Social Media App (Twitter clone)

Important Note: Please add comments while writing codes and a concept note in a separate sheet while submitting the assignment. If comments and concept notes are not present in your submission, marks will get deducted.

Hello everyone, welcome to your 2nd capstone project. Hope you had a great time learning from our course and are ready to apply what you've learned in creating real-world applications.

This project is meant to test your full-stack (MERN) development skills. You'll be applying what you've learned so far in this course to create a Social Media application. Don't worry! You have the skills to do this already, and you'll be given detailed steps, with hints, to help you every step of the way. ��

For this entire application, you'll be required to create a full-blown Social media application with both the front end (User Interface, UI) and back end (security, database and APIs).

This application is a clone of Twitter and after creating this application you can pretty much create any social media app.

Flow of creating the application - First we will build the Backend and then once we have all of the APIs ready we will create the front end.

Following are the Features included in the App.

1. Authentication using JWT token, i.e user login, and registration.
2. Create and Delete a Tweet.
3. Add an Image in a Tweet.
4. Like and Dislike a Tweet.
5. Add a reply to a tweet
6. Retweet

7. Read all the tweets in app
8. Create and Edit a User.
9. Follow and Unfollow User.
10. View a user profile details
11. Update Profile picture including uploading images using multer.

Note: In order to see how the complete website should work, please click here to watch demo video.

So let's get started with Part 1 - Backend

Part 1 – Backend

Tech Stack Used

Node.js (main backend language)
- Express.js (framework based on nodejs to create REST APIs)
- Mongoose (to connect MongoDB with express.js)
- bcrypt (to encrypt password when stored in DB)
- multer (to upload images)

Problem Statement

Create a backend app which will contain a set of REST APIs to store and get data from MongoDB which will help frontend interact and store user information. The app will have three entities and we will build REST API's around all three entities to perform all the required functionalities.

1. Auth
2. User
3. Tweet

Details on Functionalities to build

1. **Setup project and create a server on PORT**

   - Start by creating a Nodejs project and installing ExpressJs init.

   - create an index.js file and listen to the express server on any port.

2. **Connect app with MongoDB**

   - install mongoose and connect the app with either local MongoDB server or the online MongoDB atlas.

   - anyone will work, it's just the backend should connect to MongoDB. - you can do console.log("DB connected"), when MongoDB is connected to indicate and confirm that DB is connected successfully.

3. **Creating User schema**

   - Before implementing user related APIs let's first create a User MongoDB schema to define how the data will look like.

   - Below are the fields included in the User schema.

Field name	Data type	Additional attributes	Comments
Name	String	This field is required	This field will store a user full name Eg. John Doe
Username	String	This field is required and also the username should be unique.	In the app every user will have a unique username Eg. john-doe In the UI we will display username as @john-doe (@ in the front)
Email	String	This field is required and also the username should be unique.	This field will store a users email id. Eg. john@email.com

Password	String	This field is required	This field will store the user account password. Also the password stored in the DB should be encrypted. eg. actual password - john Encrypted password stored in DB - `$2a$10$HaK9HroNq1429ckGePQFNeL9O1fYU7Eb MFVDKHZC9/xT.7f4mjApm`
Profile Picture	String	can be optional	This field will store the location of the profile pic of a user. Eg. profile-pic-afsdhdf.jpg
Location	String	can be optional	This field will store location of a user Eg. Maharashtra, India
Date of birth	Date	can be optional	This field will store user DOB Eg. Actual readable Date - 29/12/2022 Date stored in DB `2021-04-22T16:56:53.152+00:00` Note: mongoose will auto convert date and store it in DB. formatting the date and deciding how to display it in user redable format should be handled from UI side.
Followers	Array of ObjectId ref : User	This field will reference to User schema. Because we will need to populate data of all the users who follow to show in the UI.	This field will store user followers data. Eg. followers: [“63a07827b36f743f88c20ee6 1”, “63a051d831f37a08f9dfb3d6”] Basically we will store _id of user who follow this user. The benefit of storing _id of user and referencing to User schema is that we can populate the user data using .populate method in mongoose

following	Array of ObjectId ref: User	This field will reference to User schema. Because we will need to populate data of all the users who follow to show in the UI.	This field will store user following data Eg. same as followers

Note: Make sure to use the { timestamp: true } feature of mongoose which will store createdAt and updatedAt dates of every document. ref here - https://mongoosejs.com/docs/timestamps.html

Eg of how a typical User document will look like

4. Creating Tweet Schema

- Before implementing user related APIs let's first create a Tweet MongoDB schema to define how the data will look like.
- Below are the fields included in the Tweet schema.

Field name	Data type	Additional attributes	Comments
Content	String	This field is required	This field will store a Tweet text content Eg. How are you doing football fans?
TweetedBy	ObjectId ref: User	This field will store the _id of user who tweeted	This field will store _id of user who tweeted. Eg. tweetedBy: “63a07827b36f743f88c20ee6” The benefit of storing _id of user and referencing to User schema is that we can populate the user data using .populate method in mongoose. And in UI we can show user name, profilePic while showing the tweet.
Likes	Array of ObjectId ref: User	This field will store list of users who liked the tweet	Eg. likes: [“63a2e78acaa0bb5a5c1d90dd”, “63a2e78acaa0bb5a5c1d90d3”]
RetweetBy	Array of type Users	This field will store list of user data which have retweeted this tweet	
Image	String	location of image uploaded in a tweet	this field is optional, user can include a image in the tweet but it is not compulsory
Replies	Array of type Tweet	This field will store list of Tweet as a reply	in tweeter every reply in a tweet is a tweet itself on which you can perform all the operations

Note: make sure to use the { timestamp: true } feature of mongoose which will store createdAt and updatedAt dates of every document. ref here - https://mongoosejs.com/docs/timestamps.html

Eg of how a typical Tweet document will look like


```
{
"_id": "63a3d63bf503d9e51cb3656b",
"content": "How are you doing football fans!!",
"tweetedBy": "63a051d831f37a08f9dfb3d6",
"likes": ["63a051d831f37a08f9dfb3d6", "63a051d831f37a08f9dfb32fr"],
"comments": [
  {
    "content": "Very good",
    "commentedBy": "63a051d831f37a08f9dfb3d6",
    "commentedAt": "2022-12-22T04:00:20.573Z",
    "_id": "63a3d654f503d9e51cb36597"
  },
  {
    "content": "good morning all\n",
    "commentedBy": "63a16d5f2d8abfc7ef627a46",
    "commentedAt": "2023-01-26T21:03:16.337Z",
    "_id": "63d2ea942ae5bb75f23a1cb7"
  }
],
"createdAt": "2022-12-22T03:59:55.104Z",
"updatedAt": "2023-01-26T22:57:34.222Z",
"retweetBy": ["3a0sf51d831f37a08f9dfb3d6", "3a051d831f37a08f9dfb3d7"],
"replies": ["3a0sf51d831f37a08f9dfb3d6", "3a051d831f37a08f9dfb3d7"]
}
```


Creating APIs

Note :

- Once API is created verify the working using Postman tool
- Refer to Instagram clone login and register flow if anything here doesn't make sense or you are stuck somewhere.

5. **Creating Auth-related APIs**

- The prefix for auth API should be /API and then every route will follow the REST API suffix.

  Eg. the register route in auth will be /auth/register and the complete route with /API at the front will be API/auth/register
- In auth entity we will have two API's/auth/register and /auth/login. register API will create a new user with input from the user and login will generate a jwt token and will help user to authenticate and login.

a. Creating /auth/register API

- Endpoint - POST /auth/register

- Below are the details required in register API

API example	validation	Success criteria
This api will accept name, email, username and password from user eg POST http://localhost:9000/api/auth/register req.body `{ "name": "John Doe", "email": "john@email.com", "username": "john", "password": "johndoe" }`	1. Validate if all the fields are sentx from the user. if not then send proper error message to response 2. Verify if the email is unique and doesn't exist in DB. if the email exist then send proper error message to response 3. Verify if the username is unique and doesn't exist in DB. if the email exist then send proper error message to response	If everything is good create a new User and save the data in DB

b. Creating /auth/login API

- Endpoint - POST /auth/login
- Below are the details required in login api

c. Creating a middleware to verify if the API is called only when the user is logged in.

- When the user is logged in, store the token and use that token when calling any API
- You can send the token in cookie or also can send the token in the API auth header. both will work
- When an API call is made, before calling the router function, add a middleware that will verify if the API has a valid token.
- Verify that token with jwt and once verified store user data in req.user object so that in the router/controller function we can refer to the logged in user and get useful info.
- If the token is valid then execute next() function otherwise return proper auth error in response.

Note: if this seems very complicated then you can also refer to "Putting it all together: Creating a Web App- Part 2 "-> authentication section. for reference.

6. Creating User related api's

- Below are the functionalities we will cover in user API's
    a. Get a single user detail
    b. Follow user
    c. Unfollow user
    d. Edit user details
    e. Get user tweets
    f. Upload user profile picture

a. Get a single user details

Endpoint - GET /api/user/:id

- find the user and send complete user data
- populate following and followers data

Note: Make sure to not send user's password in the api response anywhere.

Example of API	Success criteria
This API will return user data Note: following and followers data are not present in the example but make sure you populate the following and followers data eg GET http://localhost:9000/api/user/63a16d5f2d8abfc7Ef627a46 Response: 	Make sure you don't send the password in the response User data should include complete data

b. Follow user

- Endpoint POST /api/user/:id/follow
- In the logged in users following array add the user to follow id and in the user to follow followers array add logged in user's id
- The reason is let's say if I follow you then in my following array I'll save your id and in your followers array will add my id
- So my 1 following will increase and your 1 follower will increase.
- Save both users and user to follow data

API call	Response
PUT http://localhost:9000/api/user/63a07827b36f743f88c20ee6/follow	{success: true} with status 200

c. Unfollow user

- Endpoint POST /api/user/:id/unfollow
- Exactly the same logic as follow user API but here the logged-in user we remove the following id and in user to unfollow remove the followers id.

API call	**Response**
PUT http://localhost:9000/api/user/63a07827b36f743f88c20ee7/unfollow	{success: true} with status 200

Note: For follow and unfollow, please make sure:

1. user cannot follow/unfollow himself
2. user cannot follow whom he is already following
3. user cannot unfollow those he is not following at all.

d. Edit user details

Endpoint PUT /api/user/:id/

- Only allow name, date of birth, and location data of the user
- Because other data once created can't be changed eg. username, email - accept the name, date of birth and location data from req.body
- Add the necessary validations
- Finally save the edited user in DB

API eg	Validation

Eg
 	
PUT - http://localhost:3000/api/user/63a16d5f2d8abfc7ef627a46/
 	
req.body
 	
{
"name": "Cristiano Ronaldo",
"location": "India",
"dateOfBirth": "2023-01-03"
}
	Make sure a user cannot edit other user's details
	Only logged in users can edit his profile details

e. Get user tweet

- Endpoint POST /api/user/:id/tweets
- This API will return list of all the tweets tweeted by a user

Api eg
POST - http://localhost:3000/api/user/63a16d5f2d8abfc7ef627a46/tweets

f. Upload user profile picture

- Endpoint POST /api/user/:id/uploadProfilePic
- This is the most interesting part where we will be working with storing images. - this api will take an image and save all the uploaded images in the /images folder - by default expressjs can't handle file upload through api so we will use multer package here

  - ref - https://www.npmjs.com/package/multer

  - this api will accept user image and save the stored image location in DB

  - eg in user - profilePic: {"/images/profile-pic-232bds.jpg"}

  - Note that this api should only accept .jpg , .jpeg, .png files.

Api eg	Validation
POST - http://localhost:3000/api/user/63a16d5f2d8abfc7ef627a46/uploadProfilePic req.body profilePic: (binary) *(this will be file binary so can't be demonstrated in text)	only accept jpg, jpeg, png format

7. **Creating tweet-related APIs**

- Below are the functionalities we will cover in tweet API

    a. Create a tweet

    b. Like a tweet

    c. Dislike a tweet

    d. Reply on a tweet

    e. Get a single tweet details

    f. Get all tweet details

    g. Delete a tweet

    h. Retweet

a. Create a tweet

- Endpoint = POST `/api/tweet`

- A tweet will only have content so accept content in req.body

- Verify if the content is present

- Along with tweet add tweetedBy data with logged in user's _id (use req.user._id that we set when we verify logged in user using middleware)

- Save the tweet into DB

API eg	Note
POST http://localhost:9000/api/tweet/ req.body	A tweet also can have an image (optional) you can take any approach to handle image upload but make sure to add image

{
"content": "twitter is fun"
}
	location when saving a tweet

b. Like a tweet

- Endpoint - POST /api/tweet/:id/like

- add the tweet id to likes array in tweet document

- save the tweet data to DB

API eg	Validation
POST http://localhost:9000/api/tweet/63beada9f4368b45	Make sure we cannot like already liked tweet by that user Tweet can only be liked likes field in Tweet does not have that user id.

c. Dislike a tweet

- Endpoint - POST /api/tweet/:id/dislike

- remove the tweet id from likes array in tweet document

- save the tweet data to DB

API eg	Validation
POST http://localhost:9000/api/tweet/63beada9f4368b45 5aeb0f4c/dislike	Make sure we cannot dislike tweet which is not liked by that user in the first place. A tweet can only be disliked when the likes field in Tweet have that user id.

Note for like / dislike a tweet

- user cannot like an already liked tweet

- user cannot dislike a tweet which he doesn't like initially.

d. Reply on a tweet

Note: In Twitter, a reply is a tweet itself, so a reply can be saved as a new tweet and its id will be saved in the parent tweet replies array

- Endpoint - POST `/api/tweet/:id/reply`
- accept comment content from req.body
- assign tweetedBy data from req.user._id
- save the reply tweet as a new Tweet in the DB
- add the new reply tweet id in the parent reply array

API eg

POST
http://localhost:9000/api/tweet/63beada9f4368b455aeb0f4c/reply
 	
req.body
{
"content": "absolutely"
}

e. Get a single tweet detail

- Endpoint - GET `/api/tweet/:id`
- Populate all the field which has ref in a tweet and send the complete details in response

Note - Hide user password when sending the response

Eg of API endpoint =

GET http://localhost:9000/api/tweet/63beada9f4368b455aeb0f4c

f. Get all tweet details

- Endpoint - GET `/api/tweet/`
- Get all tweets and populate all the field which has ref in tweet and send the complete details in response

Note: a) Hide user password when sending the response

b) Also the tweet you send should be sorted in descending by createdAt field

c) This will help in UI to display the latest posted tweet first

g. Delete a tweet

- Endpoint - DELETE /api/tweet/:id

- This api will delete the tweet from user

Note:

- A tweet can only be deleted by a user who created the tweet
- so add a validation something like if(req.user._id === tweet.tweetedBy) then only delete
- This will check only logged in user who tweeted the tweet can only delete it.
- Now that we have implemented all the required API's lets move to developing the frontend of this app.

h. Retweet

- Endpoint - POST /api/tweet/:id/retweet
- add user id in the retweetBy array of Tweet
- For validation check if the user already retweeted a tweet

Part 2 – Frontend

Tech Stack Used

- Reactjs
- Bootstrap (UI library)
- react-router-dom (routing in UI)
- react-toastify (for notifications)
- axios (call APIs)
- fontawesome (for icons)

Problem Statement

Create a frontend app using react which will help a user to perform all the social media app functionalities.

The user can log in, register, post a tweet, view all the tweets, comment, like, dislike, change his profile, follow and unfollow a user, etc

Below is the detailed list of functionalities required in UI.

Functionalities / screens

- Login page (where user can enter the details and login)
- Register page (where user can enter the register detail and register) -
- Home page (where we list all the tweets)
- Profile details page (show a user profile details)
- Tweet Details Page (show a tweet details and also list all the replies related to that tweet)

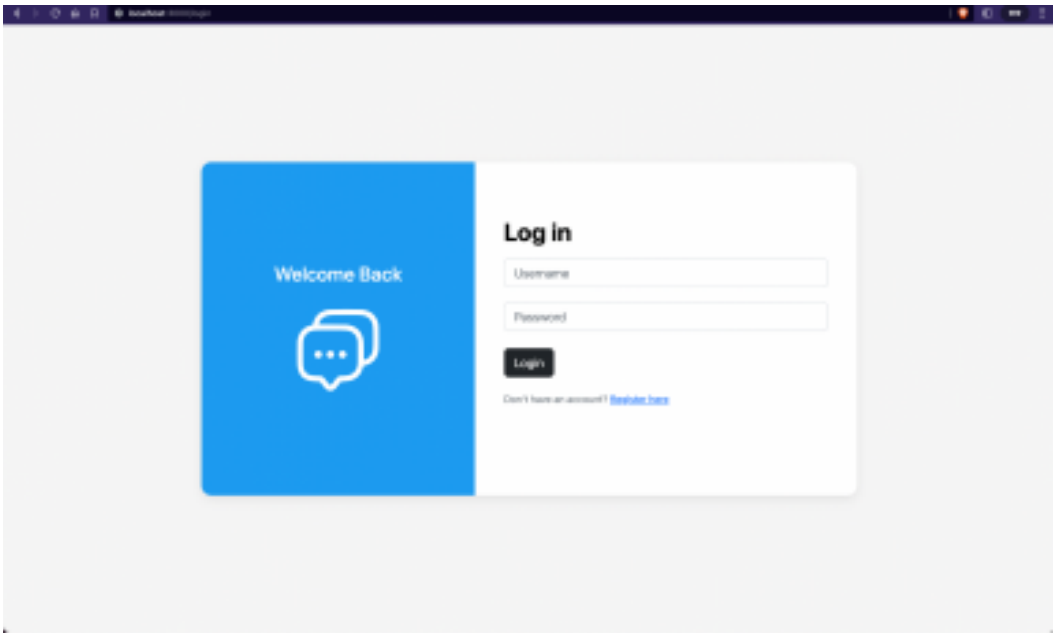
Login Flow

- if the user is not signed in show login page
- once logged in store logged in user details permanently somewhere
- if the user is signed in show home page
- if the user is not registered, then the user will go to register page from the login page and once registered redirect to the login page to login
- once login, take the user to home page
- on logout clear the user data and token from local storage and redirect to login page

Note: maintain a data of logged in user in some permanent storage in browser so that even if user refresh the page we don't logout

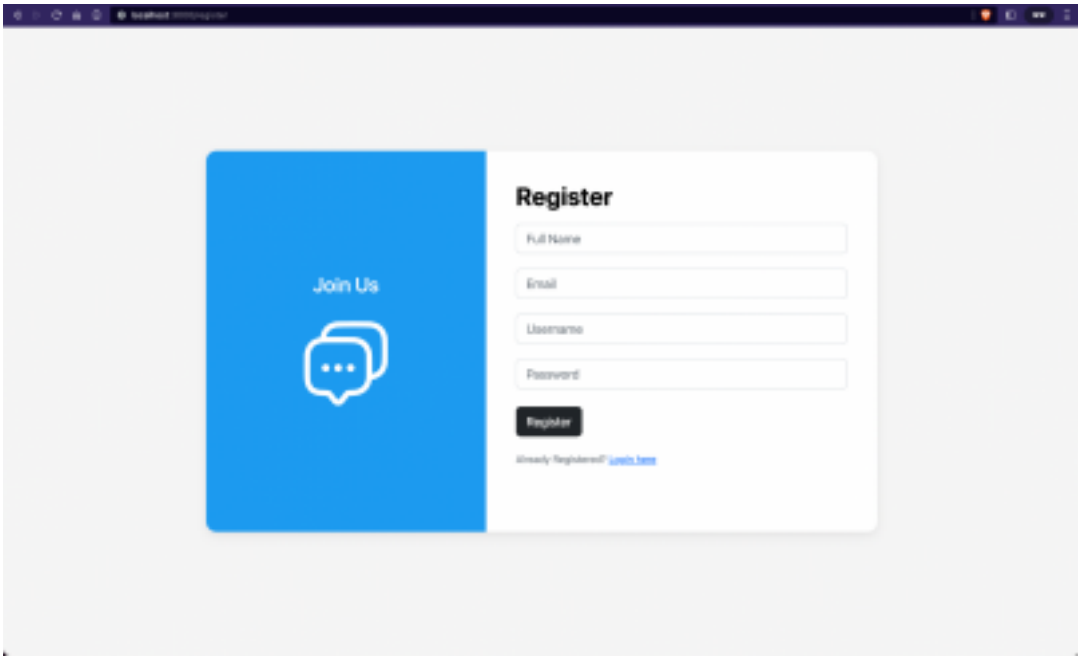
1. **Login Page**

   After user login redirect to app homepage



2. **Register Page**

After user registration redirect to login page to login



3. **Home Page**

- This page will be the main landing page of our app
- the page is divided into two columns sidebar and tweet list

a. **Sidebar**

- Sidebar will contain all the navigation links, app logo and logged in user name

and email

- Clicking on the nav links will get to appropriate page
- We also have a logout button and clicking this will logout the user and redirect to the login page again.

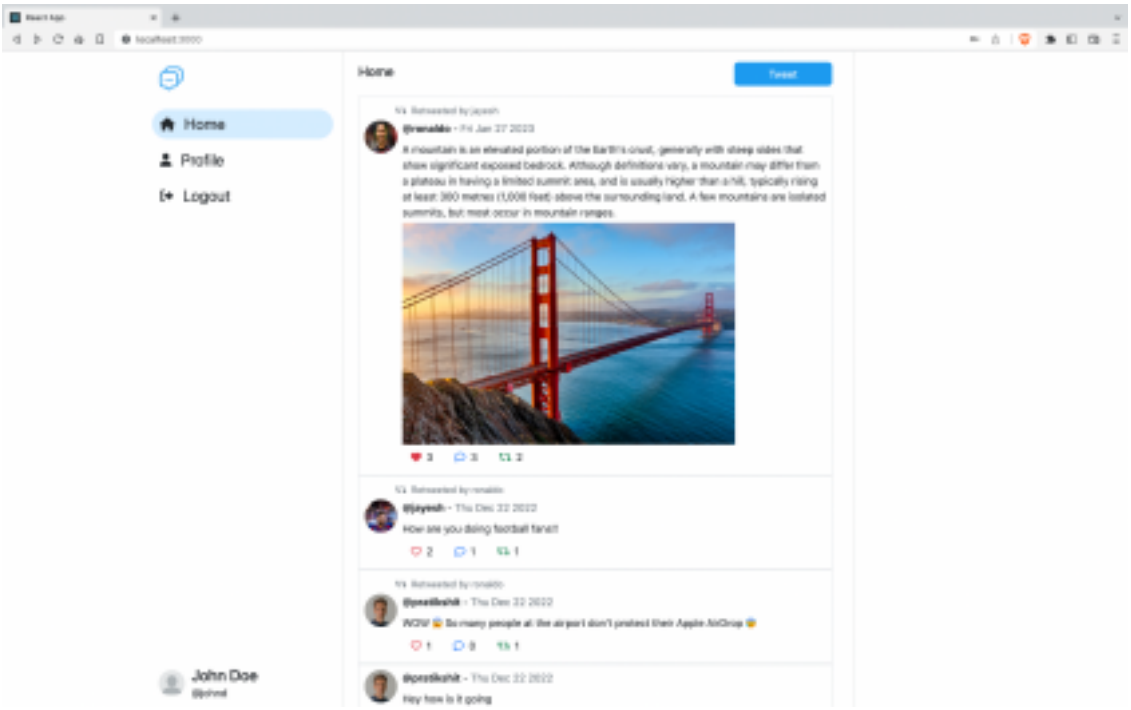
b**. Tweet list**

This section will contain list of tweets where each tweet card will contain the following details:

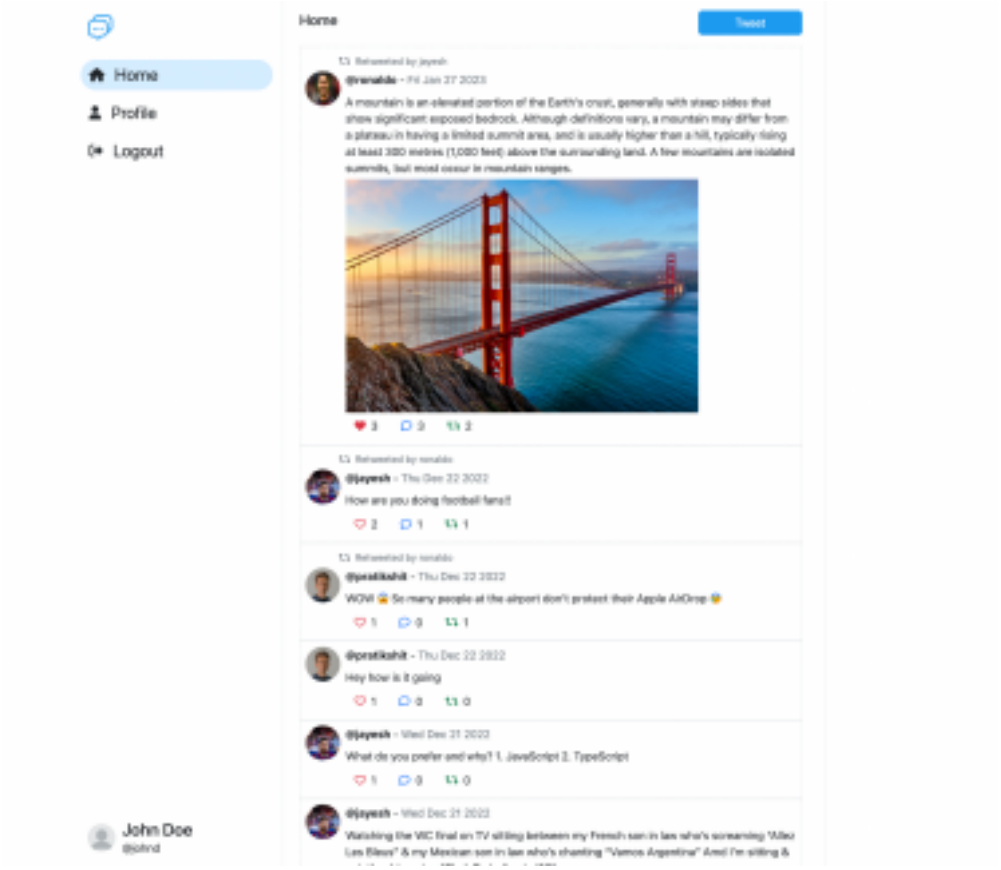
- User photo, user name, tweet time, delete button (only user who created the tweet can only delete), tweet content, like button, retweet button and reply button
- On clicking the like button a tweet will be liked and the icon will turn into a red color.
- On clicking on the reply button a dialog will open where user can enter tweet details and submit
- We will also have a tweet button when clicking on it will open a dialog asking for tweet content and will create a new tweet when submit

Note: Every page in UI will have sidebar section as common

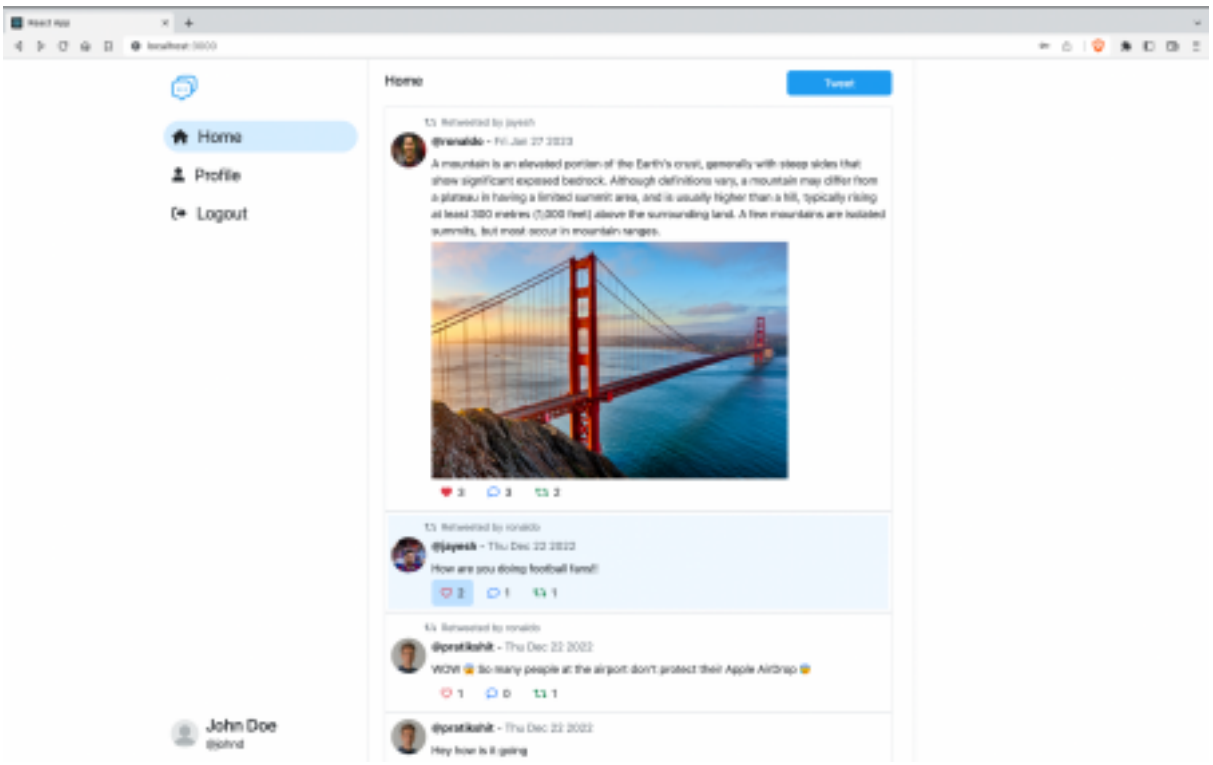
Screenshot of homepage



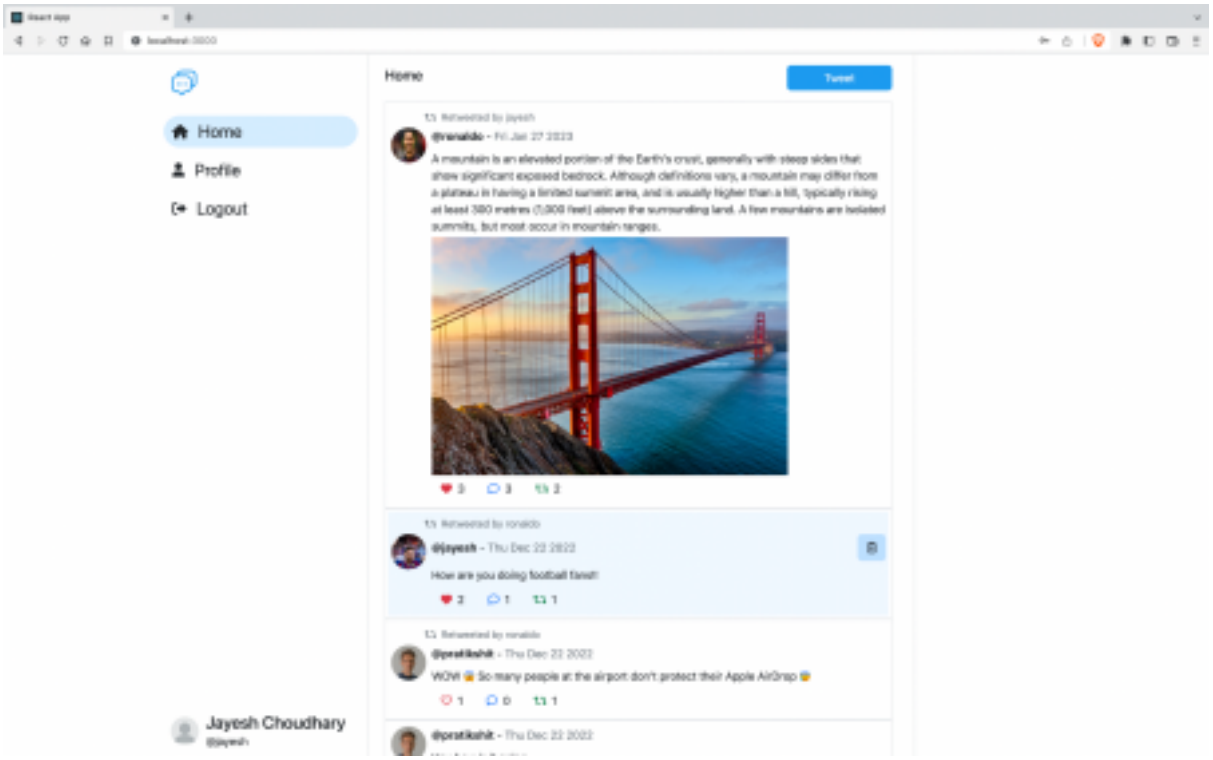
Another screenshot with more tweets (note: the tweets list section is scrollable and sidebar will remain as sticky)



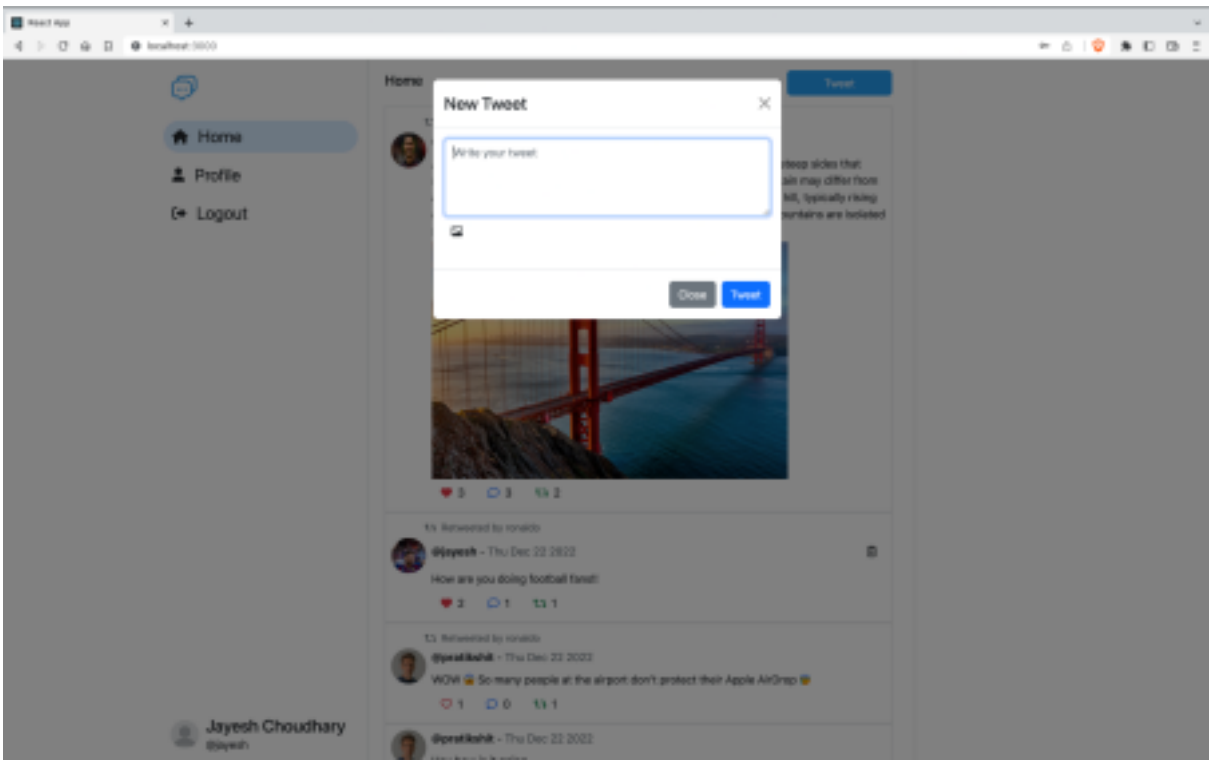
Hover state of buttons in tweet card



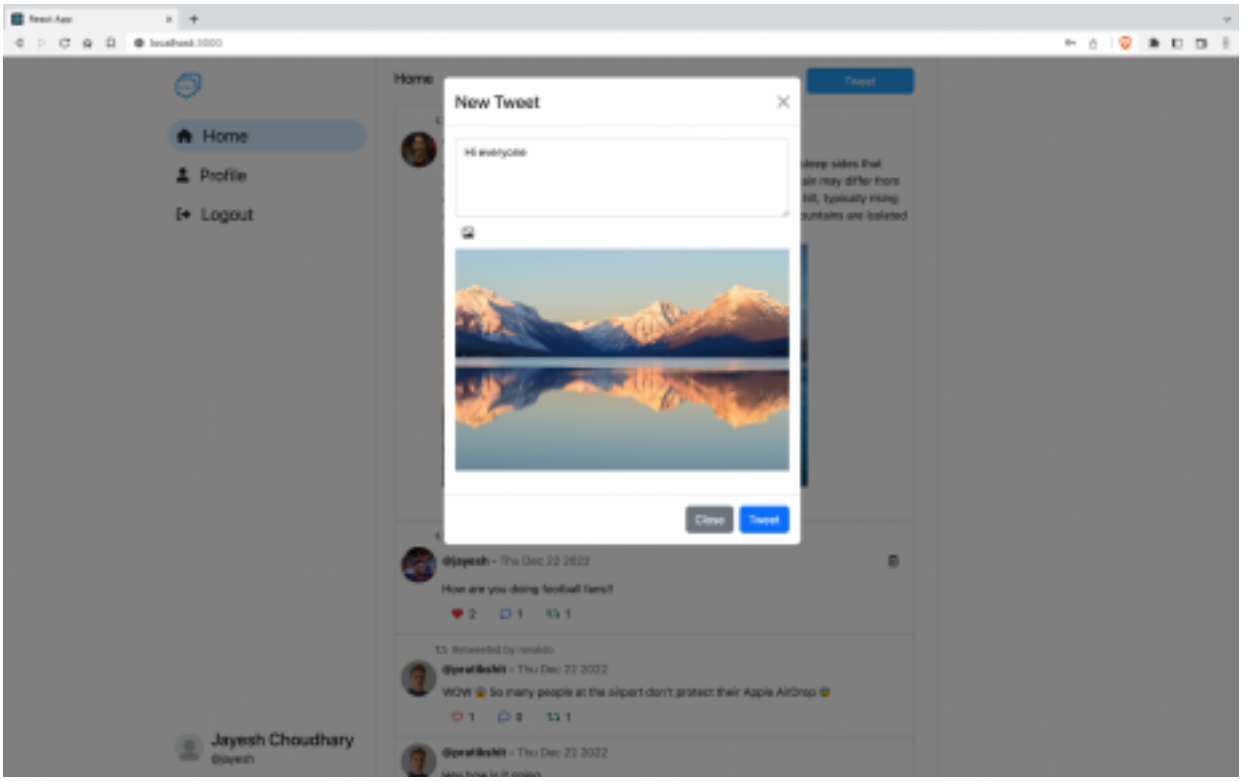
Delete button (only visible on tweets which logged in user tweeted)



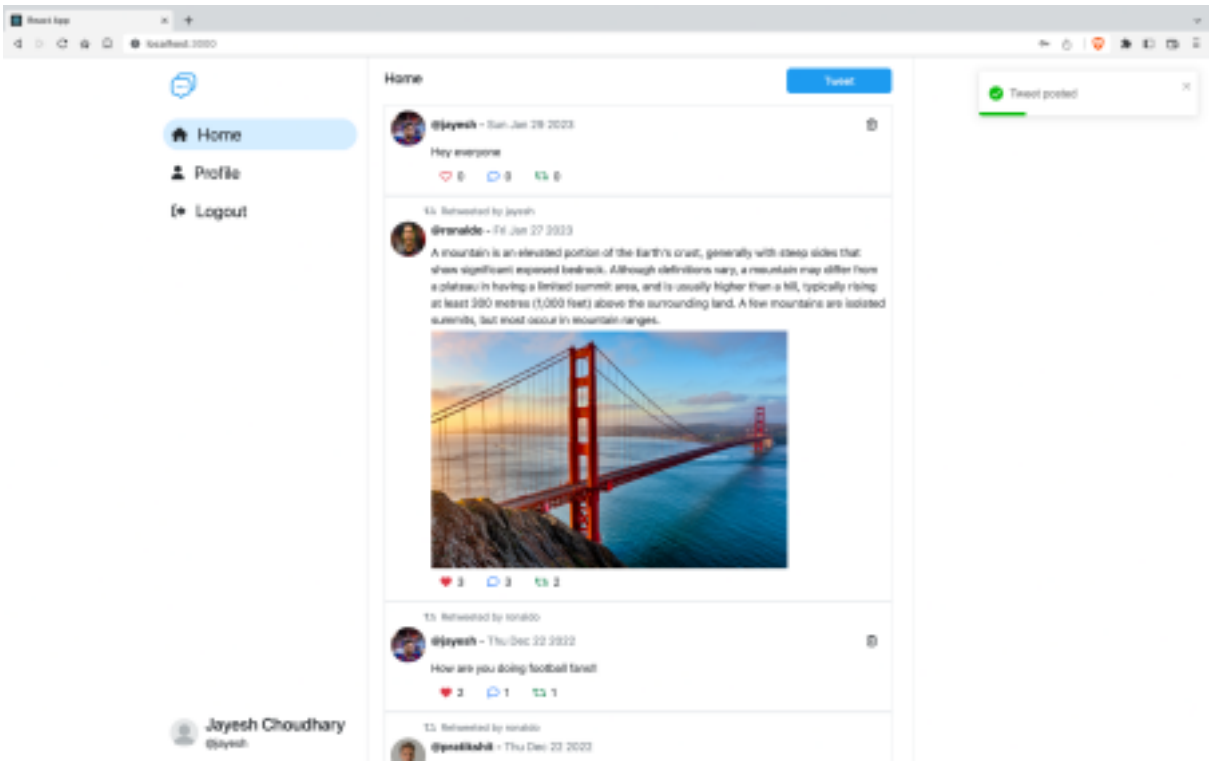
Clicking on the tweet button will open this dialog



Dialog with contents filled in



Show toast after every successful operation

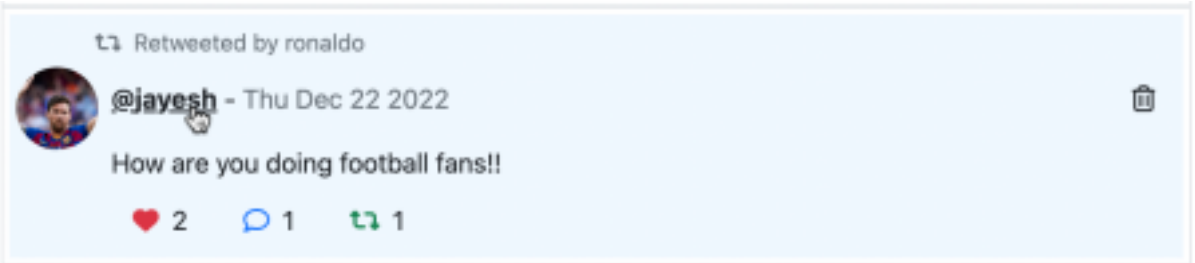


Retweet indicator

Add a text indicator like below if a tweet is retweeted by someone (with username)

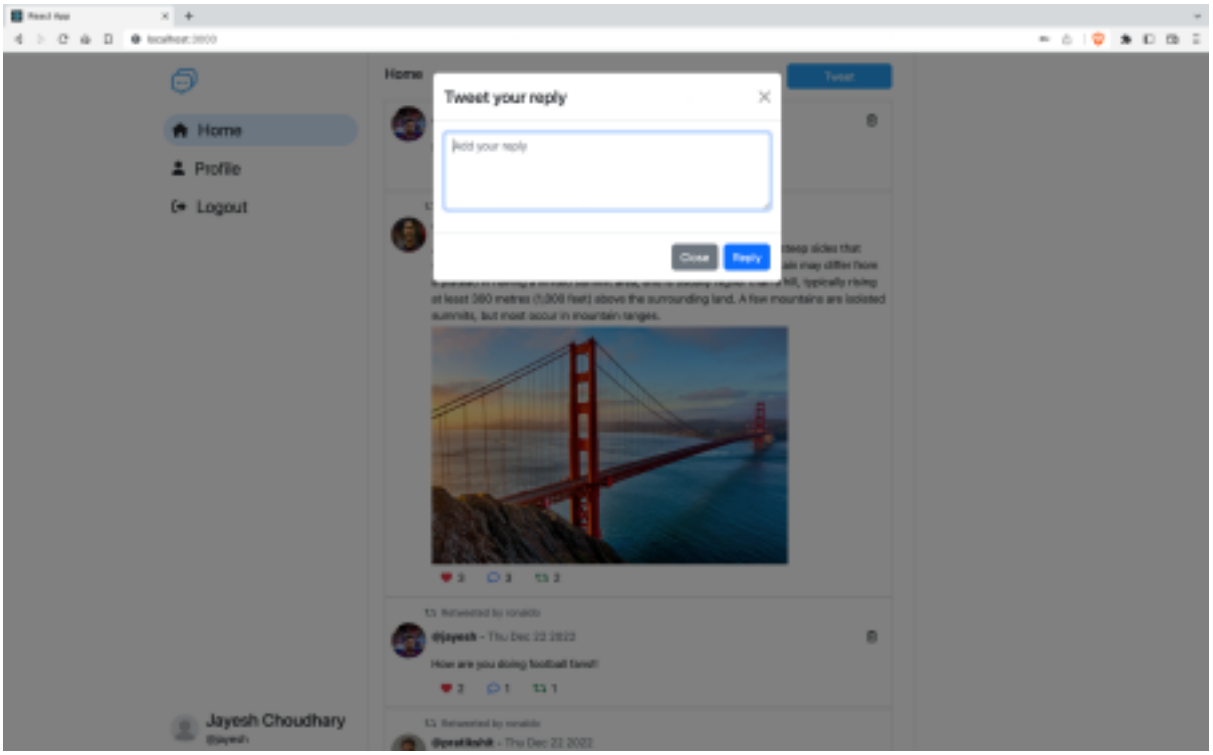
Username in the tweet card

username in the tweet card is clickable which will go to that user profile page



Reply on Tweet

Clicking on reply button will open a dialog where you can enter tweet details and reply your tweet



4. **Profile Details Page**

   - user profile page can be visited by clicking username from tweet card - we will display every user details - photo, name, username, DOB, joining date, location, followers and following data
   - List all the tweets created by this user

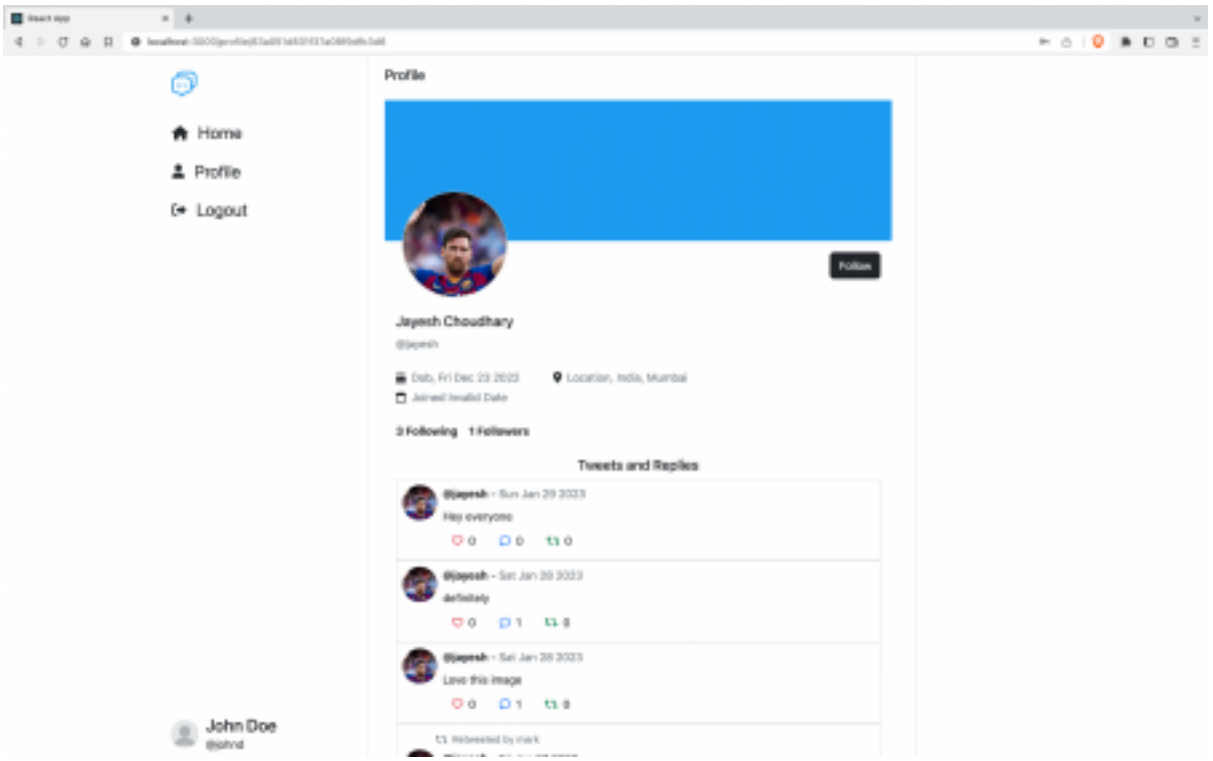
   We have two profile page

   A. **other user profile page** (here we can show follow and unfollow button, note: any one button will be shown at a time)

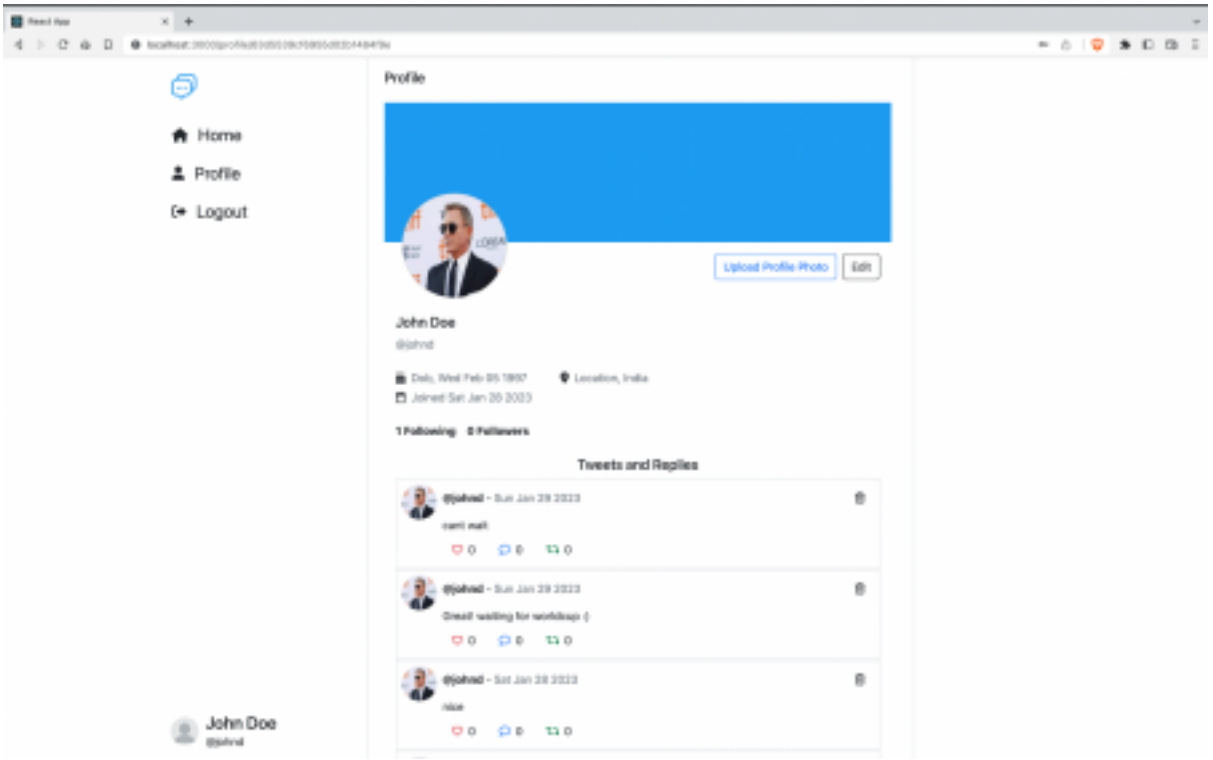
   B. **logged in user profile page** (here we can show edit details and upload profile pic button)

   user can visit this page on clicking user profile in sidebar (which is logged in user profile)

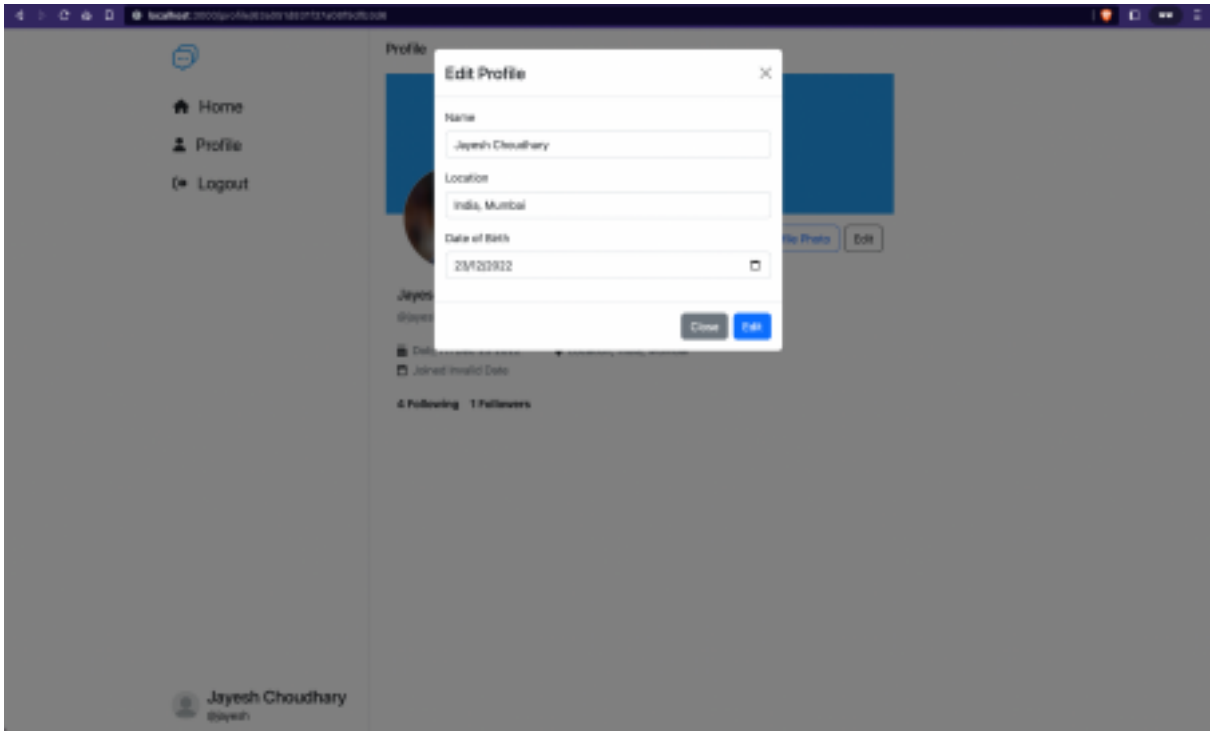
   Other than self profile details page



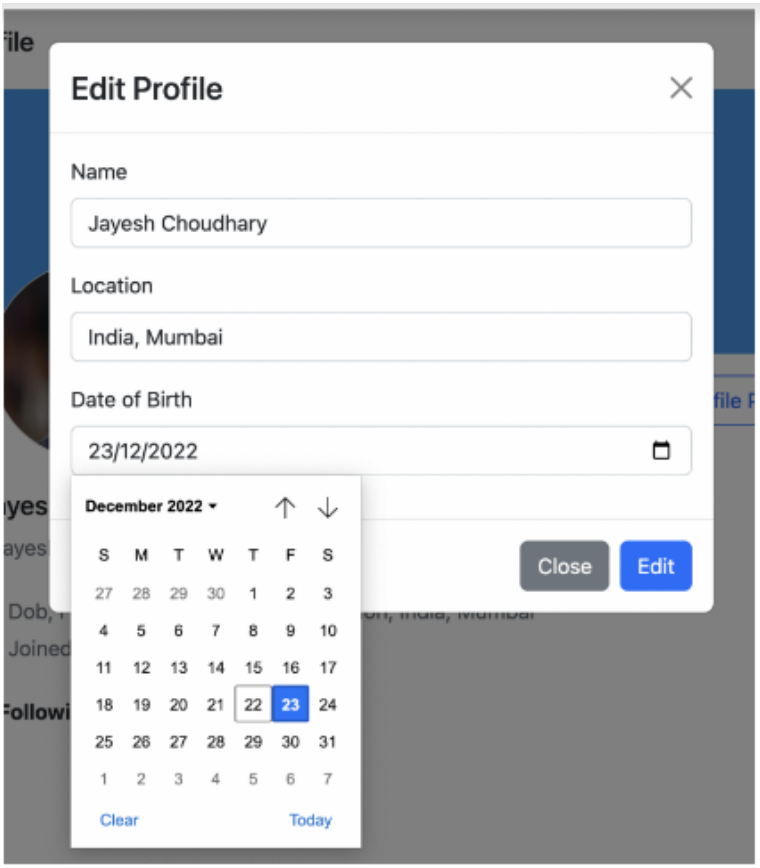
Logged in usder Profile details page (owner page where we hide follow / unfollow button and show upload profile photo and edit buttons)



Edit profile button

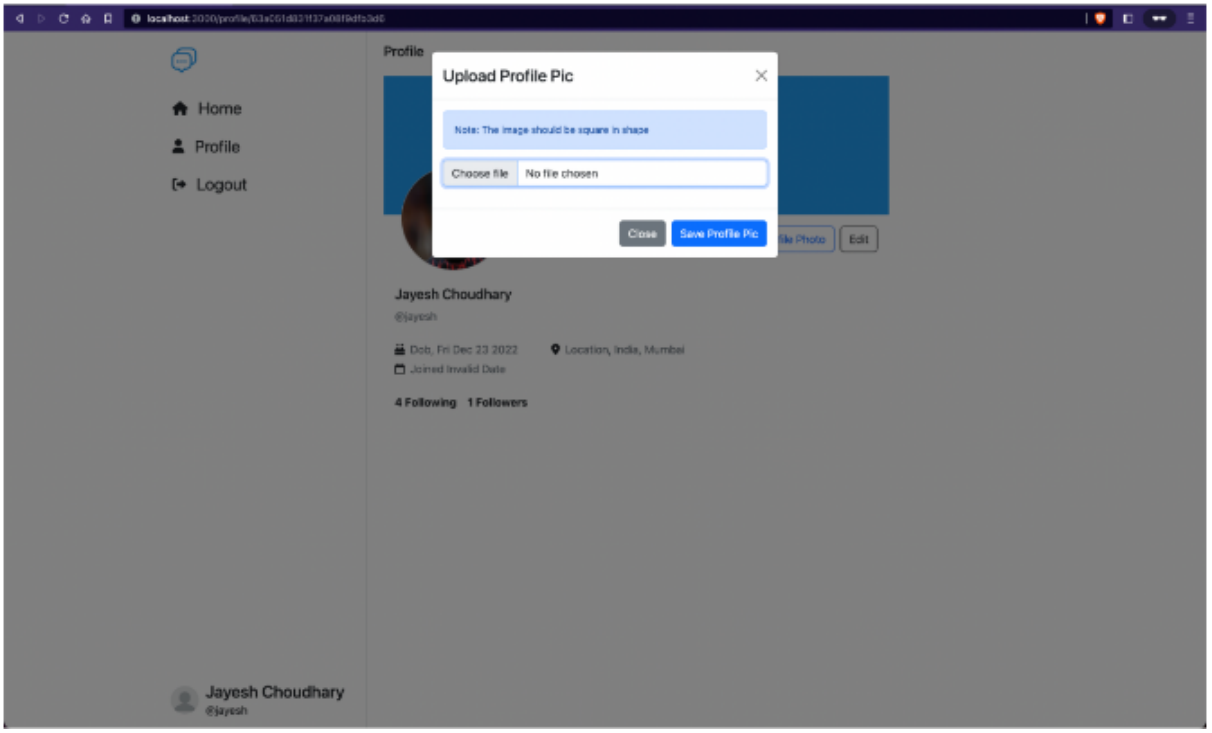


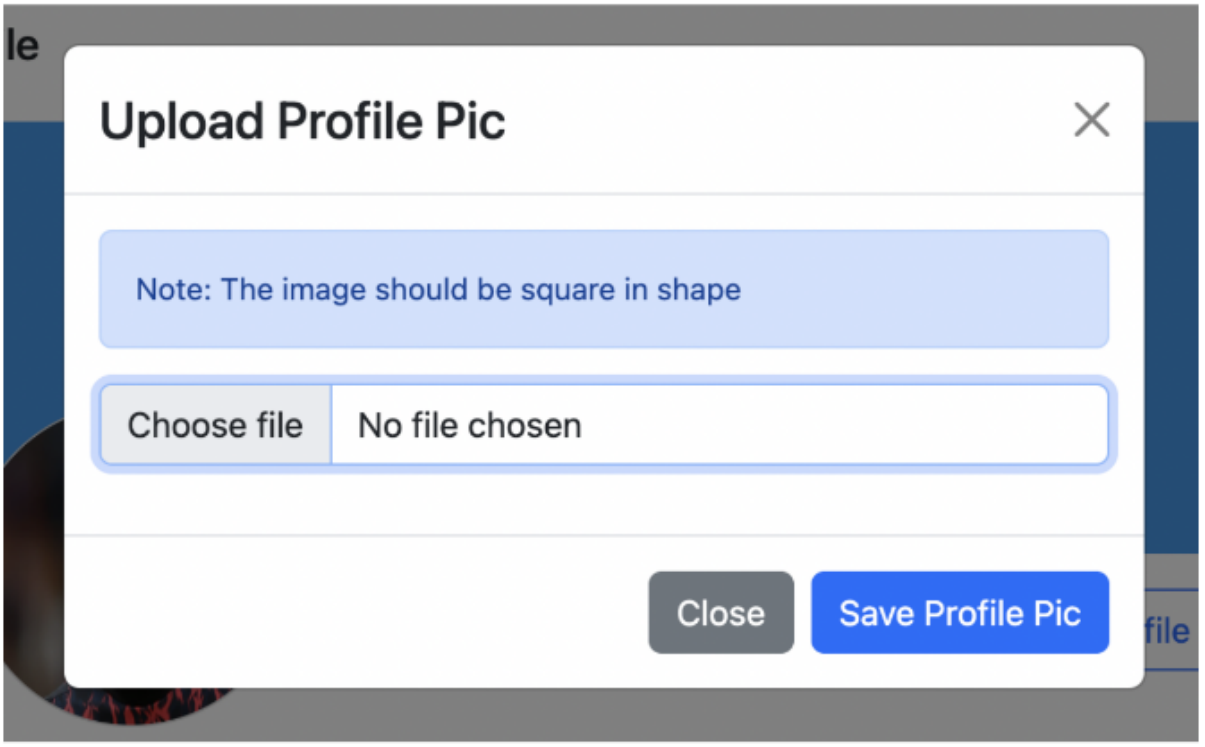
Date selector input

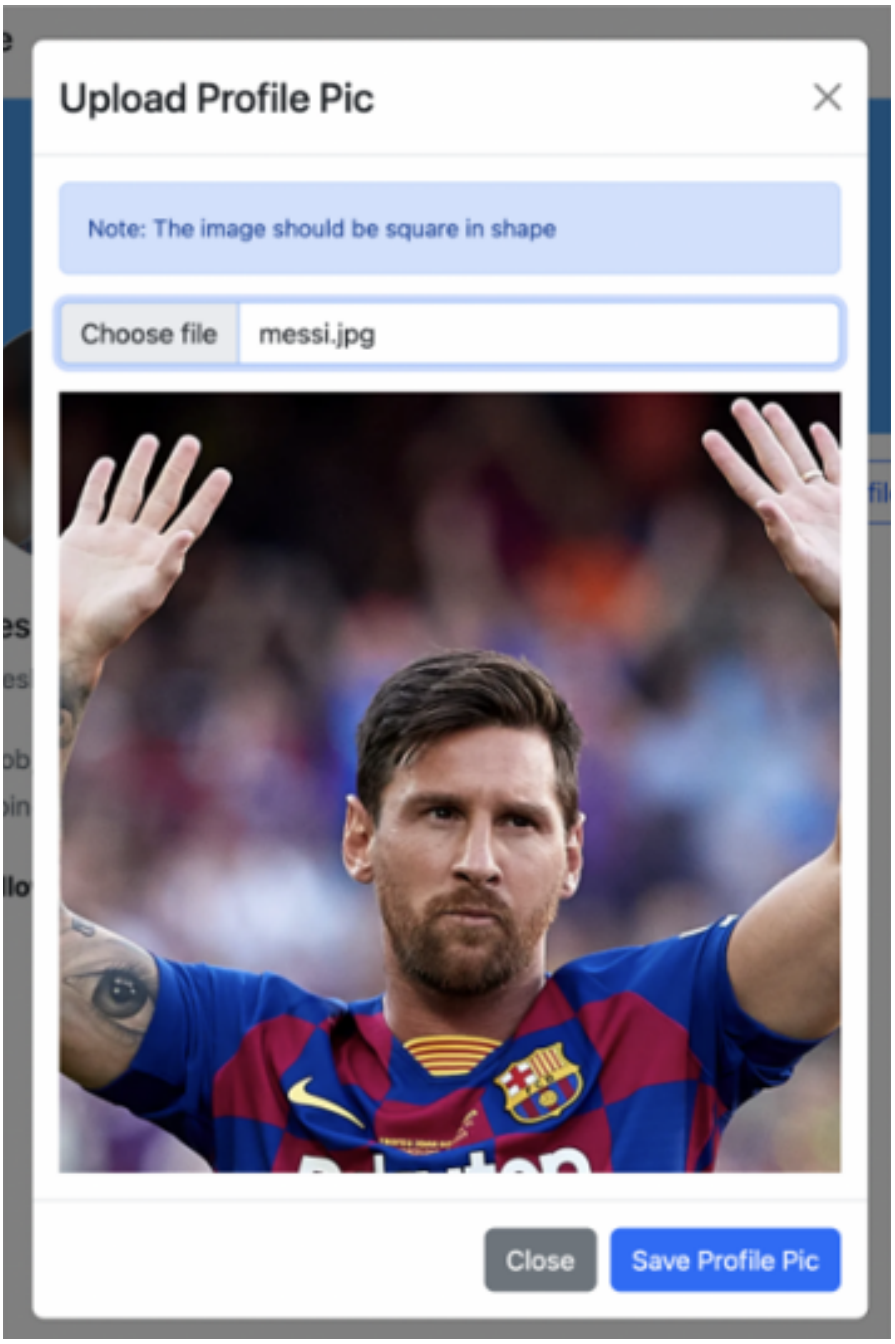


Note: All three fields are required, so show errors when a user tries to submit the form without filling all the required details

Upload profile pic


Home
Profile
Logout
Profile
Upload Profile Pic
Note: The image should be square in shape
Choose file
No file chosen
Close
Save Profile Pic
Edit
Jayesh Choudhary
@jayesh
Dob, Fri Dec 23 2022
Location, India, Mumbai
4 Following
1 Followers
Jayesh Choudhary
@jayesh


Upload Profile Pic
Note: The image should be square in shape
Choose file
No file chosen
Close
Save Profile Pic

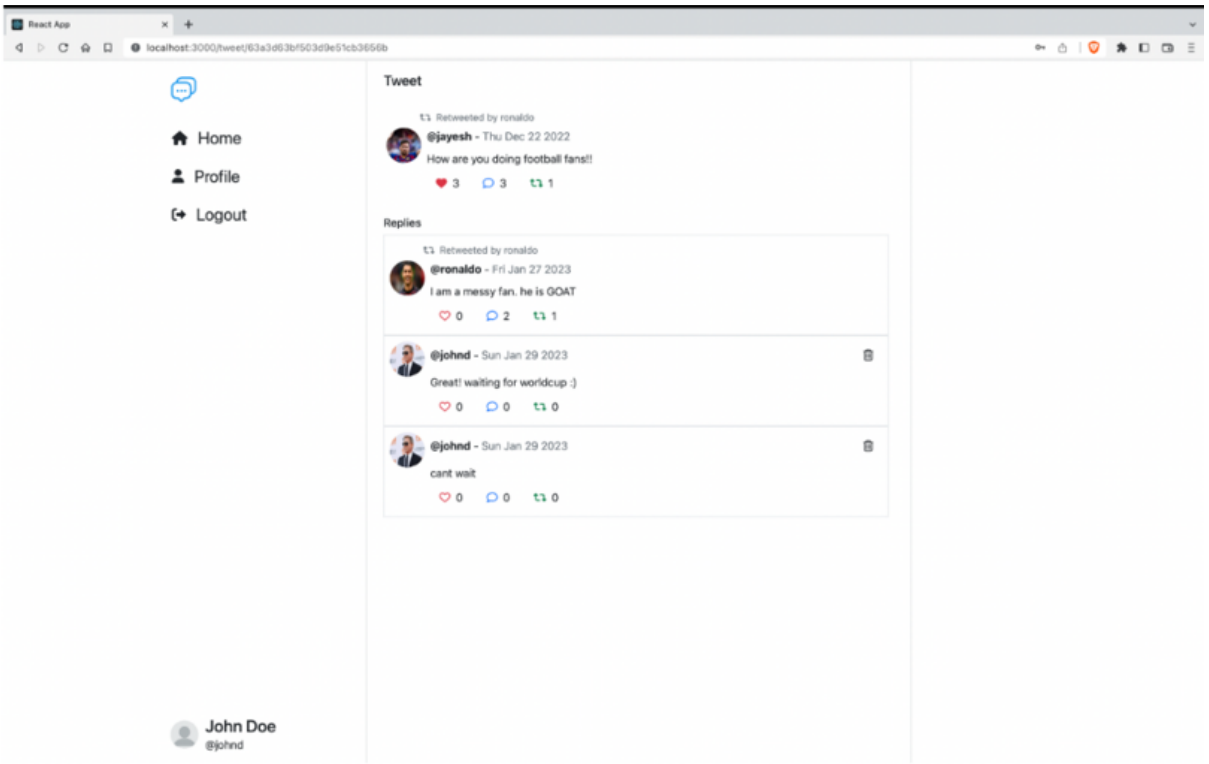


Note:

Preview the selected image file before submitting and show an error if the user tries to submit the form without selecting pic, i.e picture is a required field here.

5. Tweet Details Page

- **I**n this page show a tweet details and all the replies in that tweet
- You can click on a tweet card to arrive at this page
- Since reply is a tweet itself it will appear the same as tweet with all the same buttons and operations.



Evaluation

Total - 100 Marks

Frontend - 50 Marks

1. Create all the given pages with working functionality as mentioned – 30 Marks
2. The design looks similar to the ones shown in the screenshots – 5 Marks.
3. Proper success and error handling (i.e showing success and error notifications when required) 5 Marks
4. Authentication flow should work and behave as mentioned (when page reload user should still be logged in, pass jwt token into every api call, save and persist jwt token etc.) – 5 Marks
5. All the code files are neat, with structured folders, and comments added to segregate the sections – 5 Marks

Backend - 100 Marks

1. All APIs should work and behave as mentioned - 30 Marks
2. Define proper schema for MongoDB data - 5 Marks

3. Sending appropriate success and error message when required, also send appropriate status code (200 when success, 500 when unknown error etc…) (error handling is important) - 5 Marks
4. Image upload feature using multer - 5 Marks
5. All the code files are neat, with structured folders, and comments added to segregate the sections – 5 Marks