login

Input : Users can enter their credentials.
Processing : Our website will check that the user is authenticated or not.
Output : User is successfully logged in.

R.1.2 Register

Input : User can create new accounts by entering their personal details.
Processing : Our website will create an account for user in the database.
Output : User has successfully created an account.

R.2 Find other user's profile

Description : Users can search other users in the search box.

Input : User will enter other user's name whom he want to search.
Processing : Our website will search user's profiles in our database.
Output : user will get the list of searched user's profiles.

R.3 Sharing content

Description : User can upload the content in the form of post which can be image or video. He can also share stories with their followers.

R.3.1 Sharing post

Input : User will upload pictures and video by choosing from file manager, enter caption, location and can tag people.
Processing: Website will upload the files in cloud and show their follower's feed.
Output : Website will show the uploaded content.

R.3.2 Adding a story

Input : User will upload content which he has to share with followers and also he can tag people.
Processing: Website will upload the files in cloud and show their follower's story feed and send message to tagged people.
Output : Website will show that story is successfully uploaded.

R.4 Chat feature

Description : Users can chat with other users by sharing text messages, images, videos and other user's posts.

Input : User will type message or upload the content.
Processing : Website will sent the messages to other user.
Output : user can see that message is send in chat section.

R.5 Manipulating posts

Description : Users can like, share and comment on other user's posts.

R.5.1 Like the post

Input : User can click on the like button or double click to like the post.
Processing : Website will show to user that other user has liked his post.
Output : Show that post is liked.

R.5.1 comment on post

Input : Users can click on the comment button and enter their comment in the comment section.
Processing : Website will show to user that other user has commented on his post.
Output : Show that user's comment on other user's comment section.

R.5.1 Share the post

Input : Users can click on share button to share the post to other users.
Processing : Website will send the shared post to other user in chat section.
Output : User can see that post is shared in chat.

R.6 Update Profile

Description : Users can change or update their personal details.

Input : User will click in update profile and make changes that he want to change in profile and save changes.
Processing : Website will update user's profile by updating database.
Output : User's profile changes will be reflected in the user's profile.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

○ The system should have a response time of less than 2 for all user Requests.
○ The system should be able to process at least 100 users at a time.

5.2 Safety Requirements

○ The system should clearly indicate to the user any potential risks associated with certain actions (such as exceeding the maximum order quantity of a product
○ The system should validate user input and prevent any data injection attacks.
○ The system should have a recovery process for handling errors and Exceptions.

5.3 Security Requirements

○ The system should use encryption to protect sensitive data, such as user passwords.
○ The system should have regular vulnerability assessments and penetration tests to identify and remediate any security weaknesses.

5.4 Software Quality Attributes

○ The system should be highly available, with an uptime of at 99.5%.
○ The system should be secure and protect against unauthorized access and data breaches.
○ The system should be scalable and able to handle an increase in traffic and data storage.

5.5 Design Requirements

○ The system should have a user-friendly interface that is easy to navigate and understand.

5.6 Business Rules

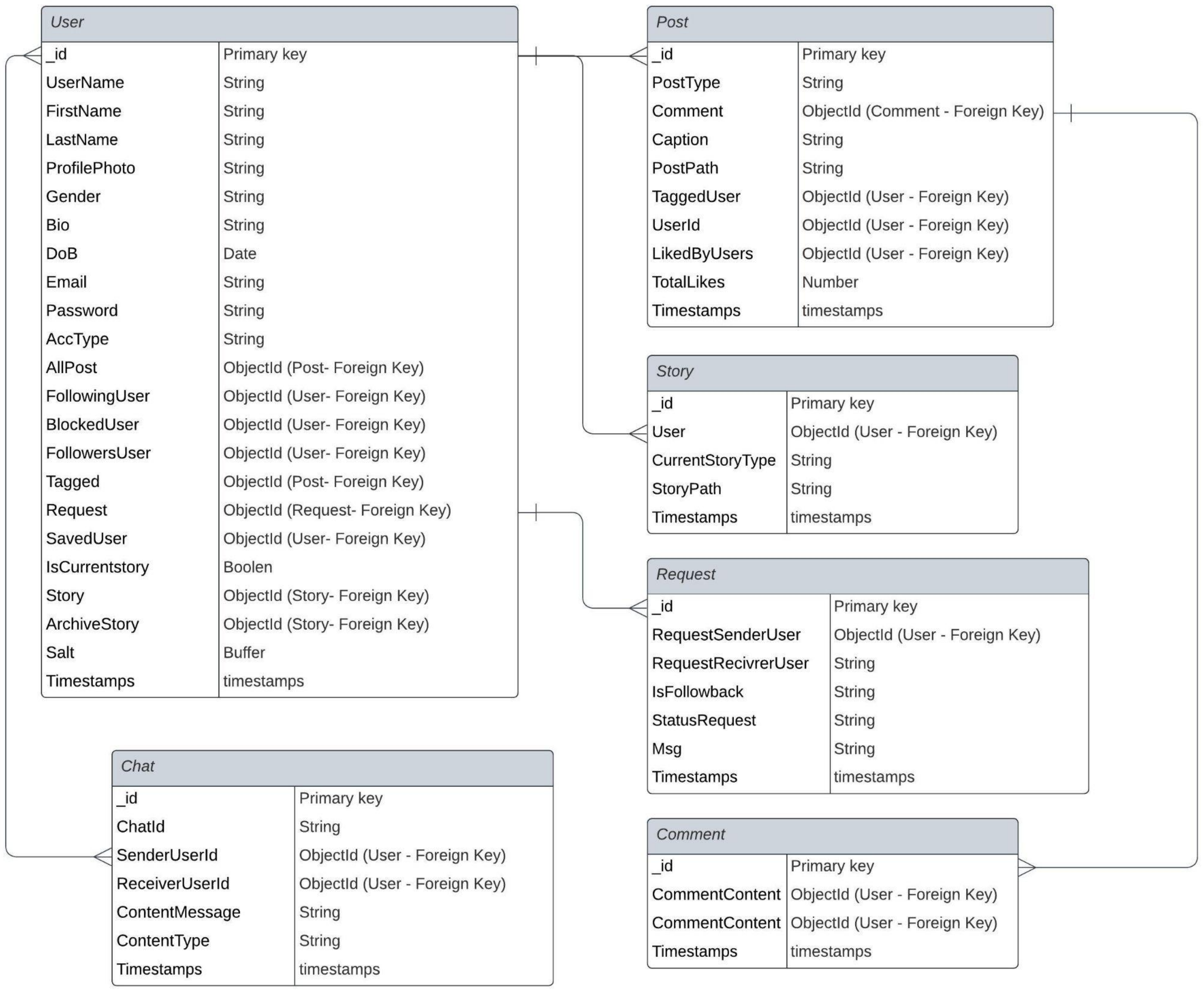
○ Content Moderation: Certain types of content should be monitored and moderated to ensure compliance with community guidelines and policies.

○ User Age Limit: Users should be required to be of a certain age (e.g., 13 years or older) to use the platform in compliance with legal requirements.

○ Prohibited Content: The application should restrict the posting of offensive, illegal, or harmful content.

○ Intellectual Property Rights: Users should not be allowed to post copyrighted or unauthorized content without appropriate permissions.

Database Design



User	
_id	Primary key
UserName	String
FirstName	String
LastName	String
ProfilePhoto	String
Gender	String
Bio	String
DoB	Date
Email	String
Password	String
AccType	String
AllPost	ObjectId (Post- Foreign Key)
FollowingUser	ObjectId (User- Foreign Key)
BlockedUser	ObjectId (User- Foreign Key)
FollowersUser	ObjectId (User- Foreign Key)
Tagged	ObjectId (Post- Foreign Key)
Request	ObjectId (Request- Foreign Key)
SavedUser	ObjectId (User- Foreign Key)
IsCurrentstory	Boolen
Story	ObjectId (Story- Foreign Key)
ArchiveStory	ObjectId (Story- Foreign Key)
Salt	Buffer
Timestamps	timestamps

Post	
_id	Primary key
PostType	String
Comment	ObjectId (Comment - Foreign Key)
Caption	String
PostPath	String
TaggedUser	ObjectId (User - Foreign Key)
UserId	ObjectId (User - Foreign Key)
LikedByUsers	ObjectId (User - Foreign Key)
TotalLikes	Number
Timestamps	timestamps

Story	
_id	Primary key
User	ObjectId (User - Foreign Key)
CurrentStoryType	String
StoryPath	String
Timestamps	timestamps

Request	
_id	Primary key
RequestSenderUser	ObjectId (User - Foreign Key)
RequestRecivrerUser	String
IsFollowback	String
StatusRequest	String
Msg	String
Timestamps	timestamps

Chat	
_id	Primary key
ChatId	String
SenderUserId	ObjectId (User - Foreign Key)
ReceiverUserId	ObjectId (User - Foreign Key)
ContentMessage	String
ContentType	String
Timestamps	timestamps

Comment	
_id	Primary key
CommentContent	ObjectId (User - Foreign Key)
CommentContent	ObjectId (User - Foreign Key)
Timestamps	timestamps

Implementation Detail

i) Modules created and brief description of each modules.

1) **User Model :** User model is created for users which are using application. In which user's details are store and it is containing ObjectId of Post, Story, Request and Chat.

2) **Chat Model :** Chat model is created for storing chat of the user and which have chatId which is made of sender and receiver's Id for finding chat between users.

3) **Post Model :** Post model is for storing the post and details of the post. Which have ObjectId of comment model and which containing comments of post.

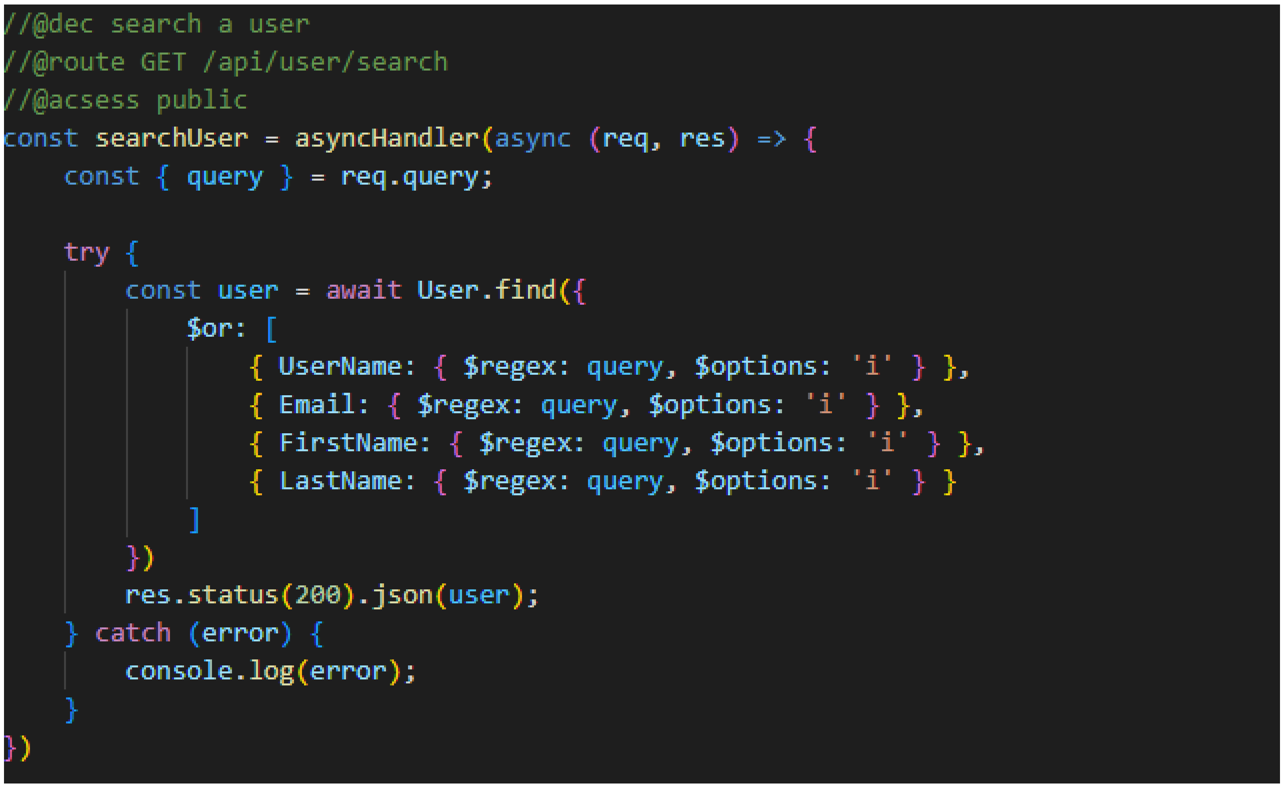
4) **Comment Model :** Comment model is containing comment for specific Post of the user.

5) **Story Model :** Story model is containing Story of the current user. And api will following the user's story which was created 24h ago.

6) **Request Model :** Request Model contains the following request from another user.

ii) Function prototypes which implements major functionality.

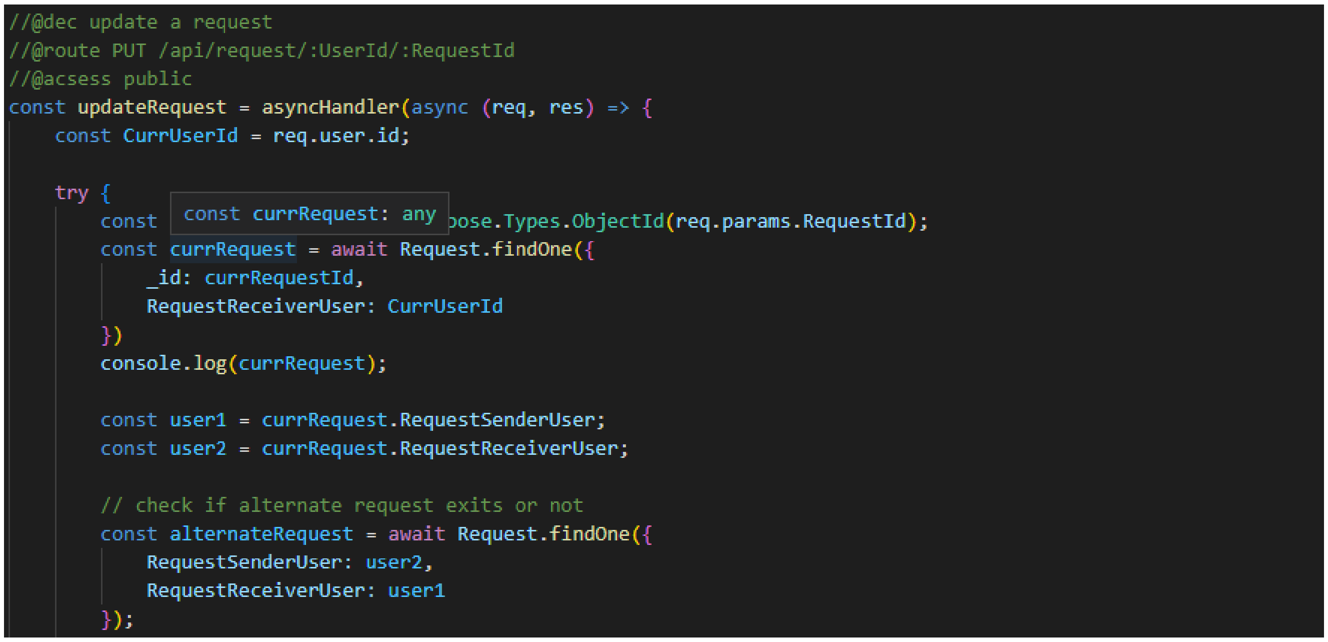
1) Search User:



```
//@dec search a user
//@route GET /api/user/search
//@acsess public
const searchUser = asyncHandler(async (req, res) => {
    const { query } = req.query;

    try {
        const user = await User.find({
            $or: [
                { UserName: { $regex: query, $options: 'i' } },
                { Email: { $regex: query, $options: 'i' } },
                { FirstName: { $regex: query, $options: 'i' } },
                { LastName: { $regex: query, $options: 'i' } }
            ]
        })
        res.status(200).json(user);
    } catch (error) {
        console.log(error);
    }
})
```

2) Update Request

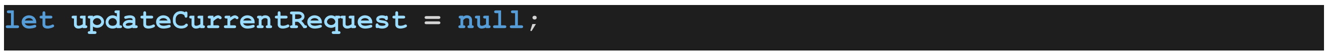


```
//@dec update a request
//@route PUT /api/request/:UserId/:RequestId
//@acsess public
const updateRequest = asyncHandler(async (req, res) => {
    const CurrUserId = req.user.id;

    try {
        const  const currRequest: any  ose.Types.ObjectId(req.params.RequestId);
        const currRequest = await Request.findOne({
            _id: currRequestId,
            RequestReceiverUser: CurrUserId
        })
        console.log(currRequest);

        const user1 = currRequest.RequestSenderUser;
        const user2 = currRequest.RequestReceiverUser;

        // check if alternate request exits or not
        const alternateRequest = await Request.findOne({
            RequestSenderUser: user2,
            RequestReceiverUser: user1
        });
```



```
let updateCurrentRequest = null;
```

```
        if (req.body.StatusRequest == "Accepted") //if it is requeset accepted
then do this
        {
            if (alternateRequest) {
                const updateAlternanteId = alternateRequest._id;
                const updateValues = {
                    StatusRequest: "Accepted",
                    IsFollowback: true,
                }
                let updateCurrentRequest = await
Request.findByIdAndUpdate(currRequestId, { ...updateValues, Msg: `${user1}
starts follow You` });

                const updateAlternanteRequest = await
Request.findByIdAndUpdate(updateAlternanteId, { ...updateValues, Msg:
`${user2} starts follow You` });
            }
            else {
                const updateValues = {
                    StatusRequest: "Accepted",
                    Msg: `${user1} starts follow You`,
                    IsFollowback: false,
                }
                let updateCurrentRequest = await
Request.findByIdAndUpdate(currRequestId, updateValues);
            }

            const updateUser1 = await User.findOneAndUpdate(user1,
                {
                    $push: { FollowingUser: user2 }
                },
                { new: true }
            );

            const updateUser2 = await User.findOneAndUpdate(user2,
                {
                    $push: { FollowersUser: user1 }
                },
                { new: true }
            );
            res.status(200).json(updateCurrentRequest);
        }
        else {
            const request1 = await Request.findById(req.params.RequestId);
            if (!request1) {
                res.status(400).json({ msg: "Request not found." });
            }
            let updateCurrentRequest = await
Request.findByIdAndRemove(req.params.RequestId);
            res.status(200).json(updateCurrentRequest);
        }
    }
    catch (err) {
        console.log(err);
        return res.status(500).json({ msg: err.message });
    }
});
```

3) Like post

```
//@dec like a post
//@route PUT /api/post/like/:PostId
//@acsess public
const likePost = asyncHandler(async (req, res) => {
    const CurrUserId = req.user.id;

    try {
        const user = await User.findById(CurrUserId);
        if (!user) {
            res.status(404).json({ msg: "User not found." });
        }

        const post1 = await
Post.findById(req.params.PostId).populate("Comment").populate("UserId");
        if (!post1) {
            res.status(404).json({ msg: "Post not found." });
            throw new Error("Post not found.");
        }

        // Check if the user has already liked the post
        if (post1.LikedByUsers.includes(CurrUserId)) {
            post1.LikedByUsers.pull(CurrUserId);
            //res.status(400).json({ msg: "User has already liked this post."
});
            //throw new Error("User has already liked this post.");
        }
        else{
            post1.LikedByUsers.push(CurrUserId);
        }

        // Add the user's ID to the LikedByUsers array
        //post1.LikedByUsers.push(CurrUserId);

        // Update the TotalLikes count
        post1.TotalLikes = post1.LikedByUsers.length;

        // Save the updated post
        const post2 = await post1.save();
        console.log(user);
        console.log(post2);
        res.status(200).json(post2);

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

4) Comment on Post

```
//@dec comment in post
//@route DELETE /api/post/comment/:PostId
//@acsess public
const commentPost = asyncHandler(async (req, res) => {
    const CurrUserId = req.user.id;

    try {
        console.log("The request body is : ", req.body);
        const { Commentcontent } = req.body;

        if (!Commentcontent) {
            res.status(400).json({ msg: "All fields are required." });
            throw new Error("All fields are required.");
        }

        const user = await User.findById(CurrUserId);
        if (!user) {
            res.status(404).json({ msg: "User not found." });;;
            throw new Error("User not found.");
        }

        const post1 = await Post.findById(req.params.PostId);
        if (!post1) {
            res.status(404).json({ msg: "Post not found." });;;
            throw new Error("Post not found.");
        }

        const comment1 = await Comment.create({
            CommentContent: Commentcontent,
            CommentedBy: CurrUserId,
        });

        const post2 = await Post.findByIdAndUpdate(req.params.PostId,
            {
                $push: { Comment: comment1._id }, // Add the new postid to the
AllPost array
            },
            { new: true }
        );
        console.log(post2);
        const newComment = await
Comment.findById(comment1._id).populate("CommentedBy");
        res.status(200).json(newComment);

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

});
```

5) Get Chats of User

```
//@dec get a chat
```

```
//@route GET /api/chat/getAllUserchats
//@acsess public
const getAllUserChats = asyncHandler( async (req,res) => {
    const CurrUserId = req.user.id;
    // console.log(CurrUserId);
    try {
        const CurrUser = await
User.findById(CurrUserId).populate("FollowingUser").populate("FollowersUser");

        const FollowingUser = CurrUser.FollowingUser;
        const FollowersUser = CurrUser.FollowersUser;
        let UserChatsData = [];

        FollowingUser.forEach(currUser => {
            const {UserName, ProfilePhoto, id} = currUser;
            UserChatsData.push({UserName, ProfilePhoto, UserID:id})
        });

        FollowersUser.forEach(currUser => {
            const {UserName, ProfilePhoto, id} = currUser;
            UserChatsData.push({UserName, ProfilePhoto, UserID:id})
        });

        let FilterUserChatsData = [];
        const uniqueKeys = new Set();

        for (const obj of UserChatsData) {
            const key = obj.UserID;
            if (!uniqueKeys.has(key)) {
              uniqueKeys.add(key);
              FilterUserChatsData.push(obj);
            }
          }
        res.status(200).json(FilterUserChatsData);
    } catch (error) {
        console.log(error);
        res.status(404).json(error);
    }
});
```

Testing

For testing our application, a mixed approach of integration testing and regression testing is used.

Integration testing : each main part of the application is tested after each small small function is tested first and then combine them and test that main part.

Regression testing : After the main part of the application is created, add them in the whole system, and then test the whole system to make sure the whole system is working fine after adding some main part.

Manual Testing:- Manual testing is used to find and fix the bug in our application.

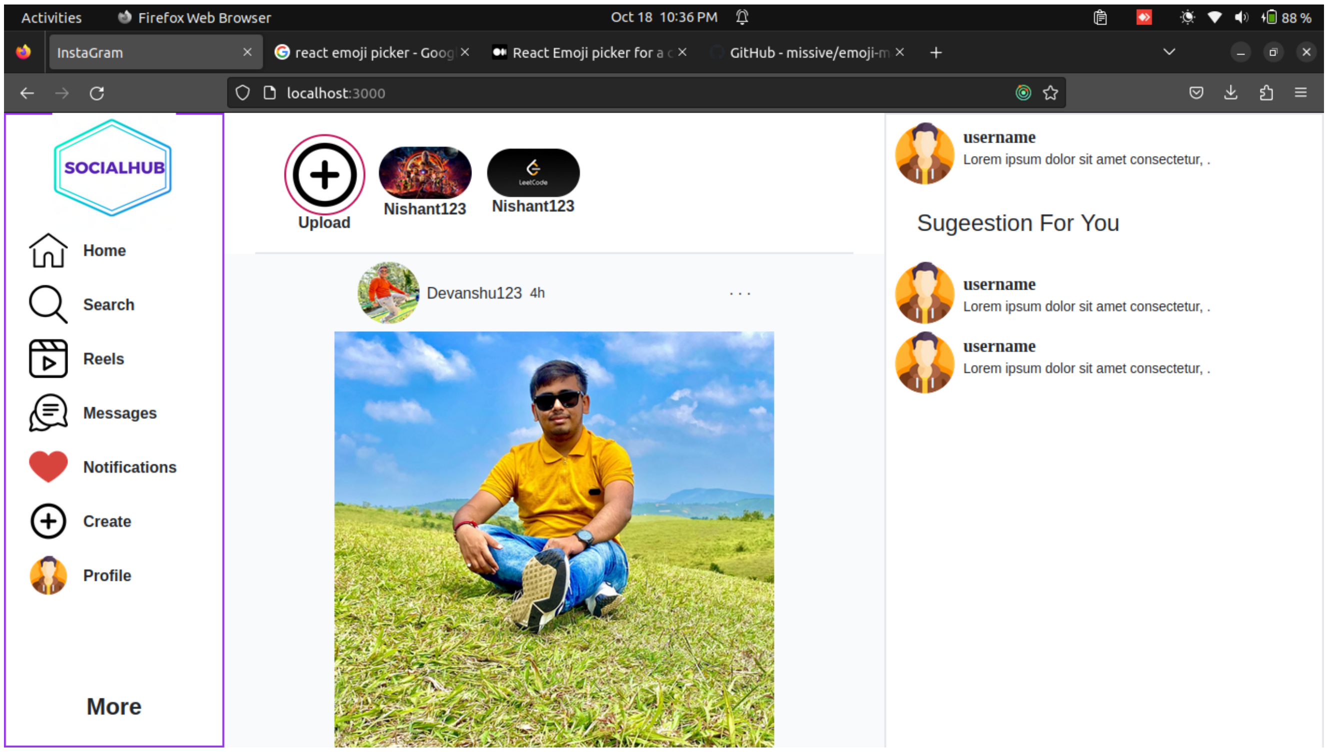
Test Cases :

Sr No.	**Test Scenario**	**Expected Result**	**Actual Result**	**Status**
1	Login	The user can login into the application.	The user is able to login	Success
2	Register / Sign Up	The user can register / sign up into the application.	The user is registered / signed up into the application.	Success
3	Register / Sign Up with already exist user	The user can not register / sign up into the application with the same email.	The user is not registered / signed up into the application with the same email.	Success

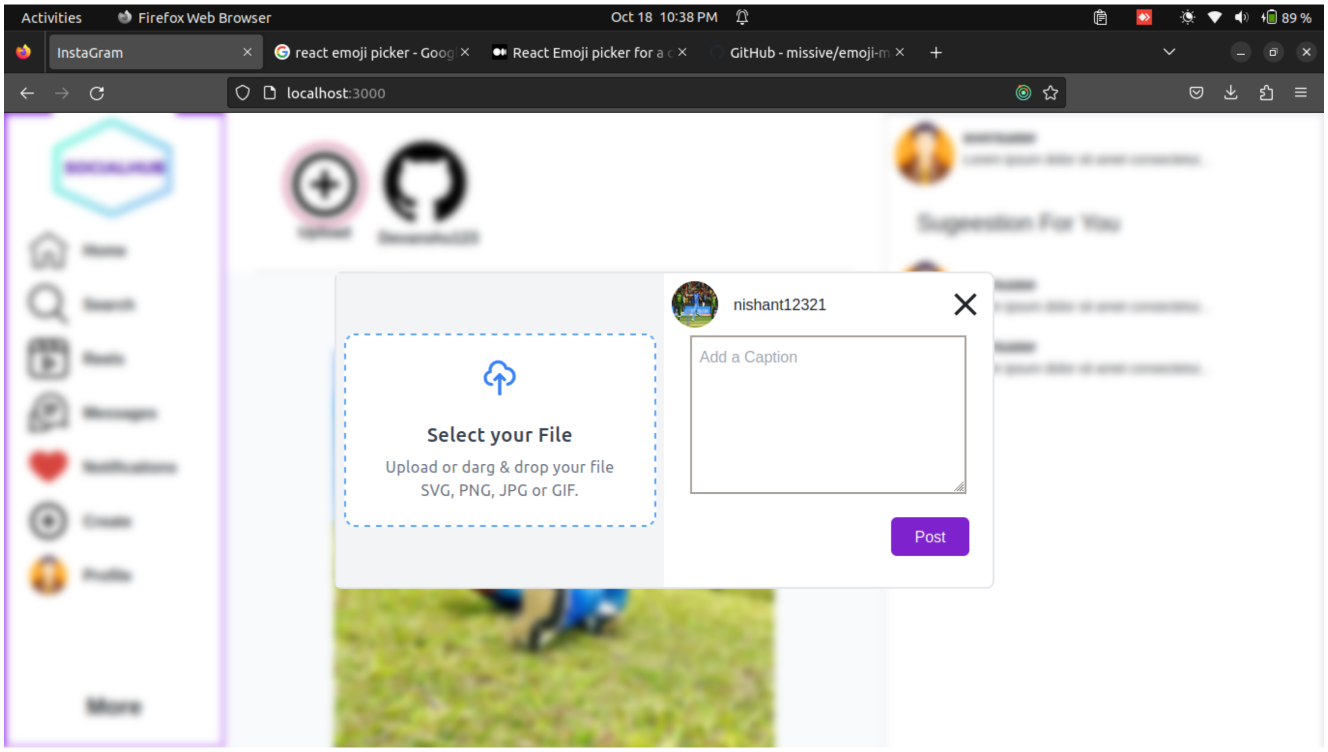
4	Add Post	The user can add post into the application and show in profile.	The user has added a post into the application and shown it in their profile.	Success
5	Like Post	The user can like the post, increase like and show in post.	The user was like the post, increased the like and showed it in the post.	Success
6	Comment in Post	The user can comment on the post and show in post.	The user commented on the post and showed it in the post.	Success
7	Chat with another user	The user can chat with another user and another user should get a chat(message).	The user has a chat with another user and another user should get a chat(message).	Success
8	Search user	The user can search other users.	The user should get searched users.	Success

Screen-Shots:

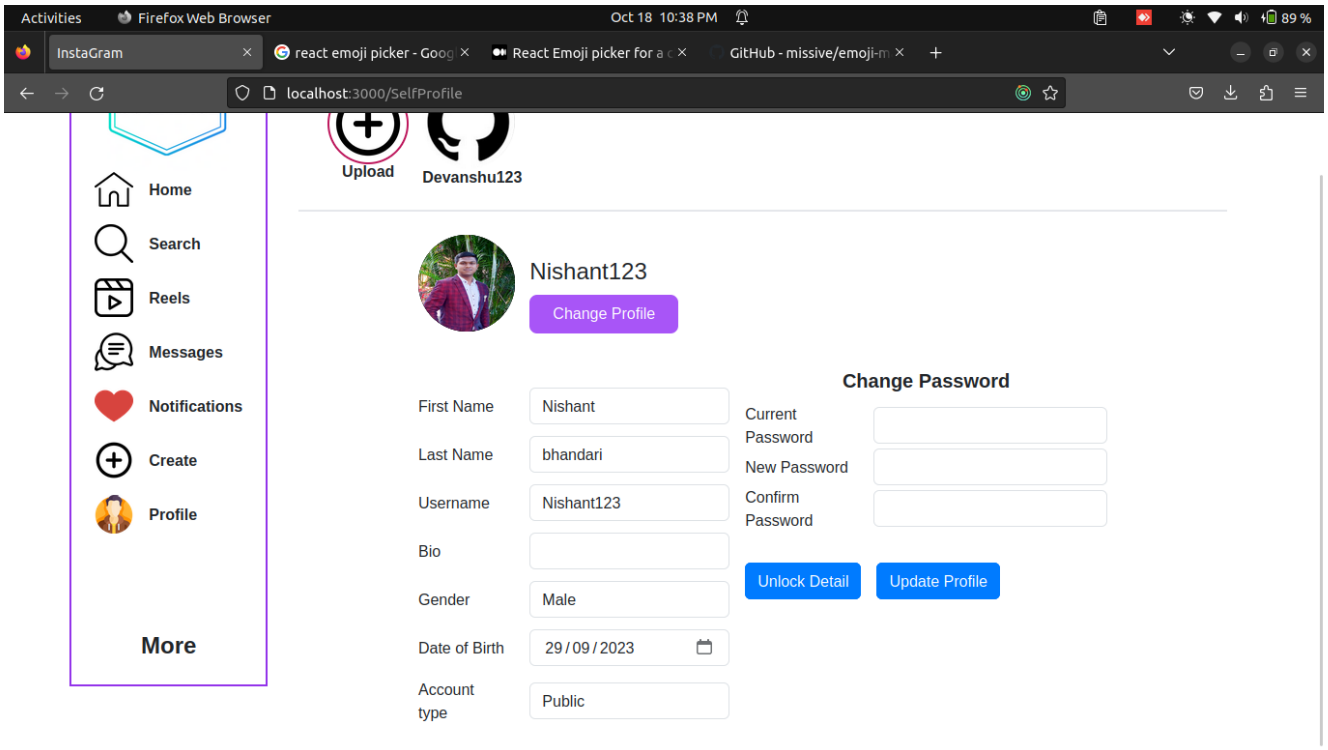
HomePage



PostUploadPage



ProfilePage


Activities Firefox Web Browser Oct 18 10:38 PM 89 %

InstaGram | react emoji picker - Goog | React Emoji picker for a | GitHub - missive/emoji-m

localhost:3000/SelfProfile

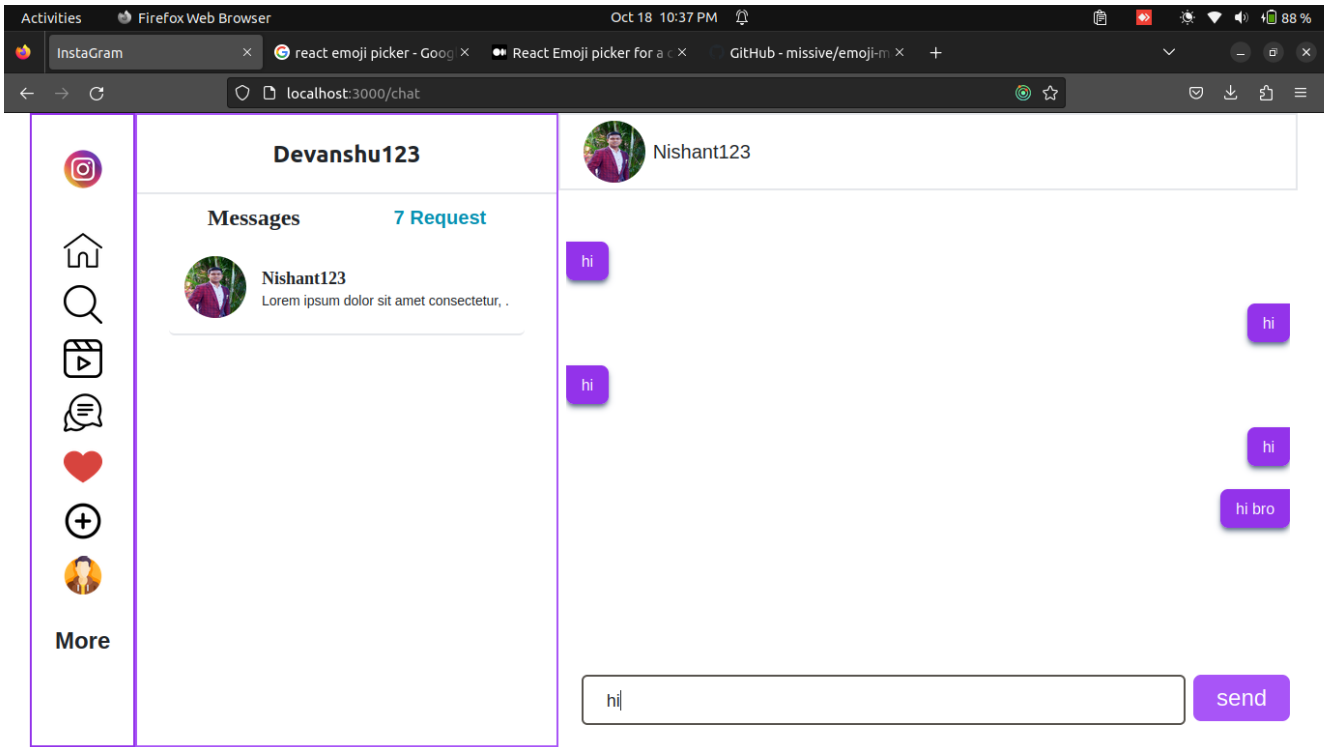
Nishant123

First Name	Nishant
Last Name	bhandari
Username	Nishant123
Bio	
Gender	Male
Date of Birth	29/09/2023
Account type	Public

Change Password

Current Password	
New Password	
Confirm Password	

ChatPage


Activities Firefox Web Browser Oct 18 10:37 PM 88 %

InstaGram | react emoji picker - Goog | React Emoji picker for a | GitHub - missive/emoji-m

localhost:3000/chat

Devanshu123

Nishant123
Lorem ipsum dolor sit amet consectetur, .

Nishant123

hi

hi

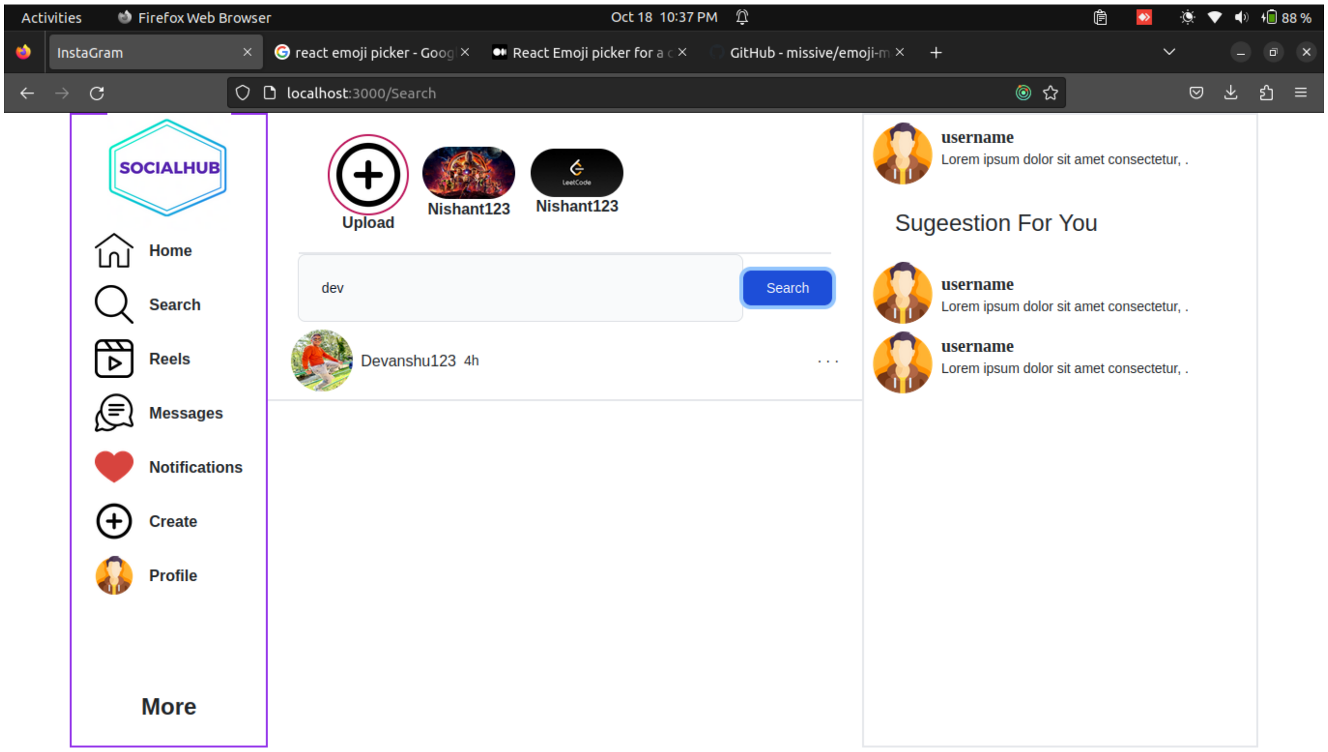
hi

hi

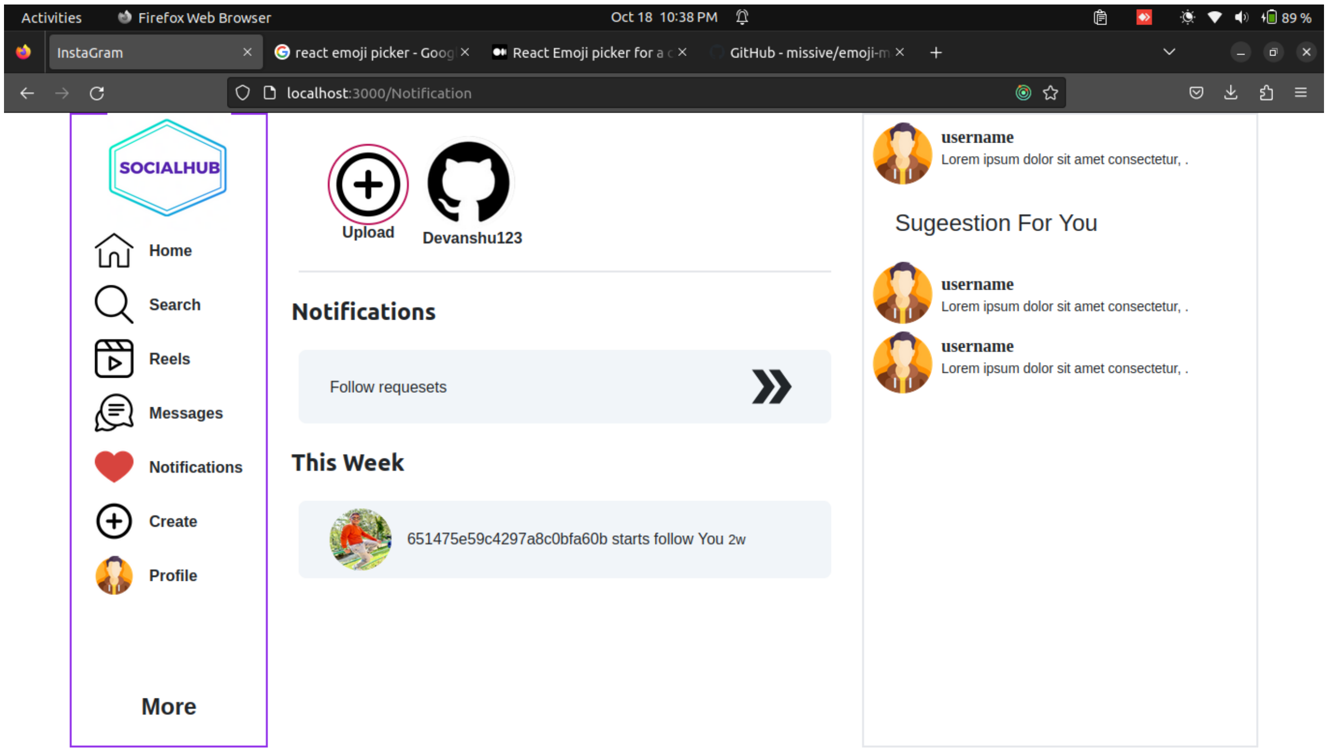
hi bro

hi

SearchPage



NotificationPage



- Describe the functionality successfully implemented by you precisely and in brief
- Don't write philosophical statements

12) Limitation and Future Extension

- **Only share images as posts, not videos**
- **The Reels feature is not available.**
- **Add Emoji picker in chatInput**

13) Bibliography

W3Schools

https://react.dev/

https://www.mongodb.com/

https://expressjs.com/

https://www.youtube.com/