Smart chat

GitHub link: https://github.com/ashutoshc8101/chatroom
Deployment Link: http://13.235.177.229/
Demo Video: Video Link

Overview:
Smart chat is a chatroom where users can interact with the room through text messages. It supports the censoring of toxic words. The following category of messages is censored by the toxicity model integrated into the chat app, identity attacks, insults, obscene, severe toxicity, sexually explicit, and threats. It also provides functions such as writing a new message by speaking (speech to text) and auto-reading new letters (text to speech). The speech-to-text transcription service is provided by google cloud.

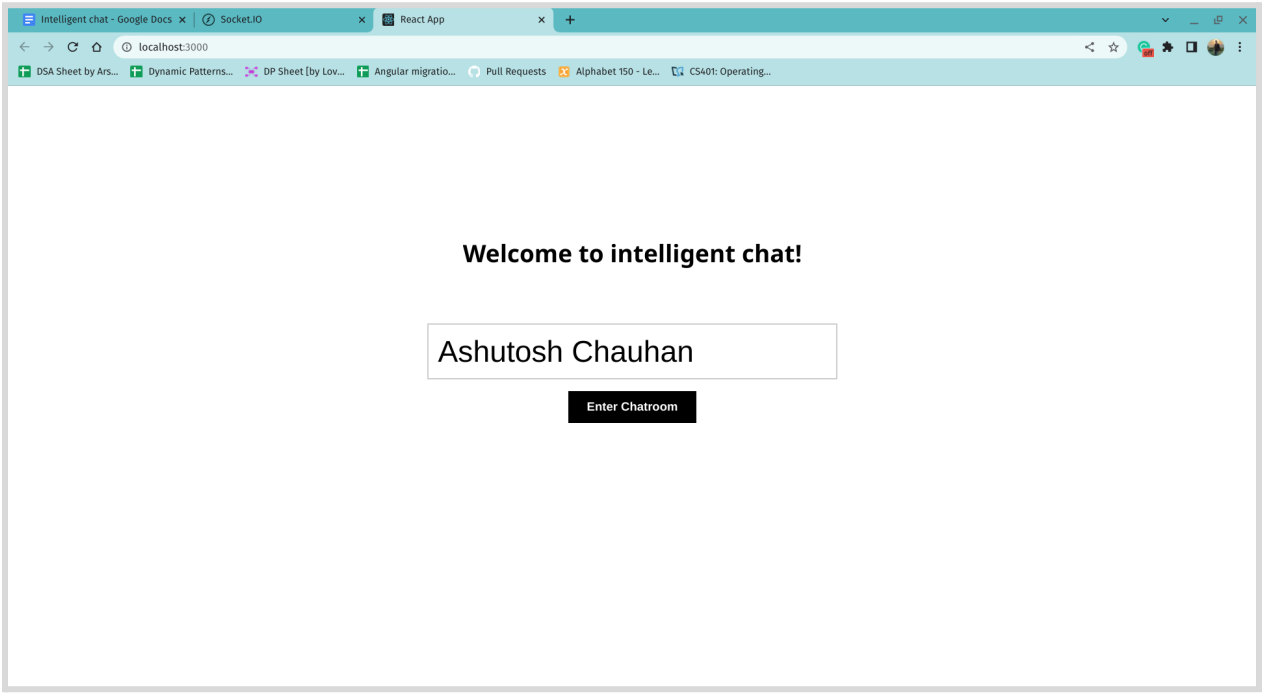
Tech Stack:
Reactjs, Node.js, MongoDB, TensorFlow, and google cloud platform.

Libraries Used:
Redux, react-router, styled-components, tensorflow.js, and socket.io

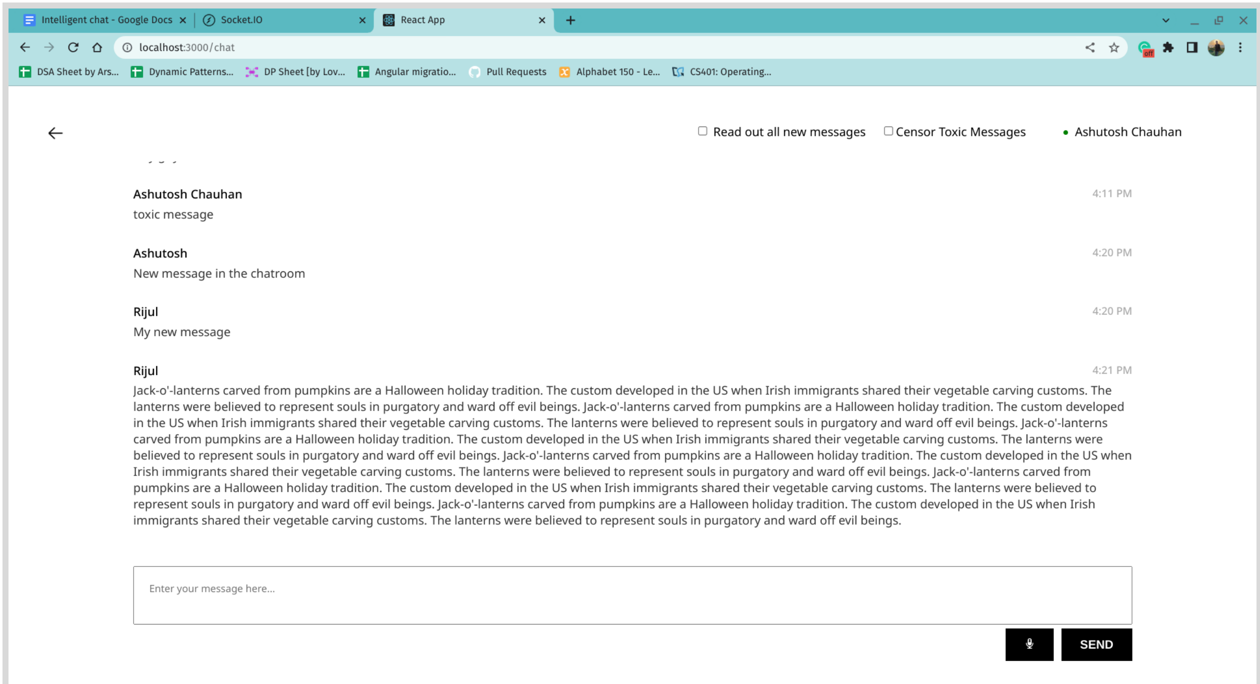
Functionalities:

1. Login Page:

This page allows the user to log in to the chatroom by providing a chat handle (nickname).



2. Chatroom:



The chatroom page has the following three sections:

1. Header:
    a. First, there is a back button that allows the user to log out of the chatroom and goes back to the login page.
    b. There is a checkbox to enable auto text-to-speech functionality, which announces all the new messages in the chatroom.
    c. Censor toxic messages run the classification model on all the messages in the chatroom and hide those which are found to be insulting and derogatory. This is done using a text toxicity detection model from tensorflow.js.
    d. To the right of the checkbox, there is a circular indicator that indicates the connectivity of the chat server. The ***green*** color indicates the client and socket server is connected and the ***red*** indicates that there is some problem with the connection.
2. The central section:
    a. The central section shows all the messages in the room, it shows the nickname of the author, the timestamp of the posted message, and the body of the message. This section auto-scrolls whenever a new message is received.
3. New message section:
    a. The text field allows the user to type a new message to be sent.
    b. The mic button allows the user to write his message using voice. This is achieved using speech-to-text ML service from the google cloud platform. A pipeline is set up from the browser to the node backend using WebSockets and the steam is further redirected to google cloud using

@google-cloud/speech library. The transcriptions received are then redirected back to the browser.

c. Send button allows the user to send his typed/spoken message to the chatroom. The messages sent are stored on a MongoDB server (NoSQL) and MongoDB Atlas is used to host the database.

The web tool is deployed on AWS ec2 instance.

Technical Design:

1. The toxicity model is deployed on the frontend because it is lightweight (< 5 MB), and takes about 2-3 seconds to load and 1-2 seconds to classify the messages.
2. Deploying it on the frontend, distributes the computation load among the client devices. On the other hand, if it is deployed on the backend, then classification for all users had to be done on the server machine, which will increase the computation load on the server and the user has to face latency as the message has to be sent to the server for classification.
3. Speech-to-text is not a lightweight model and is not available for the browser. But it can be converted manually which will result in 300 MB being loaded to the browser. Downloading 300 MB for each load of the website is not optimal and will produce very slow load times. It could have been deployed on the backend but there exist cloud services that provide very accurate transcription services such as google cloud's speech-to-text. Hence, it is used.
4. To use speech-to-text API, the client has to be authenticated but the browser is not supported as a client by the API. Hence, a pipeline is set up from the browser to the node.js backend which acts as the client for the API. The browser opens an audio stream to the node.js server using a socket connection. The backend further redirects the stream to the google cloud and redirects back the transcription received to the user.
5. MongoDB (NoSQL) is used for storing all the messages sent in the chatroom. A NoSQL database is used because the data stored is very simple and does not require establishing relationships between different schemas. Hence, as relationships are not required, a NoSQL database was found to be more scalable and will provide better performance. MongoDB Atlas is managed cloud database service which provides automated security features, built-in replication, backups, and fine-grained monitoring of data. The chat data should be stored securely and should not be lost, MongoDB Atlas was selected for hosting the database.