Documentation

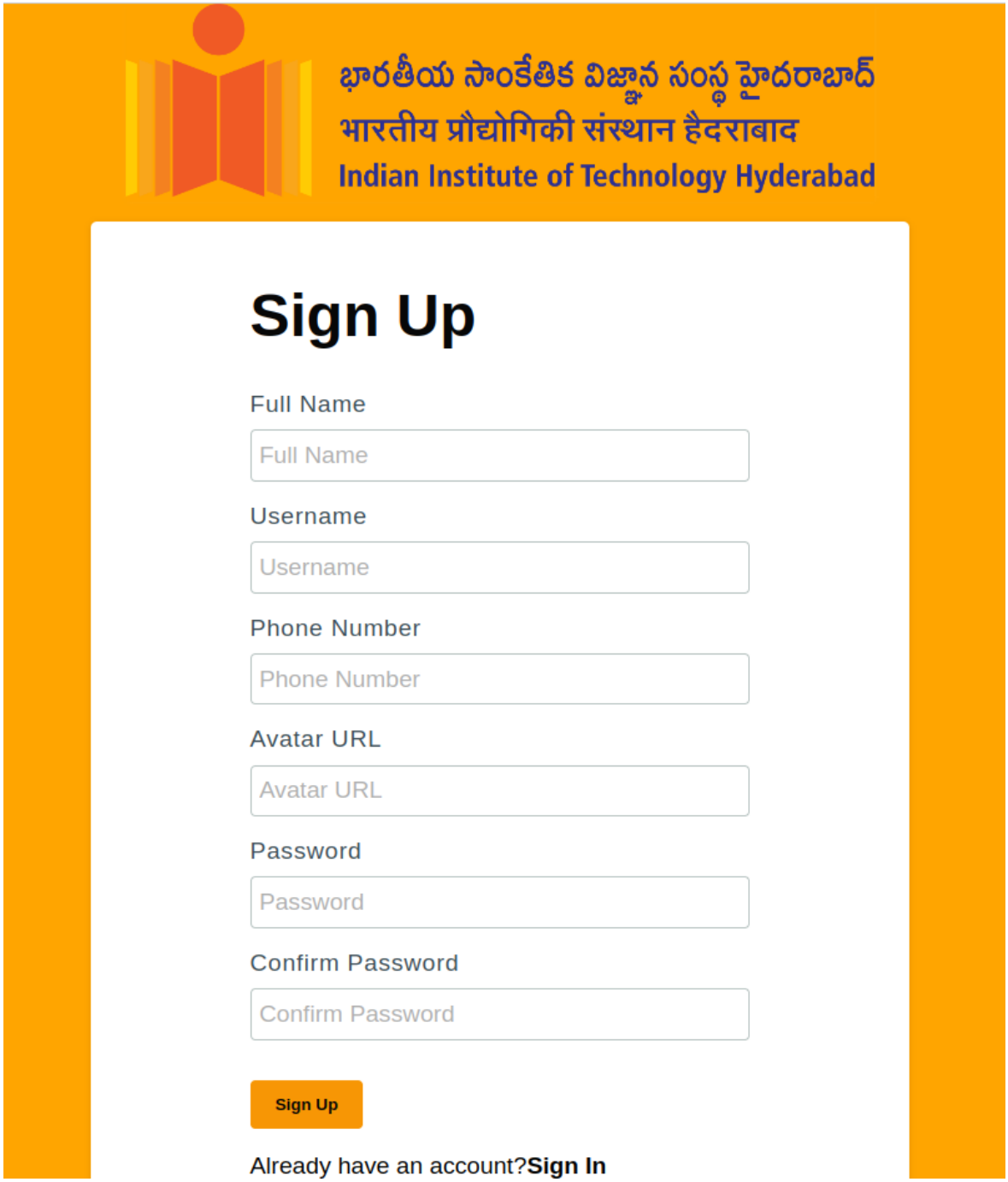
Name - Anushka Garg
Email - ep21btech11029@iith.ac.in

Its a community chat Application which will help the students to interact with each other, they can have group chats, personal chats, send images, reactions to the messages and lot more

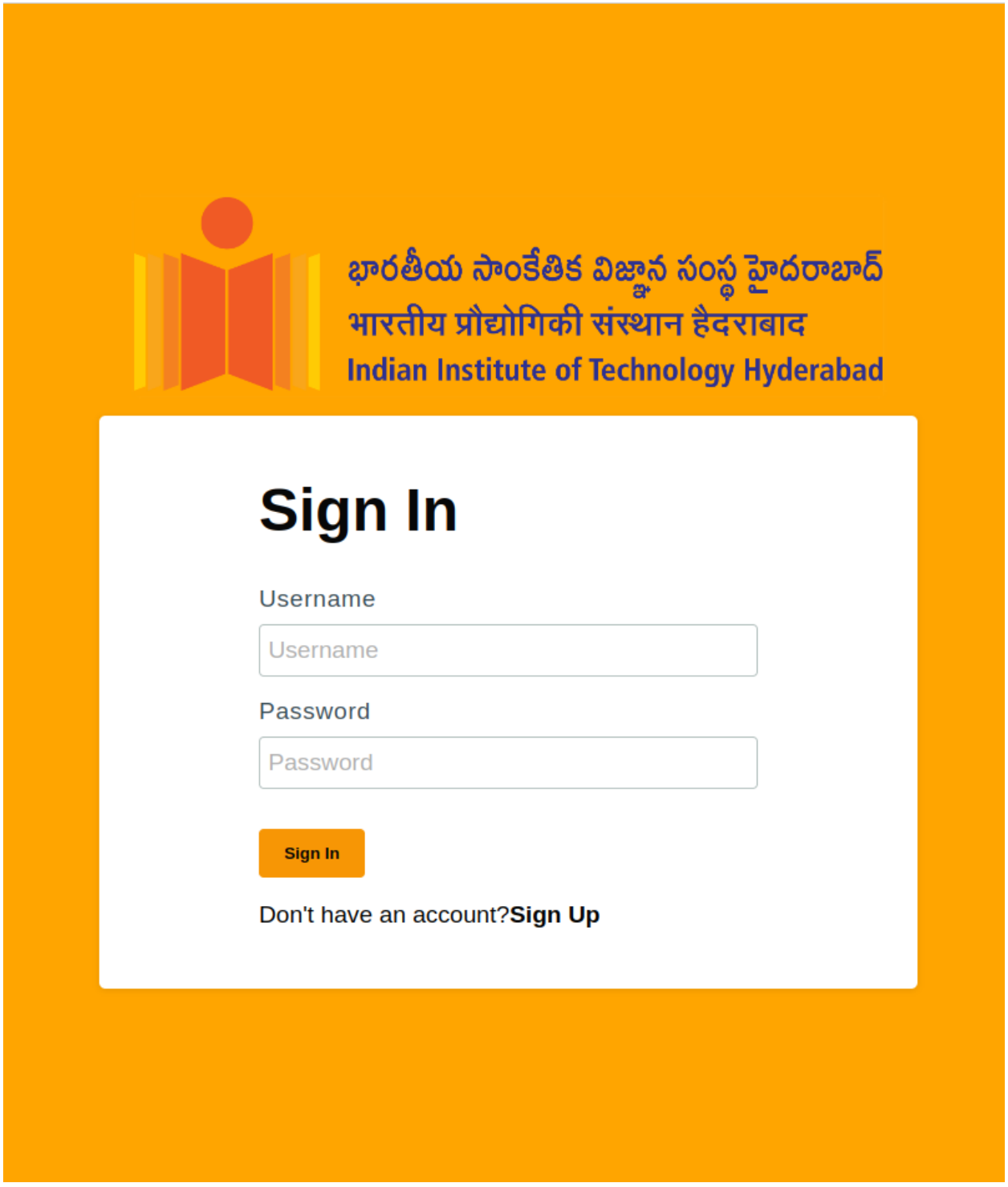
System Design:

Login/Signup-

1. Login/Signup:
   The first thing the user sees when they enter the url "http://localhost:3000" in their web browser is the Signup page. The user can enter the details to sign up and create their account if they don't have one. The Sign-up fields include:
   a. Full name of user
   b. Username for chat application
   c. Phone number
   d. Avatar URL link (not compulsory)
   e. Password field
   f. Confirm Password field

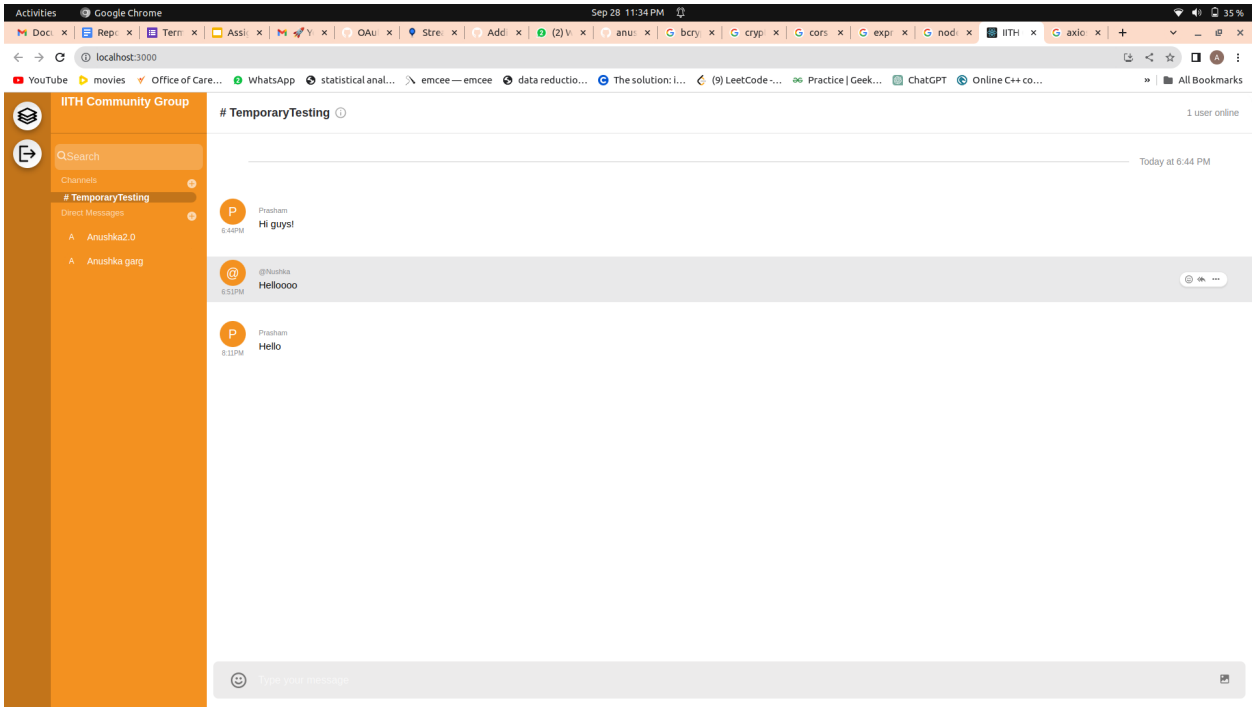


2. If the user already has an account, they can click on Sign In option below the Signup form. The Sign In form has the following fields:
    a. Username
    b. Password field

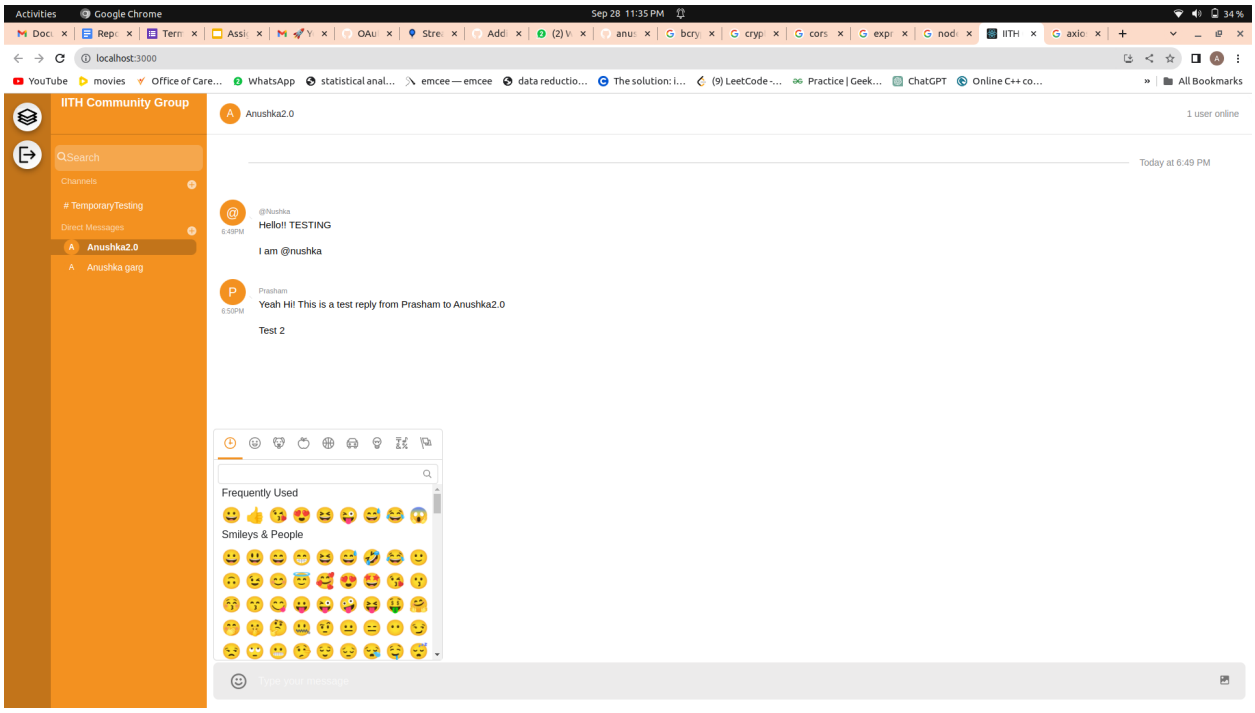


Chat Features -

3. Channel Lists -
    a. Group chat feature
    b. Send real time messages
    c. Reactions to the messages
    d. Sending emojis, images



4. Direct Messages -
    a. Chat Directly to a user
    b. Similar features like in group chats.



How to Run:

1. https://github.com/anushka1201/IITHcommunity_chatapp clone the code from here.
2. To download all the necessary things use npm i before other commands.
3. Start the server by using command - `npm i and then node index`

```
anushka@anush:~/workspace/chatappclone/project_medical_pager_chat$ cd server/
anushka@anush:~/workspace/chatappclone/project_medical_pager_chat/server$ node index
body-parser deprecated undefined extended: provide extended option index.js:18:17
Server running on port 5000
```

4. Host the app on local server by changing directory to client and run command - `npm i and then npm start`

```
anushka@anush:~/workspace/chatappclone/project_medical_pager_chat$ cd client/
anushka@anush:~/workspace/chatappclone/project_medical_pager_chat/client$ npm start
```

Technologies used

- Nodejs
- ReactJS
- CSS
- GetStream.io (StreamChat)

I have used NodeJS backend server for user authentication and sign up, with the help of GetStream.io (StreamChat API's), with ReactJS for frontend.

Dependencies and Libraries Used :

- Axios - used to make HTTP requests from node.js
- Bcrypt - used to hash the password
- Crypto - randomly generate userID to assign it to a user
- Express - writing handler from different type of http
- Nodemon - to refresh/restart the app with changes made in the code

- Cors - helps server to indicate any origins other than its own from which a browser should permit loading resource

<u>Authentication backend code</u>

Sign-up:

The sign-up form input is read from the frontend and is sent to the backend server through REST API, using an axios POST request to the server, running on port 5000. The route for signup is “/auth/signup”.

In the backend, the server receives the form inputs through the request body sent through the POST request. The password is hashed using bcrypt. The userId is randomly generated (using crypto), and a user token is created.

The token, along with the hashed password and other form fields are returned by the server. The token, along with other user details are saved in the browser cookie. Only if the authToken is present in the cookie can the user go beyond the signup/login page.

The user details are read from the cookie and using client.connect() method of Stream Chat API, initiates a connection to Stream's server infrastructure, and Stream's system will handle routing and managing the connection appropriately to provide real-time chat functionality for the application.

Log-in:

Similar to signup, form input is read from the frontend and is sent to the backend server through REST API, using an axios POST request to the server, running on port 5000. The route for login is “/auth/login”.

Using StreamChat API to query the users in the database using the username provided through the request body, we get the user instance. If the user is present in the database, then we use bcrypt to compare the hashed password of the user and the password provided through the request body after hashing. If they are the same then the user is authenticated, and the user token is created (same as in signup) and sent back to the frontend. The same process follows (storing details in cookies,

reading from cookies and connecting to Stream using StreamChat API, etc.)