## React UI

| import React, { useState, useEffect } from 'react';  const ChatApp = () => {  const [messages, setMessages] = useState([]);  const [inputValue, setInputValue] = useState('');   const handleInputChange = (event) => {  setInputValue(event.target.value);  };   const handleSubmit = (event) => {  event.preventDefault();  if (inputValue.trim() !== '') {  sendMessage(inputValue);  setInputValue('');  }  };   const sendMessage = (message) => {  *// Make API call to your backend server*  *// Include logic to interact with OpenAI API and receive the response*  *// Append the user message and OpenAI's response to the messages state*  setMessages((prevMessages) => [  ...prevMessages,  { text: message, sender: 'user' },  ]);  };   useEffect(() => {  *// Scroll to the bottom of the chat window when new messages are added*  const chatWindow = document.getElementById('chat-window');  chatWindow.scrollTop = chatWindow.scrollHeight;  }, [messages]);   return (  **<div className="chat-app">**  **<div id="chat-window" className="chat-window">**  {messages.map((message, index) => (  **<div  key={index}  className={`message ${message.sender === 'user' ? 'user' : 'bot'}`}  >**  **<span className="sender">**{message.sender}**</span>**  **<p className="text">**{message.text}**</p>**  **</div>**  ))}  **</div>**  **<form className="message-input" onSubmit={handleSubmit}>**  **<input  type="text"  placeholder="Type your message..."  value={inputValue}  onChange={handleInputChange}  />**  **<button type="submit">**Send**</button>**  **</form>**  **</div>**  ); };  export default ChatApp; |
| --- |

In this example, the ChatApp component maintains state for the messages and the input value. The handleInputChange function updates the input value as the user types. The handleSubmit function is triggered when the user submits a message, and it calls the sendMessage function to send the message to the backend and update the messages state.

The useEffect hook is used to scroll the chat window to the bottom whenever new messages are added.

Note that this is a simplified example to demonstrate the UI structure and flow. You would need to integrate it with your backend server and implement the API calls to interact with the OpenAI API as mentioned in the previous response.

You can style the components using CSS or a UI library of your choice to create a visually appealing chat interface.

## CSS

| .chat-app {  background-color: #1a1a2e;  color: #fff; }  .chat-window {  background-color: #1a1a2e;  color: #fff; }  .message.user {  background-color: #7f5af0;  color: #fff; }  .message.bot {  background-color: #4b0082;  color: #fff; }  .message .sender {  color: #fff; }  .message .text {  color: #fff; }  .message-input {  background-color: #1a1a2e; }  .message-input input {  background-color: #fff;  color: #1a1a2e; }  .message-input button {  background-color: #7f5af0;  color: #fff; } |
| --- |

## Backend

| const express = require('express'); const axios = require('axios'); const app = express(); app.use(express.json()); const OPENAI\_API\_KEY = 'YOUR\_OPENAI\_API\_KEY';  *// Handle POST request to /chat endpoint* app.post('/chat', async (req, res) => {  try {  const { message } = req.body;   *// Make API call to OpenAI*  const response = await axios.post(  'https://api.openai.com/v1/chat/completions',  {  messages: [  { role: 'system', content: 'You are a helpful assistant.' },  { role: 'user', content: message },  ],  },  {  headers: {  Authorization: `Bearer ${OPENAI\_API\_KEY}`,  'Content-Type': 'application/json',  },  }  );   *// Extract the generated message from the OpenAI response*  const generatedMessage =  response.data.choices[0].message.content.trim();   *// Send the generated message as the response*  res.json({ message: generatedMessage });  } catch (error) {  console.error('Error:', error.message);  res.status(500).json({ error: 'An error occurred' });  } }); *// Start the server* app.listen(3000, () => {  console.log('Server started on port 3000'); }); |
| --- |

In this example, we're using the Express framework to create a simple Node.js server. The /chat endpoint handles the POST requests and expects the user's message in the request body.

Inside the route handler, we make an asynchronous API call to the OpenAI API using Axios. We include the user's message along with a system message to instruct the AI assistant. The API response contains the generated message, which we extract and send back as the response to the client.

Make sure to replace 'YOUR\_OPENAI\_API\_KEY' with your actual OpenAI API key. You can obtain the API key from your OpenAI account.

To run this code, make sure you have Node.js and the required dependencies (express and axios) installed. Save the code in a file (e.g., server.js), and run node server.js in the terminal to start the server on port 3000.

Note: This is a basic example to illustrate the backend integration with the OpenAI API. You may need to add additional error handling, validation, and authentication mechanisms as per your specific requirements.