SOCIAL MEDIA WEB APPLICATION

Group Member Details

Ameeya Ranjan Sethy	(200050006)
Anubhab Ray	(200050010)
Jasani Parth Alpeshbhai	(200050053)
T E D N V S KRISHNA KAMAL	(200050142)

Overview

Increasing Social Media presence demands a way of connecting with friends / colleagues as easy as it can ever be. But there are numerous such professional platforms already developed, so what we bring on the table is a version of facebook / Instagram dedicated for the campus. Our user base shall be primarily college students, a social Media web app that curates to our needs.

User Viewpoint

There will be only one category of users (as of now) who would log into the social media service to find their friends, etc. They form a complete entity with user info / biodata / etc.
They can publish posts, stories (timed posts) in the form of daily Status and will remain forever as posts. Here they can give pictures and text captions. The posts will store the time of posting and can easily be edited later.

Can follow other people and accept such requests, i.e. be followed by other users. Now if I am following someone, then like Instagram I will see his/her posts in my feed in our home page. Can even save posts and see them separately in a dedicated section in the Profile page.

Comment on other people's posts. Like and share them, as links on all other social media websites. Can do nested comments, just like all platforms, can edit/delete and like each comment (obviously with user privileges). Basically a user forms his network of friends and accesses their posts/stories, interacts with them.

Can chat with friends / non-friends (constraints might be implemented later). See statistics of their number of followers and following on our platform, audience interaction with their posts etc.
While registering they can only provide the necessary details for registering and later on add all the other info like their address, website, etc. by using the edit profile option.

Database Viewpoint

We are using a noSQL framework namely the MongoDB. The database of our social media stores only text data and the pictures are uploaded on Cloudinary free service, there we made an API endpoint of upload and retrieval of image data. So whenever we post our images in Status/posts or profile pictures, they are directly uploaded on Cloudinary and we GET an external url for referring to it.
Now coming to the text data in MongoDb collections, we have 4 collections namely the **Posts, Users , Comments and Message** (might add more later based on requirements).

The **Comments** collection has in each of its document fields as

- _id (string),
- likes (array of user ids),
- user (a user object for another document) and
- content (string).
- Obviously the default field of "Created At" and "Updated At" will be present in all of the collection documents.

The **Posts** collection has in each of its document fields as

- _id (an Object Id),
- images (array of image objects each containing a public Id and url of external ref.),
- likes (array of user ids),
- comments (array of comment object ids),
- a user (a user object for another document) and
- content (string).
- Obviously the default field of “Created At” and “Updated At” will be present in all of the collection documents.

The **Users** collection has in each of its document fields as

- _id (an Object Id),
- avatar (cloudinary image external url),
- Cover-photo (cloudinary image external url),
- Role (string, as of now it's just “user”),
- Gender (string),
- Mobile (string),
- Website (string as in git-hub),
- Address (string),
- Story (string for bio as in instagram),
- Fullname (string),
- Username (string),
- email (string),
- Password (string of hashed value),
- Following (array of user objects each containing an _id and necessary info dynamically fetched with the Object Id),
- Followers (array of user objects each containing an _id),
- Saved (array of post objects each containing an _id),
- Obviously the default field of “Created At” and “Updated At” will be present in all of the collection documents.

The **message** collection has in each of its document fields as

- text (a message)
- sender (object id of sender)
- recipient (object id of receiver)
- Media (array of images)
- Timestamp of message (when user sends)
- Obviously the default field of “Created At” and “Updated At” will be present in all of the collection documents.

The **Conversation** collection has in each of its document fields as

- text (a message)
- recipient (object id of receiver)
- Media (array of images)

Application Structure

Backend

- For **authentication** purpose we have added following APIs
    - URL : /api/register
        - Type : POST
            - This API takes data from request.body and after validating data it creates a new User. and additionally it generates a jwt token for authentication and saves it into cookies at path “/api/refresh_token”.

    - URL : /api/login
        - Type : POST
            - This API takes data from request.body and after validating data such as hash password match and email verification it generates a jwt token for authentication and saves it into cookies at path “/api/refresh_token”.

    - URL : /api/logout
        - Type : POST
            - This API clears refresh_token from cookie which was saved while user login/register.

    - URL : /api/refresh_token
        - Type : POST
            - For users who have logged in it creates an access token that is useful in authentication.

- For **user** functionalities we have added following APIs
    - URL : /api/search
        - Type : GET
            - This API is used while implementing search-bar in the header. This API takes a query string as an input and returns a list of users for which this query string matches as per number of user limits.

    - URL : /api/user/:id
        - Type : GET
            - This API helps to get data of the user for a given id.

    - URL : /api/user
        - Type : PATCH
            - This API handles update functionality for user objects (i.e in Edit Profile). Takes data from request.body such as fullname, story etc.. and updates the user object.

    - URL : /api/user/:id/follow
        - Type : PATCH
            - This API handles the following action. API takes input as the user id of the user who is followed by the logged in user. And API adds user id to followers of logged in user who makes request to follow.

    - URL : /api/user/:id/unfollow
        - Type : PATCH
            - This API handles the unfollowing action. API takes input as the user id of the user who is unfollowed by the logged in user. And API removes user id to following of logged in user who makes request to unfollow.
    - URL : /api/suggestionsUser
        - Type : GET
            - This API handles suggestions functionality shown in the home page. API returns a list of users who are not followed by logged in users.

- For **post** functionalities we have added following APIs
    - URL : /posts
        - Type : POST
            - This API creates a new post, used in the home page where there is a status box which is used for creating new posts. Input is content and array of images and after creating new objects of post API stores in the database.
        - Type : GET
            - This API helps to list all the posts of followers of logged in users in the home page. Input is user id and this will return all posts reading for the database.

    - URL : /posts/:id
        - Type : PATCH
            - This API helps in updating the content. Inputs are updated content of posts. Used as an option in post.
        - Type : GET
            - Get the data of the post with the given id.

        - Type : DELETE
            - This API will be used to delete posts. Input is id of post and user id as we need to update an array of user posts. Used as an option in post.

- URL : /api/post/:id/like
  - Type : PATCH
    - This API will be used to implement post-like features. Inputs are post id and user id as we need to store which user likes this post and also keep like count.

- URL : /api/post/:id/unlike
  - Type : PATCH
    - This API will be used to implement post-unlike features. Inputs are post id and user id as we need to remove user-id who unlikes this post and also keep like count.

- URL : /api/user_posts/:id
  - Type : GET
    - This API will be used to get all posts of users with a given id as input. API helps to show user posts at profile page.

- URL : /api/post_discover
  - Type : GET
    - This API will be used to list all posts of users who are not in the following list of logged in user.so input is only user id and following array of user.

- URL : /api/savePost/:id
  - Type : PATCH
    - This API will be used to add post to the saved post of the user. Inputs are user-id and post-id. API will add post-id to saved array of user and can be shown separately to profile page.

- URL : /api/unSavePost/:id
  - Type : PATCH
    - This API will be used to remove post from the saved post of the user. Inputs are user-id and post-id. API will remove post-id from saved array of user.

- URL : /api/getSavePosts
  - Type : GET
    - This API will be used to fetch saved posts. User-id can be fetched from request.user and API will return an array of saved posts and we have implemented this feature at profile page where user can see these posts.

- For **chat** functionality we have added following APIs
  - URL : /api/message
    - Type : POST
      - This API will create messages from sender to recipients. Input to this API will be from a form which consist of text messages and media files if any and recipients id.
      - If the recipient is not in the conversation model database then it creates this new conversation otherwise it will update current messages.
      - This message will be added to the message database with the above conversation id. And other essential details.

  - URL : /api/conversations
    - Type : GET
      - This API will fetch conversations of any user that will help in showing conversation to the chat page. Input to this API will be user id.

  - URL : /api/message/:id
    - Type : GET
      - This API will fetch all conversations between a user with saved user id at auth cookie and user with id from request. This will help in showing the whole conversation in the chat page between sender and recipients.

    - URL : /api/message/:id
        - Type : DELETE
            - This API will help in deleting any message.

- For **Comment** functionality we have added following APIs
    - URL : /api/comment
        - Type : POST
            - This API will add a new comment to the comment model. Inputs are from user forms and fields are post_id, user_id , comment content and other relevant fields.
    - URL : /api/comment/:id
        - Type : PATCH
            - This API will update comment content which has comment id as equal to passed id in request parameters.

    - URL : /api/comment/:id/like
        - Type : PATCH
            - This API will update likes given to that comment with user id passes as request parameter.

    - URL : /api/comment/:id/unlike
        - Type : PATCH
            - This API will remove like from comment id passed as request parameter.

    - URL : /api/comment/:id
        - Type : DELETE
            - This API will delete comment with id same as passed to request parameter.

UI

Major components

- **User card**
  - Contain avatar, full-name, and user-name
  - Linked to profile of that user

- **Post Card**
  - There are three section footer, body and header.
    - Header contains avatar, username and caption of post
    - Body contains images of posts
    - Footer contains like, bookmark and share buttons and contains all comment

- **Panels, Headers, Footers, Info-Boxes, Search box, etc.**
  - Contains respective fields required for the respective pages
  - Used extensively in all pages with different UI variations

- **Status sending/Post creation window**
  - Contains a form to enable access to camera and take pictures
  - We can also upload files (images) and also see the pictures uploaded and clicked in real time.
  - We can add a text to it called caption and post it.

- **User details editing window**
  - Contains a form to see the details of the user like name, address, email, etc.
  - They can edit all of them and also upload 2 pictures - profile and cover photo. Now we have made it shown side by side to ease the UX. They can either save or cancel the updates.

Major Pages

Login Page

- Has two input fields one for **email** and another for **password**. And if the user is not registered then there will be a button **linked to the registration page.**
- User will be **redirected to the home page** after successful login else there is **toast message** appearing at upper right corner which has information about **error** occurred.

Registration Page

- Has various input fields such as **fullname, username, email, phone number, gender** etc. and if the user is **already registered** then there will be a login button which **redirects to the login page**.
- In this page there are many validation and compulsory fields like email, username must be unique, phone number must have length 10 etc. and if there is any error there will also be a toast message about error at upper right corner.

Header

- In the header we have included **our logo** and **search bar**, menu option for **changing theme**, **link to profile page** and **logout button.**
- Header will appear in all major pages like discover page, home page and also in profile page.
- A click on the logo will take you directly to the home page.
- On the right end are a search bar and our avatar pic.
- We can search a new user and go to his/her account page from the search results that would drop down.
- The avatar is a drop down menu itself used to logout, change themes, etc.

Home Page

- Home page have three sections i.e left,middle and right
- In left part
    - User card with full name, avatar and username
    - Likes redirecting to different pages like home, discover, chat, settings.
    - Below this there is a suggestion for user to connect with other users.
- In middle part
    - List of posts of users who are followers of logged in user.
- In right part
    - We had two sections containing followers and following users-cards.
    - Each has a load-more button which shows all the follower and following respectively same as profile page links.

Post Page

- This page has a single post, the one selected as triggering action before.
- Now this page has the entire post card with all the info about the post body.
- On the left is the pictures carousel and right is the content section containing the caption, the count of likes and of comments.
- At the bottom is the option to send a comment and then a space to view all comments.
- In general comments are clubbed/minimised and we can maximise and view all of them. This feature is implemented in nested comments as well.
- There is a dedicated window popping up if the caption is too large. The minimised caption is only shown in the post content section on the right.

Discover Page

- This page has a list of posts but only the post body and hovering over it shows a like and comment option and by clicking we can move to the original post.
- These are the posts of users who are not in the following list of logged in user.

Message/Chat Page

- There is an interface where users can share images.
- Users can also send text messages **.**
- We have implemented real-time messages through sockets.
- We can search a user across a global collection, to find and start a conversation.
- There are 2 panels - left and right, left has a list of all users with whom we had a chat with and the search box part as well.
- On the Right Panel, we will show a demo page when no specific user is selected from the left panel. And if selected, then we show the conversations with them over there.
- At the top of Right side (in Conversation Mode) we have a msg sending option obviously. There is an input box which contains text-area fox text messages and a button to insert emoji, files.
- By hovering on the self messages the user can delete their own message.
- Another section on this right part contains all media files that have been shared in conversation and all messages that are sent out to recipients or messages that are received.
- A special feature is the storage of the last message sent/received in a conversation. A respective info is shown on the left side, below the Username of the chat-ee.

Profile Page

- Has all details of user like avatar, cover photo, username, fullname , website, story , mobile number.
- There are two major linkes one for followers and another for following users. By clicking on this shows up lists of users respectively as list of user-cards.
- For logged in users there is an edit profile button by clicking on it form shows up with every details that user can change.
- And if a user visits another user profile then there is a follow/unfollow button based on whether the user already follows or not.
- Below this there are two tabs one for all posts and another for saved posts.

Feature can be added if more time available

- **Post of video content**
    - As video content becomes popular this days we can also add this feature to our platform
    - Here are some important uses of this feature
        - Better storytelling
        - Likely to earn additional revenue
        - More shareable content

- **Notifications**
    - The notification feature is an essential aspect of social media application.
    - Here are some important uses of this feature,
        - Keeping user engaged
        - Personalisation
        - Time-saving
        - Encourages interaction between users

- **Deployment**
    - Currently our web application provides a chat feature but the user needs to refresh for loading new messages. So if we want to work upon this factor.

- **Group Chats**
    - This is also becoming an important feature of social media platforms as it can be used as the most convenient way to send messages to groups of people.

- **Calls**
    - This call feature is also very essential for any social media application.
    - Here are some important uses of this feature
        - Real-time communication
        - Better collaboration
        - Enhances user experience while using platform
        - Increase engagement

What We Have Learnt

- As part of the group project we have learnt **Collaborative teamwork.** In a group of 4 people we need effective communication, coordination and collaboration. As we need to complete this project with some strict deadline from this we have learnt to set timelines for each group of members for respective tasks. Also we need to manage group work by dividing tasks into members.
- We have used the **MERN framework** which includes MongoDB, ExpressJs, ReactJs, and nodeJs. By working with these technologies we have learned how to build full-stack web applications from scratch. We have learnt how to integrate these different technologies together to develop this web application.
- As part of the project for front-end development we have used ReactJs. In that we have learned how to build interfaces, reusable components, and handle user interaction.
- As part of the project for backend development we have used Nodejs and expressJs. By this we have learnt how to build APIs, handle HTTP requests and responses and interaction with remote databases.
- We learnt Redux, and how powerful it is. It's very hard to maintain states of individual components and pass information from child to child and its parent components, with redux it's easy once setup successfully.
- Crucial Point is Database management to this Project and as MongoDB is a popular NoSQL database. We have learnt how to use mongoose for creating/storing new data to remote databases and retrieving this data and serving as a response.
- We have managed this project on github. From this we learnt how to share code efficiently and manage remote repositories.
- We learnt usage of 3rd party services like Cloudinary to upload and store pictures and store in the database the external public url of that image.
- Managing a project involves various tasks such as planning, organising and executing.

Overall , working on a social media web application project using MERN stack in a group of 4 provided valuable experience in web development, teamwork and project management.