To make your contact form actually send an email to your address, you cannot do this directly from frontend JavaScript for security reasons.  
However, you can use a free service like [EmailJS](vscode-file://vscode-app/c:/Users/ammar/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") or [Formspree](vscode-file://vscode-app/c:/Users/ammar/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") to send emails from your form without needing a backend.

Below is an example using **EmailJS** (recommended for React/JS projects):

**1. Sign up for EmailJS**

* Go to [https://www.emailjs.com/](vscode-file://vscode-app/c:/Users/ammar/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) and create a free account.
* Add your email service (like Gmail) in the EmailJS dashboard.
* Create an email template.
* Note your **Service ID**, **Template ID**, and **Public Key**.

**2. Add EmailJS to your project**

Add this script in your <head> (before your <script type="text/babel">):

<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/emailjs-com@3/dist/email.min.js"></script>

**3. Update your handleSubmit function**

Replace your handleSubmit with the following (fill in your own IDs/keys):

// ...existing code...

const handleSubmit = (e) => {

  e.preventDefault();

  if (!formData.name || !formData.email || !formData.message) {

    setFormError('All fields are required!');

    return;

  }

  if (!/\S+@\S+\.\S+/.test(formData.email)) {

    setFormError('Please enter a valid email!');

    return;

  }

  setFormError('');

  // EmailJS integration

  emailjs.send(

    'YOUR\_SERVICE\_ID',      // Replace with your EmailJS service ID

    'YOUR\_TEMPLATE\_ID',     // Replace with your EmailJS template ID

    {

      from\_name: formData.name,

      from\_email: formData.email,

      message: formData.message,

    },

    'YOUR\_PUBLIC\_KEY'       // Replace with your EmailJS public key

  ).then(

    (response) => {

      alert('Message sent successfully!');

      setFormData({ name: '', email: '', message: '' });

    },

    (error) => {

      setFormError('Failed to send message. Please try again later.');

    }

  );

};

// ...existing code...

1 vulnerability

* **hardcoded-credentials** Embedding credentials in source code risks unauthorized access

**4. That’s it!**

* Now, when someone submits the form, you’ll get an email at your configured address.
* You can customize the template and sender in your EmailJS dashboard.

**Note:**  
Never expose sensitive credentials in your frontend code. EmailJS public keys are safe to use on the client.

Let me know if you want a Formspree example or help with another service!

handleSubmit (e)

The provided code defines a function named handleSubmit, which is typically used as an event handler for form submissions in a React application. The function takes an event object e as its parameter and begins by calling e.preventDefault(). This prevents the default browser behavior of reloading the page when a form is submitted, allowing the form submission to be handled entirely with JavaScript.

Next, the function performs validation on the form data. It checks if any of the required fields—name, email, or message—are empty. If any of these fields are missing, it sets an error message using setFormError and exits the function early to prevent further processing. This ensures that users cannot submit the form without filling out all necessary information.

The function then validates the email address using a regular expression. If the email does not match the expected pattern (a basic check for the presence of an "@" symbol and a domain), it sets a different error message and again exits early. This helps catch simple mistakes in email input before the form is processed.

If all validations pass, the function clears any existing error messages by calling setFormError with an empty string. It then displays a simulated alert to the user indicating that the message has been sent. Finally, it resets the form fields by calling setFormData with empty values for name, email, and message, preparing the form for a new submission. This approach provides a user-friendly way to handle form submissions with immediate feedback and basic validation.