Para probar el envío de mails vamos a utilizar una herramienta que se utiliza muchísimo en el desarrollo web, y tiene que ver con estos servidores falsos o servicios mooc, como se les llama también, que simulan un servidor que presta algún servicio. En este caso, será un servidor de correo. Si lo que nosotros desarrollamos corre bien en el servidor falso, va a correr perfectamente bien también en el servidor de correo para la empresa para la cual estemos desarrollando un sistema, por ejemplo, ¿verdad? Para eso se utilizan estos servicios. Vamos a crearnos una cuenta. Utilizaremos Mailtrap. Hay muchos de estos. Hay muchos servicios. Todos funcionan muy parecidos. Y echaremos mano a una herramienta de Node.js, a una dependencia o paquete que es muy útil, para conectar nuestra aplicación web, nuestro servidor, con los servicios de envío de correos. Se llama Nodemailer. Vamos a instalarlo, vamos a aceptarlo y vamos a trabajar el envío de mails. Ahora, vamos a crearnos una cuenta en un servidor falso de correo que sirve para cuando estamos desarrollando un sistema. Son muy usados en este tipo de servidores. Hay unos cuantos. Nosotros vamos a usar mail trap en esta ocasión.. Les sugiero que vayan a mailtrap.io, aquí, y se creen una cuenta. Es totalmente gratuito. Yo tengo una cuenta ya, así que directamente voy a ingresar. Aquí 'Log in', voy a utilizar mi cuenta de google. Así es como estoy registrado.

Y bien, ya estamos dentro. Ok, cuando ustedes ingresen aquí, tendrán que crear su inbox. Una vez que estén registrados obviamente. Vamos a agregar una casilla de correo y vamos a ponerle un nombre. Pongámosle igual que nuestra página web: Kittens. Perfecto, ahora que está creada vamos a entrar aquí, en Kittens.

Y lo próximo será buscar aquí, en la sección de integraciones, integration, con qué sistema queremos utilizar nuestro servidor de correo. Vamos a buscar para Node.js y aquí nos dice que debemos usar el paquete llamado Nodemailer, y nos entrega esta cadena de conexión. Muy bien, vamos a copiarla. Y ahora, volveremos a nuestro código, cerraremos el proceso del servidor para instalar el paquete que nos pide utilizar; 'npm i Nodemailer'. Este es el paquete, la dependencia que va a encargarse de la conexión entre nuestro proyecto y el servidor de correo. Muy bien, vamos a correr nuevamente en nuestro proyecto, 'npm run dev', para tenerlo funcionando. El paquete Nodemailer ya está instalado, podemos revisar el package.json y, pues, aquí lo vemos. Muy bien, ahora lo que tenemos que hacer es configurarlo para poder trabajar para poder hacer la conexión entre el proyecto, el formulario con los datos y de ahí enviarlos a nuestro servidor de correo en Mailtrap. ¿Recuerdan que copiamos aquí los datos de conexión?, ¿bien? Ustedes pueden seleccionar, darle click y copiar o simplemente darle copiar aquí. Vamos a hacerlo nuevamente para que lo recuerden bien. Bueno, y ahora volveremos a nuestro proyecto, y en el controlador que envía nuestro correo que es aquí, en contacto.js, la ruta que viene con post; recuerden que hasta ahora lo que hicimos fue comprobar que los datos que escribimos aquí, entran a este controlador y los podemos capturar en el cuerpo de la request. Es decir, en req.body. Recuerden en la clase pasada hacíamos un console.log y los datos aparecían aquí, es decir, los tenemos disponibles. Vamos a borrar esta línea, ya no nos sirve, sólo queríamos para testear. Y vamos a copiar lo que nos trajimos. La hemos perdido, perdón. Vamos otra vez aquí, lo que nos trajimos de Mailtrap. Aquí está nuestro usuario, los datos de conexión, nuestra contraseña. Estos datos son importantes. Ustedes no tienen que poner estos, sino los que se les generan a ustedes. Estos datos digo son

importantes para establecer la conexión y el envío del correo. Ok, lo copiamos aquí. Podríamos reemplazar: var por const. Esto simplemente es porque, bueno, nos gusta que quede un poco más moderno y prolijo. Pero no hay ningún problema, como está de fábrica, es decir, como nos sugieren el uso, va a estar bien, va a funcionar bien, ¿ok?.

Y vamos a continuar con el procedimiento. A continuación aquí vamos a hacer lo siguiente, vamos a capturar

todos los elementos que vienen desde el formulario. Recuerden que si vemos nuestro formulario de contacto, tenemos en cada input y en el 'textarea', un atributo llamado 'name' con un nombre ¿ok? Esta es la variable que captura los datos que los usuarios ingresan en los campos, y los transporta cuando damos 'submit' en el formulario, hacia este controlador con este método. Por lo tanto, nosotros tenemos que preguntar por 'req.body.nombre', 'req.body.apellido', 'req.body.email'. Bueno, y así sucesivamente, ¿se entiende, verdad? Y aquí he escrito algo sin querer, perdón. Ok, volvemos entonces explicado esto a nuestro controlador. Entonces, aquí podríamos hacer lo siguiente.

Vamos a desestructurar esa entrada, para que sea más práctico para nosotros el trabajo. Vamos a capturar todos los campos que vienen del formulario de la siguiente manera. Sólo para escribir menos. Recordemos que era: nombre, apellido, email, mensaje. Tengan presente que lo que estamos capturando aquí, que ingresa en la request, no es otra cosa que… y vuelvo a mostrárselos para que les quede claro, no el nombre del campo, el nombre que puede figurar aquí. Esto es simplemente una etiqueta. Sino el atributo name, el atributo name de cada input, y en este caso, del textarea que es esté cuadrado de aquí. Son sumamente importantes porque allí es donde viajan los datos hacia nuestro back-end. Bueno, aquí otra vez parece que sin querer he roto algo. Está perfecto, muy bien, voy a cerrar esta esta vista, por las dudas, para no estropear nada. Aquí tenía la indicación de que todavía tengo trabajo por terminar. Bien, entonces aquí están los campos que vienen del formulario y todo esto viene en el objeto request. Particularmente en el cuerpo. Muy bien, he desestructurado, he metido todos estos datos en estas variables, con este nombre, que es el mismo nombre. Lo hacemos para escribir menos a continuación. Vamos a crear email saliente, por ejemplo. Esto será un objeto literal que contendrá ni más ni menos que nombre, apellido,

pero aguarden un minuto. Una regla de JavaScript moderno nos dice que cuando el nombre de la propiedad y el nombre de la variable que contiene el dato se llaman igual, solamente con poner uno está bien. Por eso escribimos menos todavía, ¿ok? Muy bien, apellido, email y mensaje. Tengan en cuenta lo siguiente, si no hubiésemos escrito esta línea, aquí tendríamos que escribir en cada caso: nombre es igual a 'req.body.nombre', y así sucesivamente en cada uno de los campos. Por eso aplicamos esta desestructuración. Espero que les quede claro, es una técnica muy importante que nos ayuda bastante a reducir los tiempos de codificación de nuestro, de nuestro programa, de nuestro código. Bien, y entonces ahora, lo que vamos a hacer aquí en el email saliente es…

Con estos campos haremos lo siguiente, miren, vamos a poner un campo ‘to’ aquí vamos a configurar un mail, ¿ok? Es como si esto, es como si esto fuera el cuerpo de un mail. Ni más ni menos. Cuando aquí nos vamos a inventar una casilla ¿ok? atención clientes

arroba kittles punto ORG. Suponemos que esto sería la dirección real, ¿verdad? pero por ahora es simplemente inventada. Y aquí en el campo ‘from’, es decir, de quién viene, bueno, vamos a utilizar precisamente esta propiedad: nombre. Muy bien, bueno, en realidad no. Vamos a usar más que nombre, email. Así sería más correcto, ¿verdad? Muy bien, ahora vamos a indicar ‘subject’, el subject de nuestro email, el tema de nuestro email. Bueno, indicamos que se trata de un mensaje desde el formulario de contacto, ¿ok? Y ahora aquí pondremos en html, es decir, en el cuerpo literalmente del email, vamos a hacer string interpolation con las vac tics, para poder escribir las propiedades, aquí las variables. Email ya lo hemos usado así que lo voy a borrar junto con algo de texto. Podríamos indicar…

aquí podríamos indicar: nombre,

nombre y dejar un espacio, y apellido;

y aquí podríamos poner “Ha enviado el siguiente mensaje” y aquí concatenamos el mensaje.

No lo usamos y aquí tenemos el mensaje. Y aquí lo que hacemos es con los datos que capturamos del formulario, conformamos el cuerpo de un email, que será enviado a través de nuestro Nodemailer al servidor de mailtrap. Recordamos mailtrap como muchos otros es un servidor de pruebas, pero si esto funciona en la etapa de pruebas, luego simplemente reemplazaremos los datos de conexión por nuestro servidor de correo verdadero y todo estará funcionando. De eso se trata. Ok, tenemos construido el email saliente. Hemos hecho la conexión aquí arriba con mailtrap a través de Nodemailer, ¿verdad? Bien, solamente nos faltaría enviar este mensaje. Vamos a ver, bueno, aquí tengo código todavía para depurar, ¿verdad? Esto debería cerrar aquí. Ahí está mejor. Muy bien, entonces ahora haremos lo siguiente, vamos a poner dentro de un bloque try catch. Un bloque try catch nos permite probar una secuencia de código y atrapar los posibles errores. Esto lo haremos en el marco de un procedimiento asincrónico porque el envío de un email no es inmediato. Toma un tiempo la conexión con el… el servidor. Por lo tanto, vamos a indicarle a JavaScript, a Node a través de JavaScript en este caso, que ésta es una función asincrónica. Vamos a indicar delante de la palabra function, la palabra clave ‘async’. Esto nos va a permitir trabajar con código asincrónico sin que nuestro programa tenga fallos o se rompa directamente. Aquí vamos a declarar una variable que se llame ‘sendMailStatus’ porque aquí vamos a capturar el estado en el que se encuentra el proceso que hemos iniciado, para ver si anduvo bien o si falló. Y como esto es un procedimiento asincrónico, vamos a decir que espere: await. ‘async’ y ‘await’ trabajan en conjunto. No puede haber una palabra ‘await’ si la función que ejecuta el bloque no es una función

de tipo asincrónico ¿ok? Entonces, aquí vamos a indicar que espere la función del envío de mails. Vamos a capturar 'transport' que ya lo hemos configurado más arriba, y vamos a utilizar su método 'sendMail'. ¿Le pusimos 'transport', verdad? Sí, aquí está bien. 'transport.sendMail' tiene un método que se llama así. Este método lo que hace es enviar un mail, ¿y dónde está nuestro mail? En este objeto que construimos acá, en 'emailSaliente'. Así que aquí pondremos simplemente 'emailSaliente'. Muy bien, una vez hecho esto podríamos hacer lo siguiente: declarar aquí una variable que se llame 'statusMessage', dónde vamos a vamos a inicializar la vacía, para luego ponerle el mensaje que corresponda de acuerdo a cómo salió nuestro procedimiento. Entonces, aquí vamos a preguntar si, por ejemplo, 'sendMailStatus' que recibió la información de este procedimiento, ¿ok? Vamos a preguntar si su objeto rejected

tiene algún tiene longitud. Si tiene una longitud quiere decir que ese objeto existe, que tiene un mensaje de error. Por lo tanto, vamos a indicar que hubo algún error. Vamos a pasarle a la variable que creamos aquí arriba para recoger los mensajes de estado 'statusMessage'. Vamos a decirle, por ejemplo, algo como "No pudimos enviar el mail. Intenté nuevamente", por ejemplo, ¿verdad? Algo así podría ser, o más corto.

Bueno, por ahora con esto está bien. Simplemente para que ustedes vean cómo funciona. Un mensaje corto. Ahora, si no hubo un error, es decir, que el mensaje fue enviado, entonces podríamos indicar a 'statusMessage'

que… bueno, el mensaje se envió correctamente. "Mensaje enviado". Bien, ahora, sin ninguna de estas cosas funcionó, es que hubo un error. No en el proceso de envío sino que directamente no se pudo establecer la conexión con el servidor. Por eso tenemos el bloque catch, y si el bloque catch efectivamente se está ejecutando si alcanzamos esta sección de código bueno es que aquí hubo un problema. Podríamos indicar… bien, primero terminemos con esta parte porque si todo anduvo bien, es decir, si funcionó la conexión con el servidor de correo ya sea que se pudo enviar o no se pudo enviar vamos a volver a mostrar el formulario, es decir, haremos un render de nuestro formulario. Recuerden que se llama contacto. Y además, después de una coma, le vamos a pasar un objeto. En este caso, le vamos a pasar el mensaje, para decirle al usuario qué es lo que ha pasado. Si estuvo bien, si estuvo mal, si se envió o no se envió. 'statusMessage' Por el contrario, si no se pudo establecer conexión con el servidor y arribamos a este bloque catch, haremos lo siguiente: vamos a renderizar también la vista, contacto, pero con otro mensaje. Con un mensaje diferente. Podríamos decir 'statusMessage' y podemos asignar el valor directamente aquí adentro. Dentro del objeto con dos puntos.

"Servidor fuera de servicio", por ejemplo.

Ok, son mensajes, no son muy buenos. Tendríamos que darle más información, decirle que intente más tarde. Esto es, reitero, simplemente para que ustedes vean cuál es la lógica, cómo funciona. Luego podemos darle muchas mejoras a esto pero esta es la lógica de funcionamiento. Bueno, muy bien, ahora podríamos ver si esto efectivamente está funcionando. Revisamos nuestro mailtrap, vemos que aquí en nuestro inbox no hay ningún mensaje, ¿ok? Muy bien, vamos al mundo de los gatitos, ponemos aquí algún nombre, vamos a actualizar por las dudas. Bien, bien estoy escribiendo no muy bien, pero no importa. Esto es simplemente para poner cualquier cosa. "Mensaje de prueba… ¿Están ahí?"

Y vamos a enviar. Enviamos. Esto toma un momento porque es asincrónico, recuerden. Bueno y aquí hemos tenido un problema que pasaremos a resolver de inmediato. Vamos a ver qué ocurre aquí, dónde está el problema. Aparentemente hemos olvidado algo muy importante. Hemos configurado y hemos importado el paquete Nodemailer, lo hemos configurado para utilizarlo pero ¿lo ven aquí? yo tampoco lo veo. Nunca lo importamos, caramba. Vamos a resolverlo: const nodemailer es igual a require. Esto requiere el paquete Nodemailer.

Nodemailer, ok, ahí estamos corriendo nuevamente. Vamos a ver qué ocurre. Pedimos nuevamente nuestro formulario. Enviamos, Eduardo, cualquier cosa. Vamos a hacerlo rápido para que la clase no se haga muy larga, y aquí, este mensaje. Veamos nuevamente, está intentando. Vamos a ver qué ocurre, y bueno, aparentemente el mensaje habría… habrá sido enviado. Vamos a ver si esto es cierto. Actualizamos aquí

y tenemos un mensaje enviado, caramba, parece que esto está funcionando. Y aquí está nuestro mensaje. Mensaje de Eduardo. Bueno, fíjense que acá se construye con el email de dónde vino. Estos son los datos que escribimos nosotros. Aquí dice "Eduardo ha enviado el siguiente mensaje" y llega el mensaje. Así recibiríamos, entonces, nuestro correo. Solamente nos falta ver aquí cómo podemos hacer para recibir en el formulario los mensajes que le estamos pasando a la vista, pero no estamos mostrando aquí. Lo haremos muy rápidamente para terminar esta clase. Venimos a contacto, por ejemplo, y recuerden que cuando renderizamos contacto le estamos enviando el objeto 'statusMessage'. Recuerden, aquí, tanto si sale bien o no sale bien siempre enviamos a la vista le enviamos, a la vista, el estado del mensaje, es decir, el estado de la operación en un mensaje. Por lo tanto, aquí podemos recibirla y podríamos aquí mismo, dentro, preguntar con un helper de handlebars, un ayudante que nos permite utilizar un if. Un if de JavaScript exactamente. Vamos a preguntar si existe 'statusMessage'

Y en el caso de que exista, ¿por qué preguntamos si existe? porque la primera vez que se renderiza este formulario, antes de enviar ningún mensaje, no tenemos esa variable. Por lo tanto, no queremos mostrar un mensaje erróneo, pero si existe efectivamente… aquí abrimos el bloque, de aquí lo cerramos… bueno, vamos a mostrar que… vamos a hacerlo simplemente en un span,

la variable. Es decir, que ese objeto con una clave y un valor que nos ha llegado desde el formulario. Simplemente para que esto quede un poco mejor vamos a agregar algo de estilos para que el mensaje se muestre bien. Entonces venimos a public css, y aquí al final, simplemente porque, bueno, está bien, en este caso que esté aquí en el final, agregamos un poco de estilos para ese mensaje. Y vamos allá, otra vez, enviamos un nuevo mensaje: Pedro…

Vamos a hacerlo bien esta vez.

Perfecto, aquí tenemos un hermoso mensaje que va a enviar Pedro. Y vamos a ver qué ocurre. Lo enviamos, aparentemente sale. ¡uy! buenísimo, tenemos un mensaje. Un feedback: “Mensaje enviado”. Solamente faltaría chequear que el mensaje de Pedro Fuentes haya entrado. Miren, justo llegó. Ok, y aquí tenemos el mail y el mensaje. Prueba completada.