Group Members: Khoa Diep(ID:28730583), Man Ting Tang (ID:029677412)
CECS 327

Project 2 : Chat Web Application Using Client-Server Architecture

Contributors:

1. Khoa Diep
    a. Project Report
    b. Backend Server
    c. Video output:
2. Min Tang Ting
    a. Fronted Client
    b. Presentation
    c. Readme

Node.js & React.js Chat Web Application Setup

Prerequisites

Before starting, ensure you have the following installed:

- Node.js
- npm (Node Package Manager)
- A REST Client extension for your IDE or code editor (for testing API endpoints).
- VS Code (Installed Extension Rest Client for testing the server)

Step 1: Project Initialization

1. Create a Project Directory:
    - Create and navigate into a new directory for the project:

```
mkdir nodejs-reactjs-cecs327-chat
cd nodejs-reactjs-cecs327-chat
```

2. Initialize Node.js Backend:

- Create a backend subdirectory and initialize a Node.js project within it:

```
mkdir backend
cd backend
npm init -y
```

Step 2: Install Dependencies

Install the necessary npm packages:

```
npm install express cors axios
```

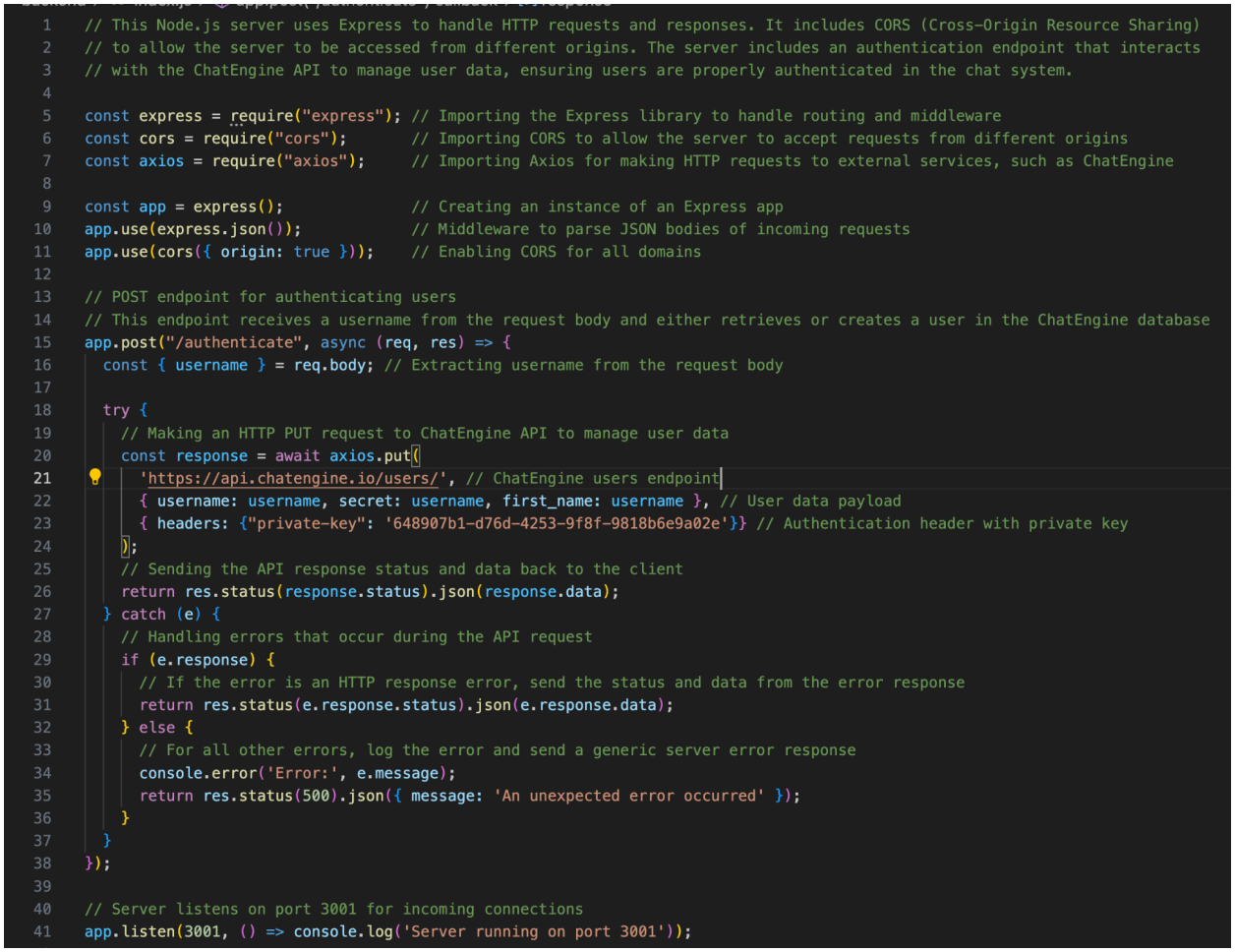
- Express: A web framework for Node.js
- CORS: Middleware to enable cross-origin resource sharing.
- Axios: A promise-based HTTP client for making requests to external APIs.

Additionally, install 'nodemon' as a development dependency:

```
npm install --save-dev nodemon
```

Step 3: Create the index.js (Backend - Node.js server using Express)

This Node.js server, built with the Express framework, handles HTTP requests efficiently and employs CORS to facilitate secure cross-domain requests, important for separating frontend and backend hosting. It uses axios to interact with the ChatEngine API for chat functionality, focusing on an authentication endpoint that manages user access by either fetching existing details or registering new users. The server is configured to handle JSON data smoothly and includes robust error handling within its authentication route to ensure reliability. Listening on port 3001, it's optimized for local development but can be adapted for production use.



```
// This Node.js server uses Express to handle HTTP requests and responses. It includes CORS (Cross-Origin Resource Sharing)
// to allow the server to be accessed from different origins. The server includes an authentication endpoint that interacts
// with the ChatEngine API to manage user data, ensuring users are properly authenticated in the chat system.

const express = require("express"); // Importing the Express library to handle routing and middleware
const cors = require("cors");       // Importing CORS to allow the server to accept requests from different origins
const axios = require("axios");     // Importing Axios for making HTTP requests to external services, such as ChatEngine

const app = express();              // Creating an instance of an Express app
app.use(express.json());            // Middleware to parse JSON bodies of incoming requests
app.use(cors({ origin: true }));    // Enabling CORS for all domains

// POST endpoint for authenticating users
// This endpoint receives a username from the request body and either retrieves or creates a user in the ChatEngine database
app.post("/authenticate", async (req, res) => {
  const { username } = req.body; // Extracting username from the request body

  try {
    // Making an HTTP PUT request to ChatEngine API to manage user data
    const response = await axios.put(
      'https://api.chatengine.io/users/', // ChatEngine users endpoint
      { username: username, secret: username, first_name: username }, // User data payload
      { headers: {"private-key": '648907b1-d76d-4253-9f8f-9818b6e9a02e'}} // Authentication header with private key
    );
    // Sending the API response status and data back to the client
    return res.status(response.status).json(response.data);
  } catch (e) {
    // Handling errors that occur during the API request
    if (e.response) {
      // If the error is an HTTP response error, send the status and data from the error response
      return res.status(e.response.status).json(e.response.data);
    } else {
      // For all other errors, log the error and send a generic server error response
      console.error('Error:', e.message);
      return res.status(500).json({ message: 'An unexpected error occurred' });
    }
  }
});

// Server listens on port 3001 for incoming connections
app.listen(3001, () => console.log('Server running on port 3001'));
```

Step 4: Configure Scripts

Modify the 'package.json' to include a script for starting the server using 'nodemon':

```
"scripts": {
  "start": "nodemon index.js"
}
```

Step 5: Running the Server

To start the server, run:

```
npm run start
```

This will launch the Node.js server with '**nodemon'**, which automatically restarts the server upon file changes

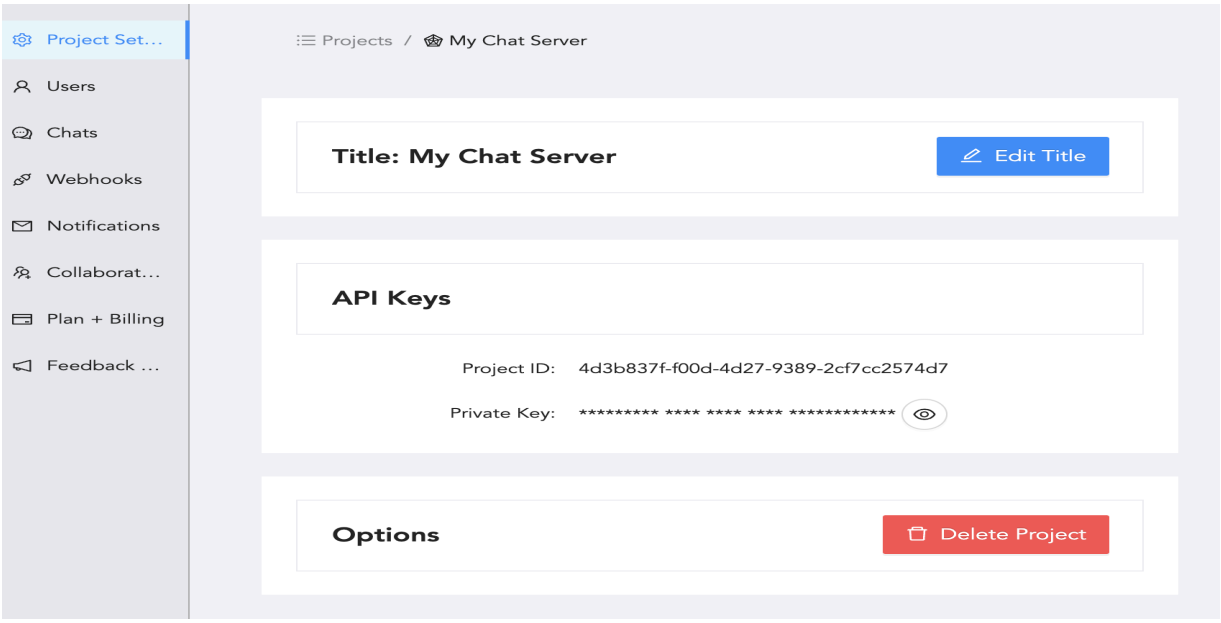
Step 6: Testing the Setup

Create a file named 'request.rest' in the backend folder to test the server's authentication endpoint:

```
POST http://localhost:3001/authenticate
Content-Type: application/json
{
  "username": "testuser"
}
```

Step 7: Integrate Chat Functionality

Register with ChatEngine.io and create a project. Use the provided API keys in the backend to authenticate users and manage chat functionalities. This allows all users in the ChatEngine project to communicate through the API and platform.



Step 8: Frontend Development with React

To start developing the frontend of the application, use Vite, a modern build tool for JavaScript applications, particularly well-suited for React projects.

1. Navigate to the Root Directory:
    - Open the terminal and change the directory back to the root of your project (if you are not already there) by running: `cd..`
    - Then make a new directory named **“fronted”** by running: `mkdir fronted`
2. Create a New React Project:

    Execute the following command to create a new React project using Vite. This command initializes a new project and allows us to specify React as your framework : `npm create vite@latest`
3. Install Dependencies

    Change to the newly created “**fronted”** directory and install all necessary dependencies: `cd fronted` then `npm install`
4. Modify the App.jsx file

    This component manages user authentication and the display of the chat interface. First it checks if a user is logged in; if not, it renders the **AuthPage** for user login. Once logged in, it switches to the **ChatsPage** to allow the user to participate in chat conversations. This component uses a React hook, **useState**, to track and update the user’s authentication status dynamically

```jsx
// Importing necessary React hook for state management
import { useState } from "react";

// Importing stylesheet for the application
import "./App.css";

// Importing components for authentication and chat pages
import AuthPage from "./AuthPage";
import ChatsPage from "./ChatsPage";

// Main App component
function App() {
  // State hook for managing user state, initially set to undefined
  const [user, setUser] = useState(undefined);

  // Conditional rendering based on user state
  if (!user) {
    // If there is no user, AuthPage is rendered to allow user authentication.
    // onAuth is a prop expecting a function that updates the user state
    return <AuthPage onAuth={(user) => setUser(user)} />;
  } else {
    // If there is a user, ChatsPage is rendered to display the chat interface.
    // user is passed as a prop to ChatsPage for user-specific operations
    return <ChatsPage user={user} />;
  }
}

// Exporting App component to be used in other parts of the application
export default App;
```

5. Create AuthPage.jsx Component

This component handles user authentication through a form that submits a username to a server using axios. On successful authentication, it triggers the **onAuth** prop with user data, enabling logged-in user functionality. The form includes a welcoming title, a subtitle for usernam input, and submit button. Errors during authentication are logged to the console. This component is crucial for managing access to application's features and is designed for ease of use with a clean, card-like interface.

frontend > src > AuthPage.jsx > AuthPage

```jsx
// Importing axios for HTTP requests
import axios from "axios";

// AuthPage component for handling the user authentication
const AuthPage = (props) => {
  // Handler for the form submission event
  const onSubmit = (e) => {
    e.preventDefault(); // Prevents the default form submit action

    // Extracting the value from the first input field of the form (assumed to be the username)
    const { value } = e.target[0];

    // Makes a POST request to the authentication endpoint with the provided username
    axios
      .post("http://localhost:3001/authenticate", { username: value })
      .then((r) => {
        // On successful authentication, calls the onAuth prop with the returned data and secret
        props.onAuth({ ...r.data, secret: value });
      })
      .catch((e) => {
        // Logs an error message if the authentication fails
        console.log("Auth Error", e);
      });
  };

  // The component renders a form that allows the user to set their username
  return (
    <div className="background">
      <form onSubmit={onSubmit} className="form-card">
        {/* Title displayed on the authentication card */}
        <div className="form-title">Welcome 👋</div>

        {/* Subtitle prompting the user to set their username */}
        <div className="form-subtitle">Set a username to get started</div>

        {/* Auth block containing the username label, input and submit button */}
        <div className="auth">
          <div className="auth-label">Username</div>
          <input className="auth-input" name="username" />
          <button className="auth-button" type="submit">
            Enter
          </button>
        </div>
      </form>
    </div>
  );
};

// Exporting AuthPage for use in other parts of the application
export default AuthPage;
```

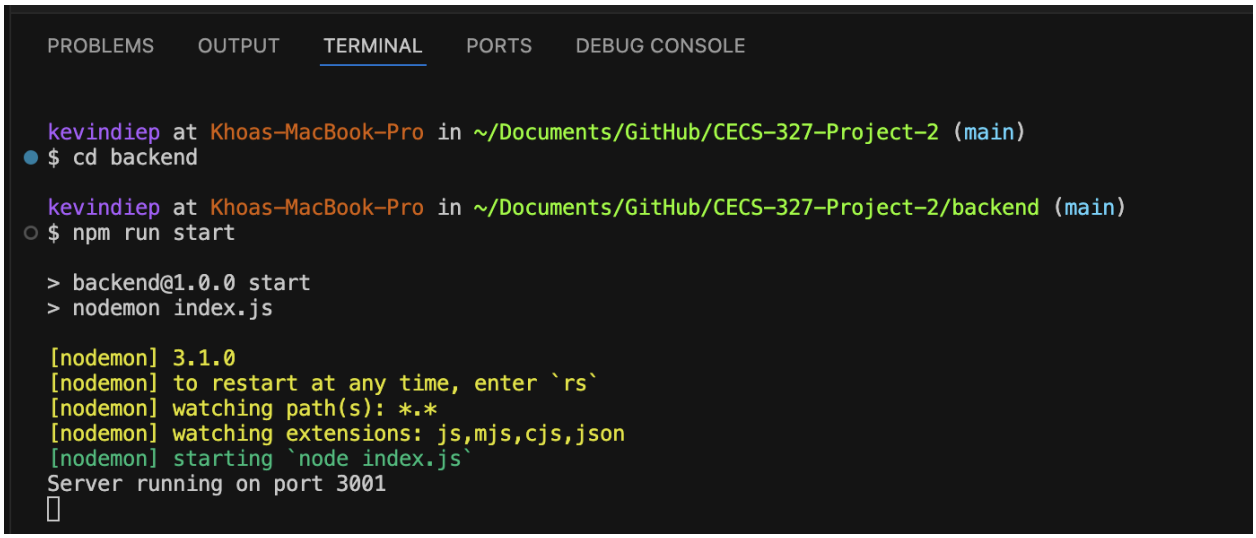
6. Create ChatsPage.jsx Component

The **ChatsPage** utilizes the **react-chat-engine-advanced** library to manage and display chat functionalities. It sets up chat properties using **useMultiChatLogic** hook with provided project ID, username and user secret (password) from **props**. The component establishes a websocket connection for live chat using **MultiChatSocket**, and display the chat interface with **MultiChatWindow**. This setup ensures a full viewport height for an immersive chat experience. The component is crucial for integrating advanced chat features into the application, and it requires installing the **react-chat-engine-advanced** library to function.
To install: npm install react-chat-engine-advanced

Step 9. Testing the project

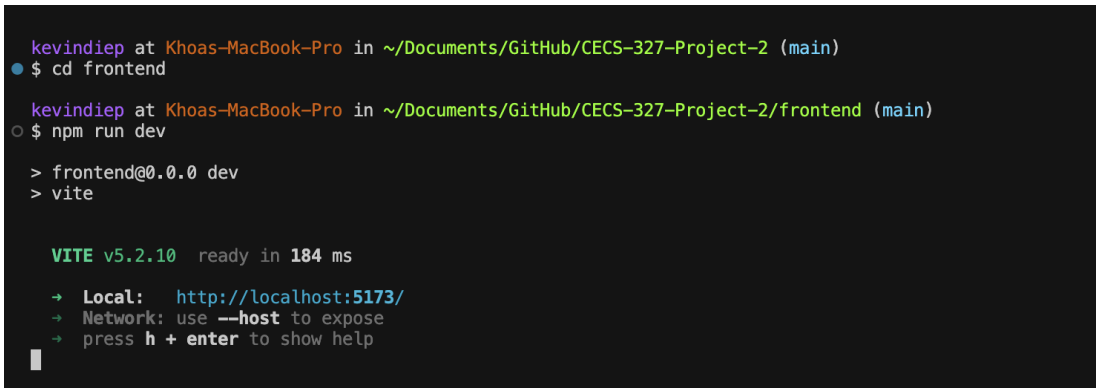
Turn on the server:

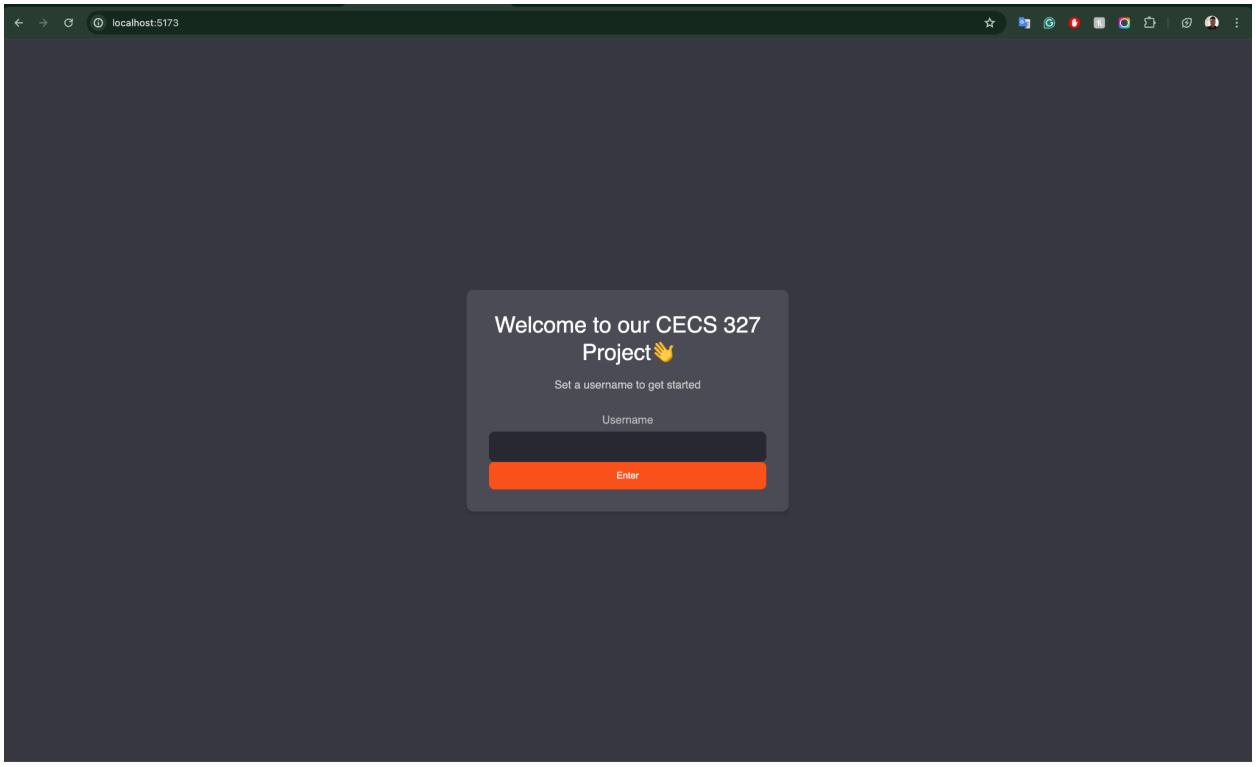
Open the terminal window , go to the backend directory and : npm run start to start the server.



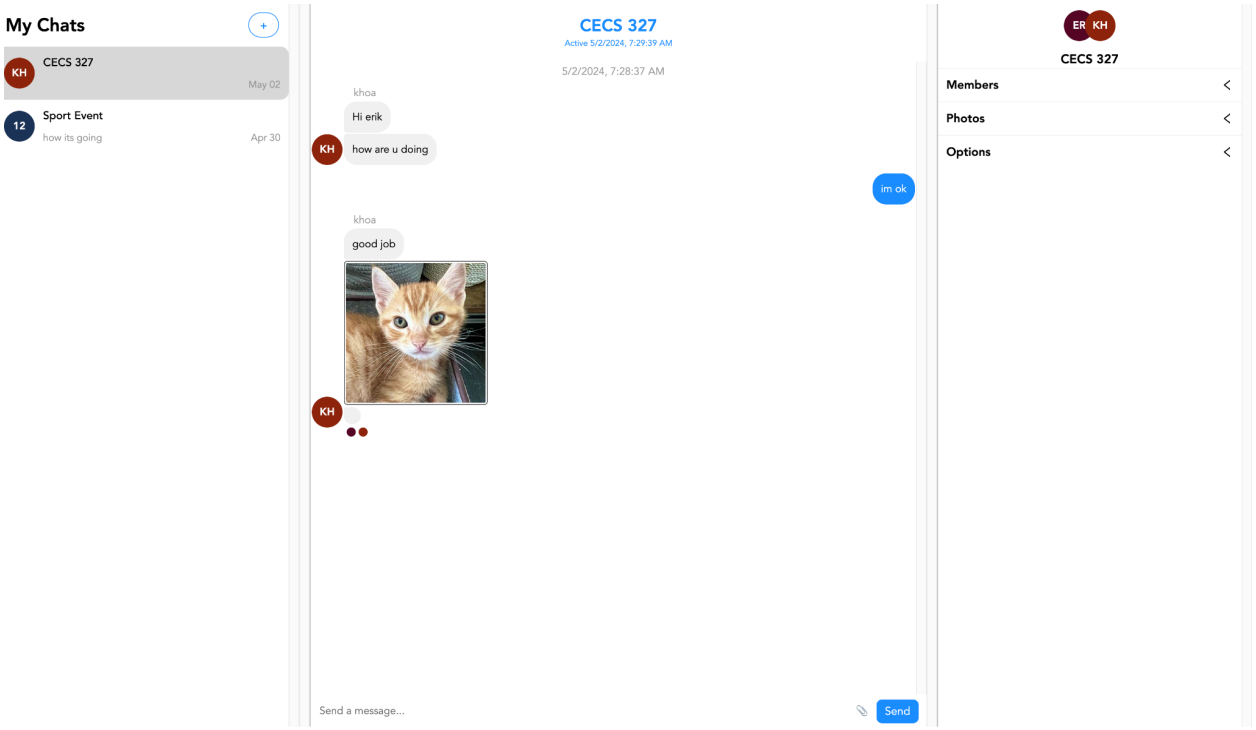
Open the React Client:

Open a new terminal, go to fronted directory and : npm run dev: and open the localhost on the browser





When the user enters their username:, they should able to chat with other, group message and send picture



References

https://chatengine.io/docs/react/v2/getting_started
https://chatengine.io/docs/react/v2/sockets
https://www.youtube.com/watch?v=SqcY0GlETPk&t=301s&ab_channel=ProgrammingwithMosh
https://chatengine.io/docs/react/v2