“A non-addictive social media app”

ABSTRACT

Most of the social media apps we use today are overwhelming and addictive which affects students' mental health. Nowadays free social media apps are not actually free, they use our data as their prime revenue to generate personalized ads which many people won't like as they feel insecure. And all the features are not available in a single social media. We have to use multiple social media for multiple purposes.

And here we are with our app Mediaverse which solves the above-mentioned problems. In this app, users can post images, share their thoughts, send or receive messages and many more just like any social media. But the difference is we are planning to provide a disappearing post/thoughts feature that helps users to avoid doom scrolling. We are also planning to include some of the unique features of most of the famous apps. This app doesn't use user data to generate personalized ads.

One of the most useful features for students and developers is discussion platforms like StackOverFlow, StackExchange, Quora, Brainly, etc. are not much focused on the famous social media apps that we are using today. We are here to solve this problem with our Discussion Forum where users can post their doubts and other users can help them with the answers.

The most liked feature in Snapchat is SnapMap which is not available in most of the famous apps. We are planning to build a modified version of that feature that helps users to track their followers and enables them to know the location of the users around them. These are some of the features of our app but we also planning to include more features. From this project, we hope to build a better, non-addictive, secured, and all-in-one social media app.

TEAM MEMBERS

- Dhev Sabarish – 2020103518
- Logeshwaran – 2020103538
- Velmurugan - 2020103585

FEATURES OFFERED

- Disappearing posts
- Trending posts based on Trending Hashtags
- Mapverse – Users Map
- Discussion forum
- Private chat and Global Chat

TECH STACK USED

“MERN Stack”

➔ Frontend

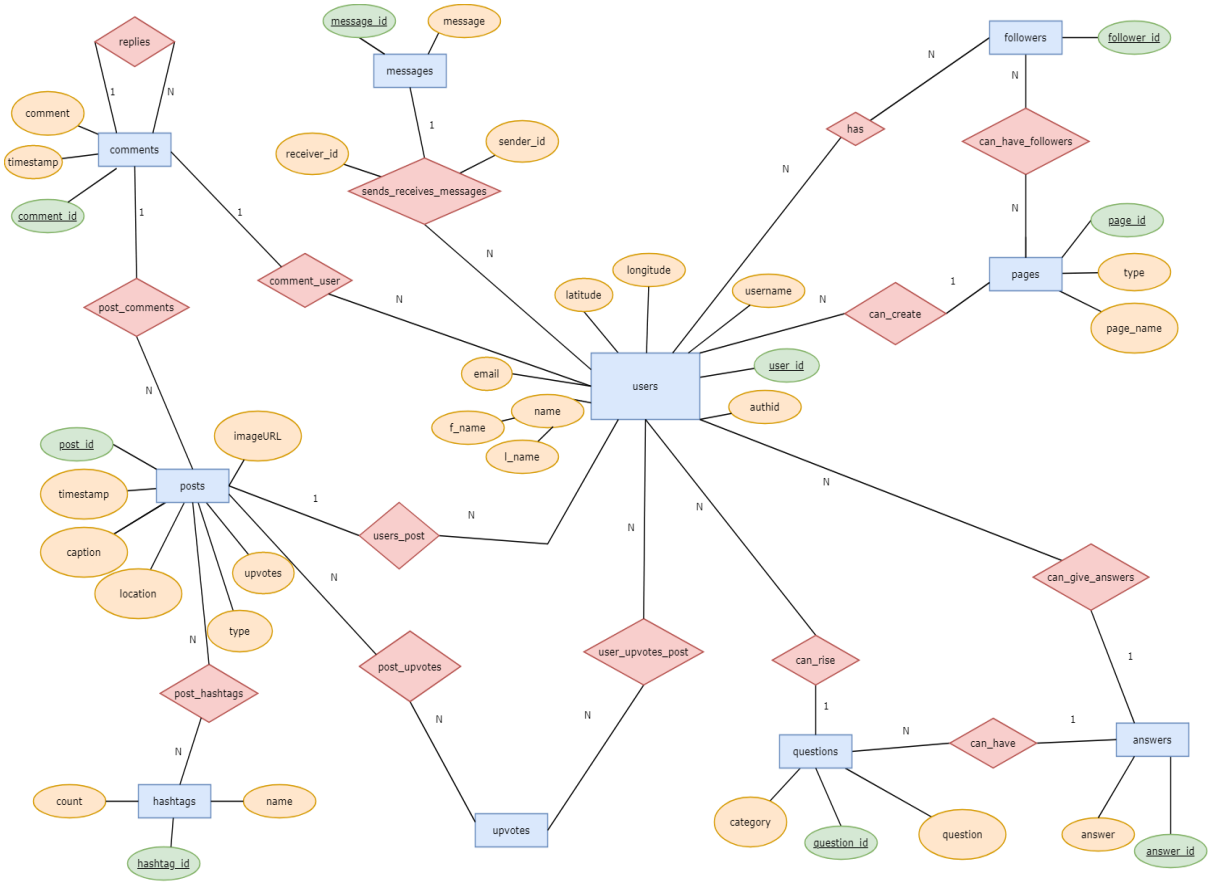
- React Native
- Expo
- Firebase Authentication
- Firestore database for chatting
- App will be published in Expo GO

➔ Backend

- Node JS – Runtime environment for JavaScript
- Express JS – Creating REST API’s
- MongoDB – NoSQL database (cloud)
- Mongoose – provides abstraction over MongoDB
- Babel JS for transpiling JavaScript
- API will be deployed in HEROKU

BACKEND OF THE APP

Entity-Relationship diagram:



Since MongoDB is a NOSQL database we don't table to store the data. It's a document based database and it is capable of processing structured, semi-structured and unstructured data. So we need to define specific schemas for each collections.

In MongoDB,

- Set of Entities are called as **Collections**
- Each Entities are called as **Documents**.
- Documents can have attributes as key value pairs.

Schemas completed till now:

1) User Schema :

```
const userSchema = mongoose.Schema(
   {
      authid: { type: String, required: true }, // comes from
firebase
      fname: String,
      lname: String,
      username: { type: String, required: true },
      email: String,
      latitude: Number,
      longitude: Number,
      profileImg: {
         type: String,
         default:'https://firebasestorage.googleapis.com/v0/b/soc
ialapp-5e56c.appspot.com/o/avatar.png?alt=media&token=37bb52ed-
61ed-4e00-88de-e9a5eb6ae2bb',
      },
      upvotedPosts: [{ type: mongoose.Types.ObjectId, ref:
'Posts' }],
   }
);
```

2) Post Schema:

```
const postSchema = mongoose.Schema(
   {
      userid: { type: mongoose.Types.ObjectId, ref: 'Users',
required: true },
      type: { type: String, enum: ['text', 'image'], default:
'text' },
      upvotes: { type: Number, default: 0 },
      comments: [{ type: mongoose.Types.ObjectId, ref: 'Comments'
}],
      caption: String,
      hashtags: [{ type: mongoose.Types.ObjectId, ref: 'Hashtags'
}],
      imageURL: String,
      location: String,
   },
);
```

3) Follower Schema:

```
const FollowerSchema = new mongoose.Schema({
   userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Users',
      required: true,
   },

   followerId: [
      {
         type: mongoose.Schema.Types.ObjectId,
         ref: 'Users',
         required: true,
      },
   ],
});
```

4) Hashtag Schema:

```
const HashtagSchema = mongoose.Schema({
      name: String,
      count: Number
});
```

5) Upvote Schema:

```
const UpvoteSchema = mongoose.Schema(
   {
      userid: { type: mongoose.Types.ObjectId, ref: 'Users',
required: true },
      postid: { type: mongoose.Types.ObjectId, ref: 'Posts',
required: true },
   }
);
```

6) Comment Schema:

```
const commentSchema = mongoose.Schema(
   {
      userid: { type: mongoose.Types.ObjectId, ref: 'Users' },
      comment: String,
      replies: [{ type: mongoose.Types.ObjectId, ref: 'Comments'
}],
   }
);
```

Rest API routes completed till now:

/Users

- Get all users of the app
- Sign up
- Sign in
- Edit user details
- Delete a user

/Posts

- Get posts of an user
- Get a post by PostId
- Get Trending Posts based on Hashtags
- Get Global Trending Post
- Create new post
- Upvote/Downvote Post
- Delete a post

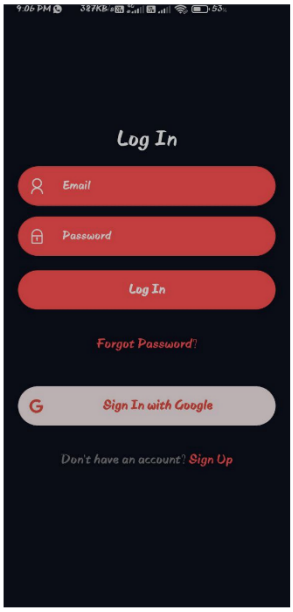
/Comments

- Get comments of a post
- Create a Comment on Post
- Create a reply on Comment
- Delete a comment on Post

/Follow

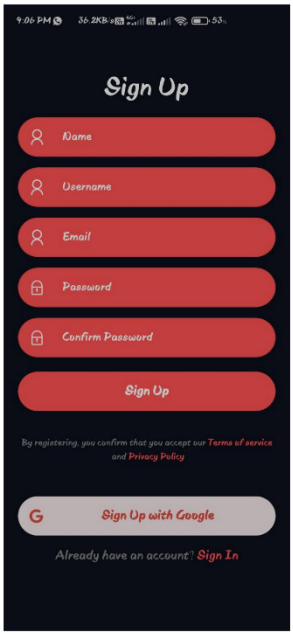
- Follow / Unfollow any user
- Get all followers of a user

FRONTEND OF THE APP



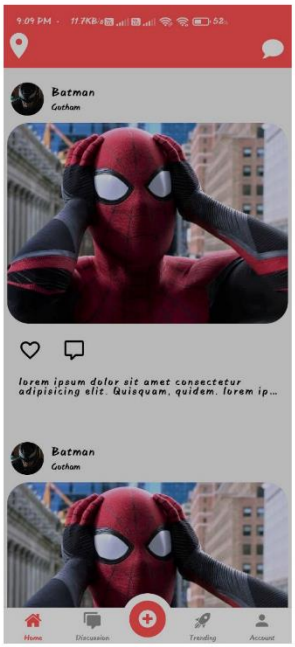
Login Screen

The login screen asks for an email address and a password, then checks to see if the user already exists in the database (Powered by Firebase). The user can now navigate to the home page if he/she exists. It stays on the same page if it doesn't exist.



Sign Up Screen

The Sign up screen allows the user to provide basic information in order to create a new account, which links to the home page once completed. If the input does not satisfy any database constraints, it stays on the same page .

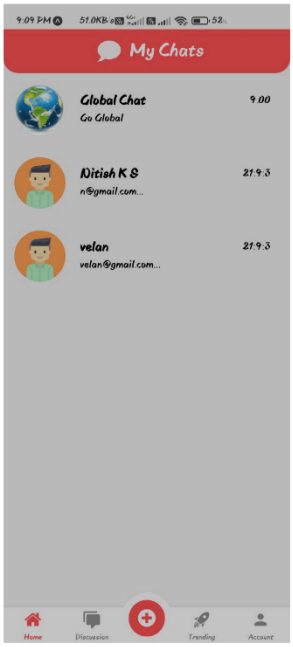


Home Screen

On the home screen, the posts of the mediaverse users are displayed. The post can be liked and a comment can be left on it. He can also see all of the comments and details that have been added to the post.

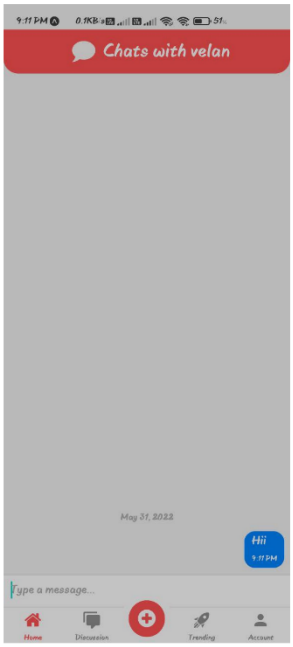
Add Post Screen

To create a post, the add post screen looks for an image component, a caption, and a hashtag. If successful, the post is published and can be accessed by anybody who has access to the mediaverse. If he is unsuccessful, he will remain on the same page with corresponding errors displayed.



All Chat Screen

The chat screen displays a list of mediaverse users who are friends with the current user. The present user can either go to global chat and communicate with everyone in the mediaverse, or he can go to private chat and chat with only his friends. The firebase firestore database stores all chat messages.

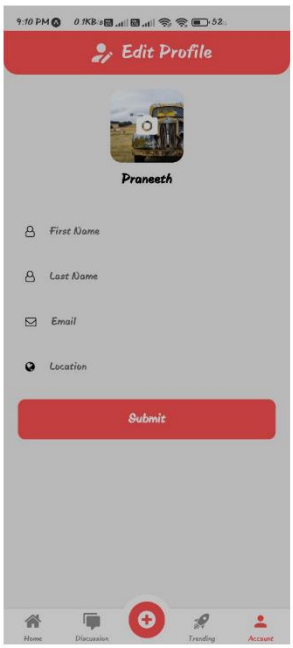


Private Chat Screen

The user can speak privately with his friends using the private chat interface. All chats are saved in a firestore with timestamps and retrieved when the user enters the private chat.

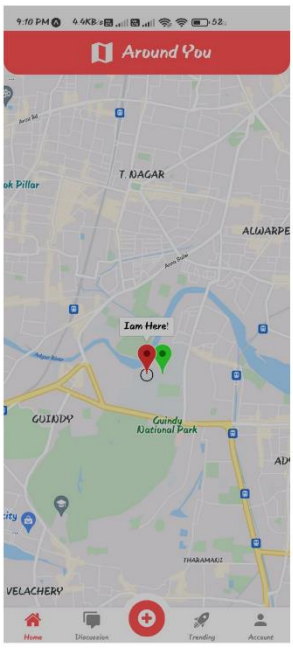
Account Screen

The account screen displays all relevant information about the current user, such as the number of followers, followers, posts, and so on. the user can view all post he posted in mediaverse.



Edit Profile Screen

Users can change their profile picture, username, and other account information on the edit profile screen.

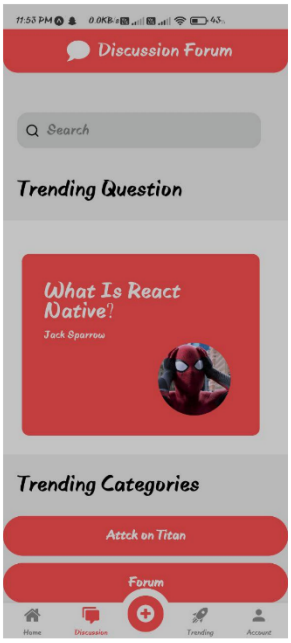


Map Screen

The map screen takes the user's location as input and displays his current position as well as all of his friends' locations. When the user's location changes, the location is dynamic and varies over time. This allows the present user to be aware of all of his friends' whereabouts.

Trending Screen

The trending screen shows posts that have been popular in the last several days, as well as posts that have been trending with hashtags.



Discussion Forum Screen

The discussion forum screen is similar to quora and stackoverflow sites in that it allows users to pose questions that can be addressed by anybody in the mediaverse. For the convenience of the users, the hot topics and questions based on categories are grouped together and displayed on the discussion forum screen.