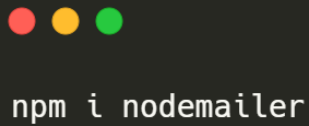


# **Material Complementario**

**Envio de formulario**

## Paso 1

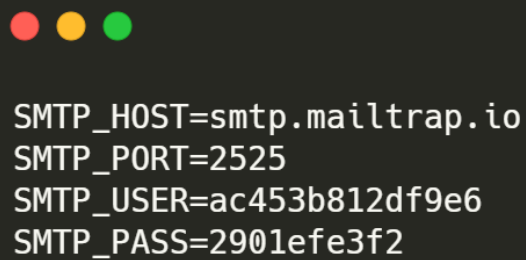
Instalamos la librería **nodemailer** para enviar correos con el siguiente comando.



```
npm i nodemailer
```

## Paso 2

Definimos las variables necesarias en nuestro archivo **.env**



```
SMTP_HOST=smtp.mailtrap.io  
SMTP_PORT=2525  
SMTP_USER=ac453b812df9e6  
SMTP_PASS=2901efe3f2
```

## Paso 3

Creamos un nuevo manejador de ruta dentro del archivo **api.js**. En este controlador capturamos los datos enviados y confeccionamos el cuerpo del mensaje y sus diferentes propiedades para enviarlo utilizando la librería instalada y las variables de entornos definidas.

```

router.post('/contacto', async (req, res) => {
  const mail = {
    to: 'flavia.ursino@gmail.com',
    subject: 'Contacto web',
    html: `${req.body.nombre} se contacto a traves de
la web y quiere más informacion a este correo:
${req.body.email} <br> Además, hizo el siguiente
comentario: ${req.body.mensaje} <br> Su tel es:
${req.body.telefono}`
  }

  const transport = nodemailer.createTransport({
    host: process.env.SMTP_HOST,
    port: process.env.SMTP_PORT,
    auth: {
      user: process.env.SMTP_USER,
      pass: process.env.SMTP_PASS
    }
  }); // cierra transp

  await transport.sendMail(mail)

  res.status(201).json({
    error: false,
    message: 'Mensaje enviado'
  });

});

```

## Paso 4

En el archivo **ContactoPage.js** vamos a importar useState de React y axios. Utilizaremos estas dependencias para controlar el estado de nuestro formulario, el feedback del mismo y realizar el envío de los datos cargados por el usuario hacia nuestra API.

En primera instancia vamos a definir el estado inicial del formulario con todos sus valores vacíos. Hacemos esto dentro de una variable para poder reutilizarla más tarde cuando queramos reiniciar el formulario.

A continuación definimos algunas variables de estado y sus correspondientes setters:

**sending / setSending:** es la encargada de mostrar o no al usuario el estado de "enviando..."

**msg / SetMsg:** almacena y muestra el mensaje recibido por la API para indicar al usuario que su información fue enviada.

**formData / setFormData:** contiene un objeto cuyas propiedades están enlazadas a los campos de nuestro formulario. Estas definen lo que se muestra en los campos mediante su propiedad value y son modificadas mediante la función **handleChange** que recibe un objeto Event del cual extraemos el nombre del campo que cambio y su valor para actualizar el estado.

Para finalizar con las definiciones tenemos la función **handleSubmit**, la cual controla el envío de los datos y la actualización de las variables de estado que controlan el feedback que el usuario recibe.



```
import React, { useState } from 'react';

import axios from 'axios';

const ContactoPage = (props) => {

  const initialForm = {
    nombre: '',
    email: '',
    telefono: '',
    mensaje: ''
  }

  const [sending, setSending] = useState(false);
  const [msg, setMsg] = useState('');
  const [formData, setFormData] = useState(initialForm);

  const handleChange = e => {
    const { name, value } = e.target;
    setFormData(oldData => ({
      ...oldData,
      [name]: value //forma dinamica
    }));
  }

  const handleSubmit = async e => {
    e.preventDefault();
    setMsg('');
    setSending(true)
    const response = await
    axios.post('http://localhost:3000/api/contacto', formData);
    setSending(false);
    setMsg(response.data.message);
    if (response.data.error === false) {
      setFormData(initialForm)
    }
  }
}
```

Solo resta modificar el formulario y sus elementos para que hagan uso de las funcionalidades que acabamos de definir.

Primero pasaremos la función **handleSubmit** a la propiedad onSubmit del elemento **<form>**. Esto hará que la función que definimos controle el proceso de envío en vez de realizarse de la forma convencional.

A cada input del formulario agregaremos una propiedad name. Es importante que coincidan los nombres que les damos a los inputs y los nombres que definimos dentro del objeto **initialForm**. También debemos enlazar la propiedad value al valor del estado correspondiente dentro de **formData**. Por último asignamos la propiedad onChange a nuestra función **handleChange** para que actualice el estado ante cada cambio de los campos.

Lo único que nos resta es hacer uso del operador ternario para hacer el renderizado condicional del estado de enviando y el mensaje de feedback.

```
<form className="formulario" onSubmit={handleSubmit}>
  <p>
    <label>Nombre</label>
    <input type="text" name="nombre" value={formData.nombre}
      onChange={handleChange} />
  </p>
  <p>
    <label>Email</label>
    <input type="text" name="email" value={formData.email}
      onChange={handleChange} />
  </p>
  <p>
    <label>Teléfono</label>
    <input type="text" name="telefono" value={formData.telefono}
      onChange={handleChange} />
  </p>
  <p>
    <label>Comentario</label>
    <textarea name="mensaje" value={formData.mensaje}
      onChange={handleChange} ></textarea>
  </p>
  {sending ? <p>Enviando...</p> : null}
  {msg ? <p>{msg}</p> : null}
  <p className="centrar"><input type="submit" value="Enviar" /></p>
</form>
```

# Bibliografía utilizada y sugerida

## Artículos de revista en formato electrónico:

**MDN Web Docs.** Disponible desde la URL: <https://developer.mozilla.org/>

**React.**Disponible desde la URL: <https://es.reactjs.org/>

## Libros y otros manuscritos:

**Alex Banks & Eve Porcello.** Learning React: desarrollo web funcional con React y Redux. 2017.