

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

Project 2: Chat Web Application Using Client-Server Architecture

California State University, Long Beach
College of Engineering
CECS 327, Section 02, Spring 2024
May 3, 2024

Table of Content

Introduction	03
Design	05
Implementation	07
Results	15
Exhaustive Analysis: Performance, Scaling Characteristics	16
Conclusion	17

Introduction

In the digital era, real-time communication has become paramount, fostering collaboration and connectivity across diverse platforms and communities. Leveraging the power of modern web technologies, our project endeavors to encapsulate this essence through the creation of a robust Chat Web Application using Client-Server Architecture. Developed by the adept collaboration of Khoa Diep and Man Ting Tang, this endeavor represents the culmination of our effort in CECS 327, Section 02, Spring 2024 at California State University, Long Beach.

This project optimizes innovation and application, intertwining a plethora of class-discussed concepts. With a primary focus on employing HTTP, WebSockets, Middleware, and REST API Interaction, our endeavor aims to transcend conventional boundaries, offering users an immersive and dynamic chatting experience.

At its core, our Chat Web Application harnesses the prowess of Node.js for the backend server, with the Express framework. This backend infrastructure not only handles HTTP requests efficiently but also seamlessly integrates with external services such as the ChatEngine API, fostering real-time communication among users. Meanwhile, the frontend, meticulously crafted using React.js, presents a sleek and intuitive interface, ensuring a seamless user experience.

This project is not merely a culmination of technical prowess but also a testament to meticulous planning and execution. From project initialization to the integration of advanced chat functionalities, every step has been crafted to ensure optimal performance, scalability, and reliability.

Contributors:

1. Khoa Diep
 - a. Project Report
 - b. Backend Server
 - c. Video output:
https://www.youtube.com/watch?v=RwjyDQ-tEEg&ab_channel=KhoaDi%E1%BB%87p
 - d. Github Repo for source code:
<https://github.com/Misax10/CECS-327-Project-2>
2. Min Tang Ting
 - a. Fronted Client
 - b. Presentation
 - c. Readme

B Level (Insight): This project adeptly applies a range of class-discussed concepts:

- 1) HTTP(Hypertext Transfer Protocol): Node.js server handles HTTP requests using the Express framework, which simplifies the creation and management of server routes. The Axios library will be used to make requests to the external API
- 2) WebSockets: provides full-duplex communication channels over a single long-lived connection. This allows the server and client to send messages back and forth as soon as data is available without having to re-establish a connection for each message.
- 3) Middleware: The use of middleware in Express: `express.json()` is crucial for managing the flow of data between the server and clients and between different parts of the server application itself.
- 4) REST API Interaction: the use of axios for making HTTP requests to external services(ChatEngine API) demonstrates interaction with RESTful APIs, which is a key component in modern distributed systems for enabling interoperability between different software applications.

Design

In crafting Chat Web Application using Client-Server Architecture, meticulous attention was given to both the backend and fronted structures, ensuring seamless integration and optimal performance.

Backend Structure

At the heart of our application lies the backend infrastructure, responsible for processing incoming requests, managing data, and facilitating communication between clients. Built using Node.js and the Express framework, our backend server embodies efficiency, scalability, and reliability

1. Node.js Server

- a. Node.js servers as the foundation of our backend, offering a non-blocking, event-driven architecture ideal for handling concurrent connections and real-time communication.

2. Express Framework:

- a. Leveraging the Express framework, we streamline the creation and management of server routes, middleware, and request handling. This enable us to build a robust and flexible backend architecture with minimal boilerplate code.

3. Middleware:

- a. Middleware plays a pivotal role in our backend structure, facilitating the flow of data between the server and clients, as well as between different parts of the server application itself. For example, **express.json()** middleware ensures seamless parsing of JSON data, while CORS middleware enables secure cross-origin resource sharing.

4. HTTP Request and REST API Interaction:

- a. Our backend interacts with external services, notably the ChatEngine API, to enable chat functionality. Ultizing the Axios library, we make HTTP requests to the ChatEngine API, adhering to RESTful principles for seamless integration and interoperability.

5. WebSocket Support

- a. Enable real-time communication between clients, our backend supports WebSockets, providing full-duplex communication channels over a single long-lived connection.

Fronted Structure

Complementing the backend, our fronted interface embodies intuitiveness, responsiveness, and aesthetic appeal, offering users a seamless chatting experience

1. **React.js Framework:**

- a. React.js serves as the cornerstone of our fronted development, providing a declarative and component-based approach to build user interfaces. With React, we can efficiently manage state, handle user interactions, and dynamically update the UI as needed

2. **Component-based Architecture:**

- a. Our fronted is structured around reusable and composable components, each encapsulating specific functionality and rendering logic. This modular approach enhances maintainability, scalability, and code reusability.

3. **User Authentication and Authorization:**

- a. User Authentication allows user to securely login and access chat functionalities. Through forms and authentication endpoints, user can authenticate their identity and gain access to personalized chat experiences.

4. **Real-time Chat Interface:**

- a. Leveraging libraries such as **react-chat-engine-advanced**, our fronted offers a feature-rich chat interface, complete with real-time messaging, group chat support and multimedia sharing capabilities such as images and files. WebSocket connections enable instant message delivery, fostering interactive and engaging conversations.

5. **Responsive Design:**

- a. Our fronted interface is designed with responsiveness in mind, ensuring optimal viewing and interaction experiences across devices of varying screen sizes. Through media queries and flexible layouts, we accommodate users accessing the application from desktops, tablets, and smartphone alike.

Implementation

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.

```
1 // This Node.js server uses Express to handle HTTP requests and responses. It includes CORS (Cross-Origin Resource Sharing)
2 // to allow the server to be accessed from different origins. The server includes an authentication endpoint that interacts
3 // with the ChatEngine API to manage user data, ensuring users are properly authenticated in the chat system.
4
5 const express = require("express"); // Importing the Express library to handle routing and middleware
6 const cors = require("cors"); // Importing CORS to allow the server to accept requests from different origins
7 const axios = require("axios"); // Importing Axios for making HTTP requests to external services, such as ChatEngine
8
9 const app = express(); // Creating an instance of an Express app
10 app.use(express.json()); // Middleware to parse JSON bodies of incoming requests
11 app.use(cors({ origin: true })); // Enabling CORS for all domains
12
13 // POST endpoint for authenticating users
14 // This endpoint receives a username from the request body and either retrieves or creates a user in the ChatEngine database
15 app.post("/authenticate", async (req, res) => {
16   const { username } = req.body; // Extracting username from the request body
17
18   try {
19     // Making an HTTP PUT request to ChatEngine API to manage user data
20     const response = await axios.put(
21       'https://api.chatengine.io/users/', // ChatEngine users endpoint
22       { username: username, secret: username, first_name: username }, // User data payload
23       { headers: { "private-key": '648907b1-d76d-4253-9f8f-9818b6e9a02e' } } // Authentication header with private key
24     );
25     // Sending the API response status and data back to the client
26     return res.status(response.status).json(response.data);
27   } catch (e) {
28     // Handling errors that occur during the API request
29     if (e.response) {
30       // If the error is an HTTP response error, send the status and data from the error response
31       return res.status(e.response.status).json(e.response.data);
32     } else {
33       // For all other errors, log the error and send a generic server error response
34       console.error('Error:', e.message);
35       return res.status(500).json({ message: 'An unexpected error occurred' });
36     }
37   }
38 });
39
40 // Server listens on port 3001 for incoming connections
41 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

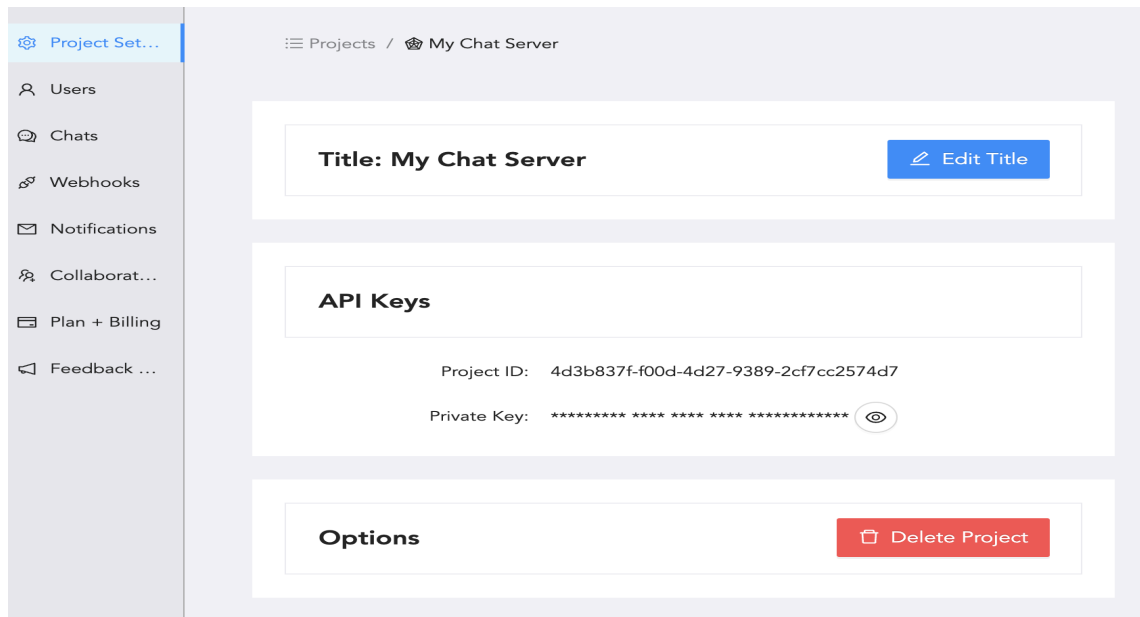
Step 6: Testing the Setup

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 frontend`

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 frontend` 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

```
1 // Importing necessary React hook for state management
2 import { useState } from "react";
3
4 // Importing stylesheet for the application
5 import "./App.css";
6
7 // Importing components for authentication and chat pages
8 import AuthPage from "./AuthPage";
9 import ChatsPage from "./ChatsPage";
10
11 // Main App component
12 function App() {
13   // State hook for managing user state, initially set to undefined
14   const [user, setUser] = useState(undefined);
15
16   // Conditional rendering based on user state
17   if (!user) {
18     // If there is no user, AuthPage is rendered to allow user authentication.
19     // onAuth is a prop expecting a function that updates the user state
20     return <AuthPage onAuth={(user) => setUser(user)} />;
21   } else {
22     // If there is a user, ChatsPage is rendered to display the chat interface.
23     // user is passed as a prop to ChatsPage for user-specific operations
24     return <ChatsPage user={user} />;
25   }
26 }
27
28 // Exporting App component to be used in other parts of the application
29 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 username input, and a submit button. Errors during authentication are logged into the console. This component is crucial for managing access to the application's features and is designed for ease of use with a clean, card-like interface.

```

frontend > src > AuthPage.jsx > AuthPage
1  // Importing axios for HTTP requests
2  import axios from "axios";
3
4  // AuthPage component for handling the user authentication
5  const AuthPage = (props) => {
6    // Handler for the form submission event
7    const onSubmit = (e) => {
8      e.preventDefault(); // Prevents the default form submit action
9
10     // Extracting the value from the first input field of the form (assumed to be the username)
11     const { value } = e.target[0];
12
13     // Makes a POST request to the authentication endpoint with the provided username
14     axios
15       .post("http://localhost:3001/authenticate", { username: value })
16       .then((r) => {
17         // On successful authentication, calls the onAuth prop with the returned data and secret
18         props.onAuth({ ...r.data, secret: value });
19       })
20       .catch((e) => {
21         // Logs an error message if the authentication fails
22         console.log("Auth Error", e);
23       });
24   };
25
26   // The component renders a form that allows the user to set their username
27   return (
28     <div className="background">
29       <form onSubmit={onSubmit} className="form-card">
30         { /* Title displayed on the authentication card */ }
31         <div className="form-title">Welcome 🙌</div>
32
33         { /* Subtitle prompting the user to set their username */ }
34         <div className="form-subtitle">Set a username to get started</div>
35
36         { /* Auth block containing the username label, input and submit button */ }
37         <div className="auth">
38           <div className="auth-label">Username</div>
39           <input className="auth-input" name="username" />
40           <button className="auth-button" type="submit">
41             Enter
42           </button>
43         </div>
44       </form>
45     </div>
46   );
47 };
48
49 // Exporting AuthPage for use in other parts of the application
50 export default AuthPage;
51

```

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 the provided project ID, username, and user secret (password) from **props**. The component establishes a WebSocket connection for live chat using **MultiChatSocket** and displays the chat interface with **MultiChatWindow**. This

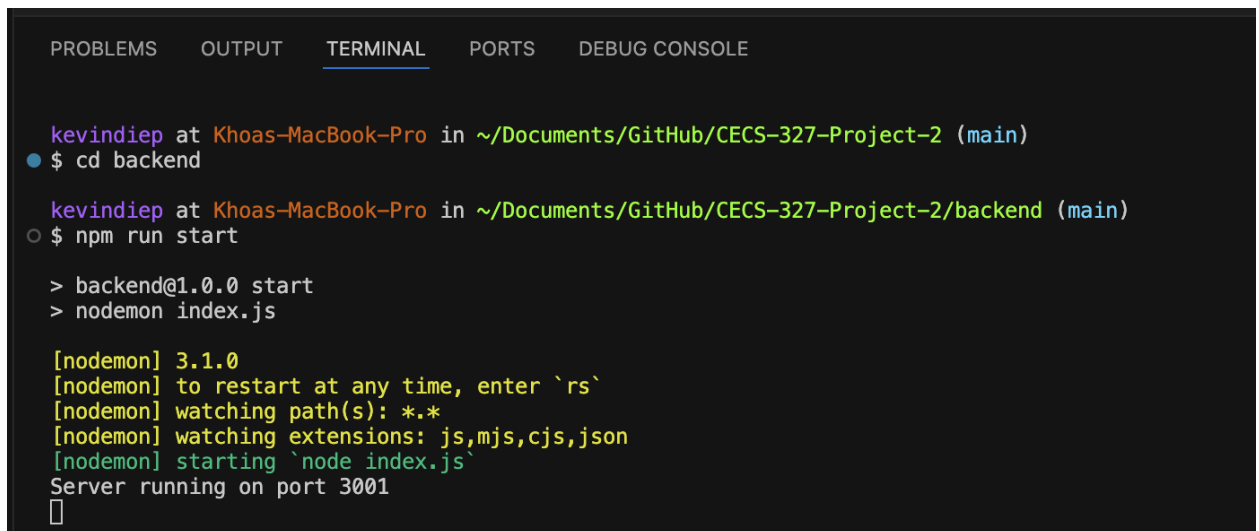
setup ensures a full viewport height for an immersive chat experience. The component is crucial for integrating advanced chat features into the application, and it requires installing the **react-chat-engine-advanced** library to function.

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.



```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE

kevindiep at Khoas-MacBook-Pro in ~/Documents/GitHub/CECS-327-Project-2 (main)
● $ cd backend

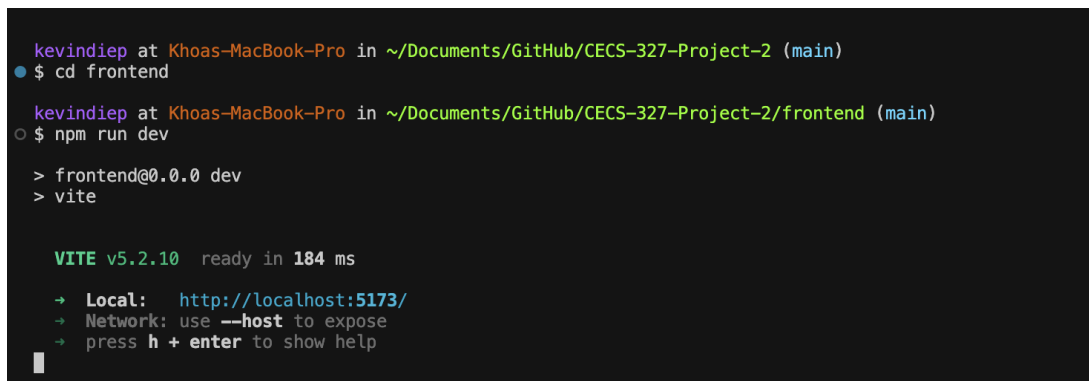
kevindiep at Khoas-MacBook-Pro in ~/Documents/GitHub/CECS-327-Project-2/backend (main)
○ $ npm run start

> backend@1.0.0 start
> nodemon index.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server running on port 3001
█
```

Open the React Client:

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



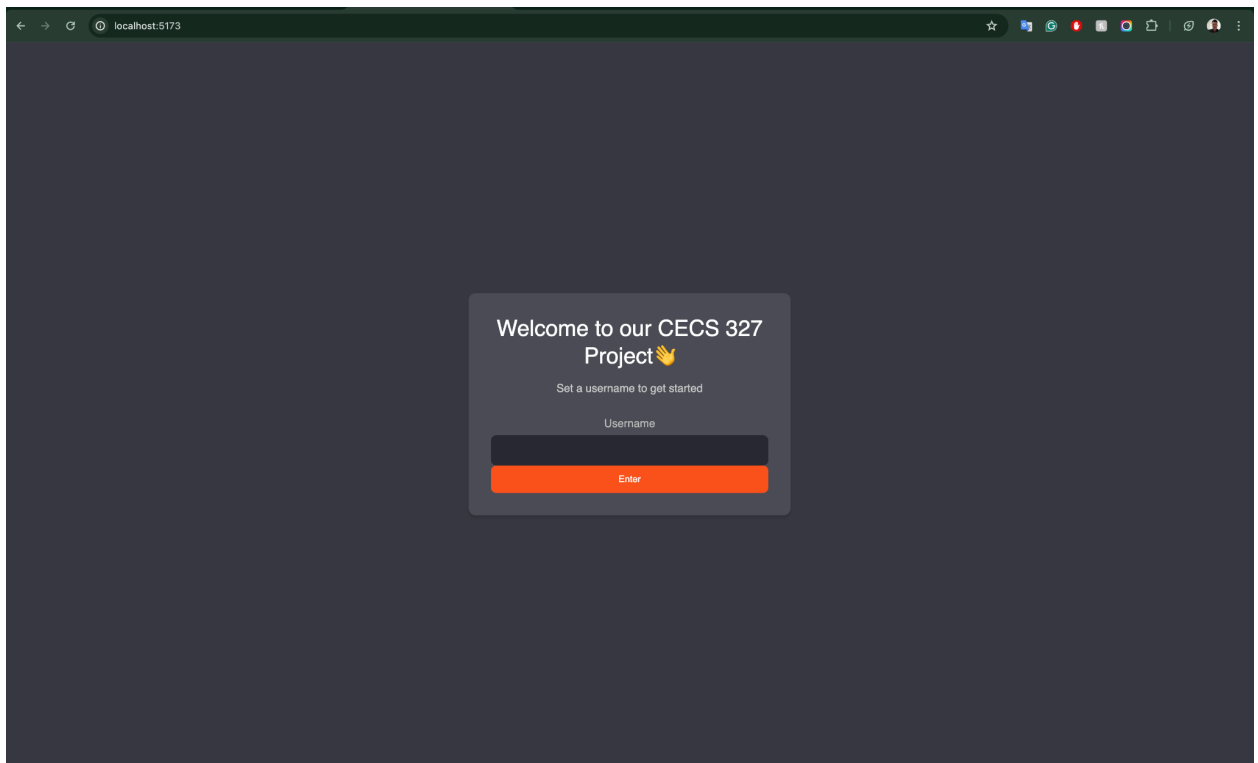
```
kevindiep at Khoas-MacBook-Pro in ~/Documents/GitHub/CECS-327-Project-2 (main)
● $ cd frontend

kevindiep at Khoas-MacBook-Pro in ~/Documents/GitHub/CECS-327-Project-2/frontend (main)
○ $ npm run dev

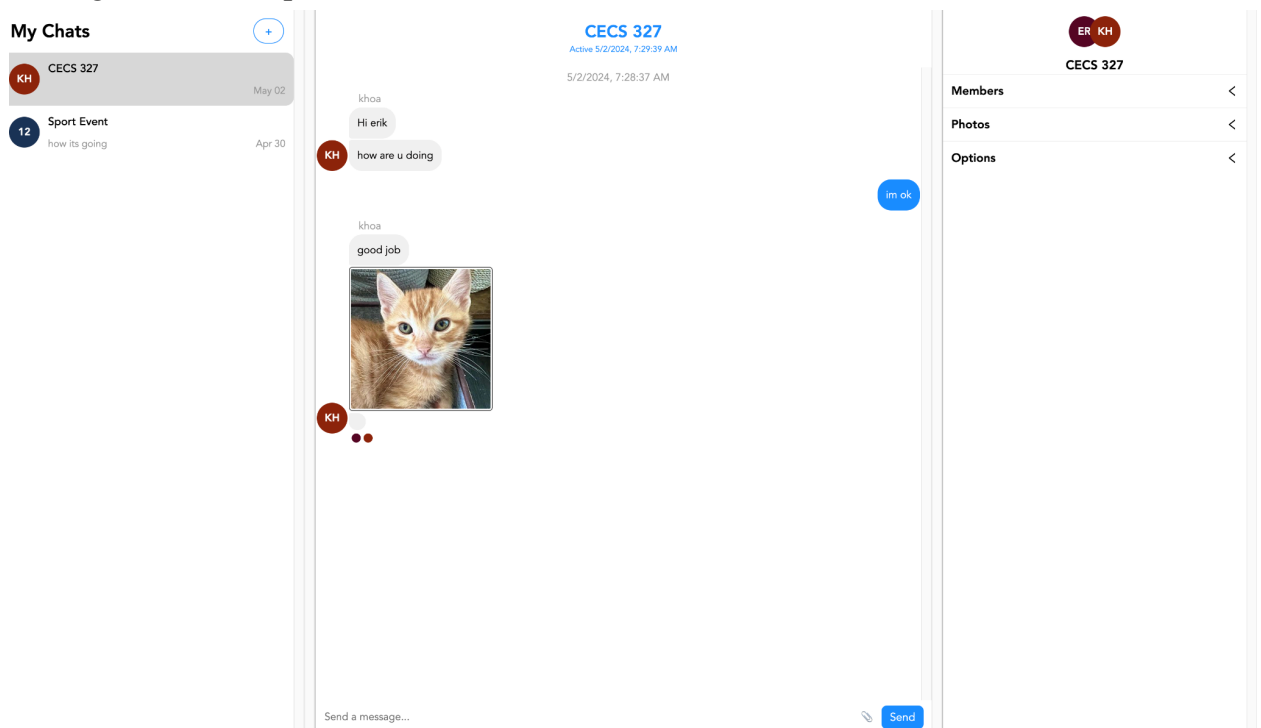
> frontend@0.0.0 dev
> vite

VITE v5.2.10 ready in 184 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
█
```



When the user enters their username: they should be able to chat with others, group messages, and send pictures



Results

After meticulous planning, diligent implementation, and rigorous testing, we are pleased to announce the successful deployment and operation of our Chat Web Application using Client-Server Architecture. The application has undergone extensive testing across various scenarios, ensuring its robustness, reliability, and user-friendliness.

1. Successful Application Launch:

- a. The Chat Web Application has been successfully launched and is accessible to users via their web browsers. Upon accessing the application, users are greeted with a sleek and intuitive interface that facilitates a seamless navigation and interaction.

2. User Authentication and Access Control:

- a. Users can securely authenticate their identity using the provided authentication mechanism. Users can gain access to personalized chat functionalities, including the ability to initiate one-one-one conversations, create group chats, and participate in existing chat rooms.

3. Real-Time Messaging:

- a. The core functionality of real-time messaging has been seamlessly integrated into the application. Users can exchange messages in real-time, experiencing minimal latency and delivery of messages. This real-time communication capability enhances user engagement and fosters interactive conversations.

4. Group Chat Support:

- a. Users have the option to create group chats, enabling collaboration and communication among multiple participants. Group chat functionality facilitates efficient information sharing, brainstorming, and decision-making among team members or communities.

5. Multimedia Sharing:

- a. Our Chat Web Application supports multimedia sharing, allowing users to send pictures, files and other media within chat conversations. This feature enhances the richness and versatility of communication, enabling users to convey information and express themselves effectively.

6. Smooth User Experience:

- a. Throughout the testing phase, users have reported a smooth and seamless experience while interacting with the application. The intuitive user interface, coupled with responsive design principles, ensures optimal usability and accessibility across devices of varying screen sizes.

Exhaustive Analysis: Performance, Scaling Characteristics:

1. Performance Analysis:

a. Server-Side Performance:

- i. **Response Time Optimizaiton:** Through efficient route handling and asynchronous processing, we aim to minimize response times for incoming requests. This involves optimizing database queries, reducing I/O operations, and leveraging caching mechanisms to retrieve and serve data swiftly.
- ii. **Resource Utilization:** Monitoring server resource utilization metrics such as CPU usage, memory consumption, and disk I/O helps identify performance bottlenecks and optimize resource allocation. Scaling resources dynamically based on demand ensures optimal server performance and reliability.

b. Client-Side Performance:

- i. **Rendering Efficiency:** Utilizing React.js for fronted development allows for efficient rendering of UI components through the virtual DOM. Techniques such as code splitting, lazy loading, and memoization optimize rendering performance, ensuring smooth user interactions and responsiveness

2. Scaling Characteristics:

- a. **Horizontal Scaling:** Deploying multiple instances of the backend server behind a load balancer enables the distribution of incoming traffic across servers, improving reliability and scalability.
- b. **Database Scaling:** Implementing database sharding or replication techniques allows for efficient data distribution and storage across multiple database instances. This ensure scalability and high availability, enabling the applicaiton to handle gowing data volumes and concurrent request effectively

Conclusion

In the journey of conceptualizing, designing, and implementing our Chat Web Application using Client-Server Architecture, we have traversed through various dimensions of web development, performance optimization, and scalability. As we conclude this project, several key takeaways and reflections emerge:

1. **Achievements and Milestones:** We take pride in the successful deployment and operation of our Chat Web Application, which embodies the culmination of our efforts, creativity, and technical expertise. From crafting intuitive user interfaces to optimizing backend infrastructure, each milestone represents a testament to our dedication and perseverance.
2. **Empowering Communication:** Our Chat Web Application stands as a testament to the transformative power of digital communication, fostering collaboration, connectivity, and community engagement. By providing users with a versatile platform for real-time messaging, group discussions, and multimedia sharing, we aim to empower individuals and organizations to connect, communicate, and collaborate effectively.
3. **Continuous Improvement:** As with any software project, the journey does not end with deployment but marks the beginning of a new phase of continuous improvement and iteration. We remain committed to soliciting user feedback, addressing performance bottlenecks, and incorporating new features and enhancements to further enrich the user experience and expand the application's capabilities.
4. **Learning and Growth:** Throughout this project, we have embraced challenges as opportunities for learning and growth, expanding our technical skills, and honing our problem-solving abilities. The collaborative nature of the project has fostered teamwork, communication, and collaboration, laying the groundwork for future endeavors in software development and engineering.

In conclusion, our Chat Web Application represents more than just lines of code; it embodies our collective vision, passion, and commitment to harnessing technology for the betterment of society. As we bid farewell to this project, we look forward to the myriad possibilities and opportunities that lie ahead, knowing that the journey of innovation and discovery is never truly complete