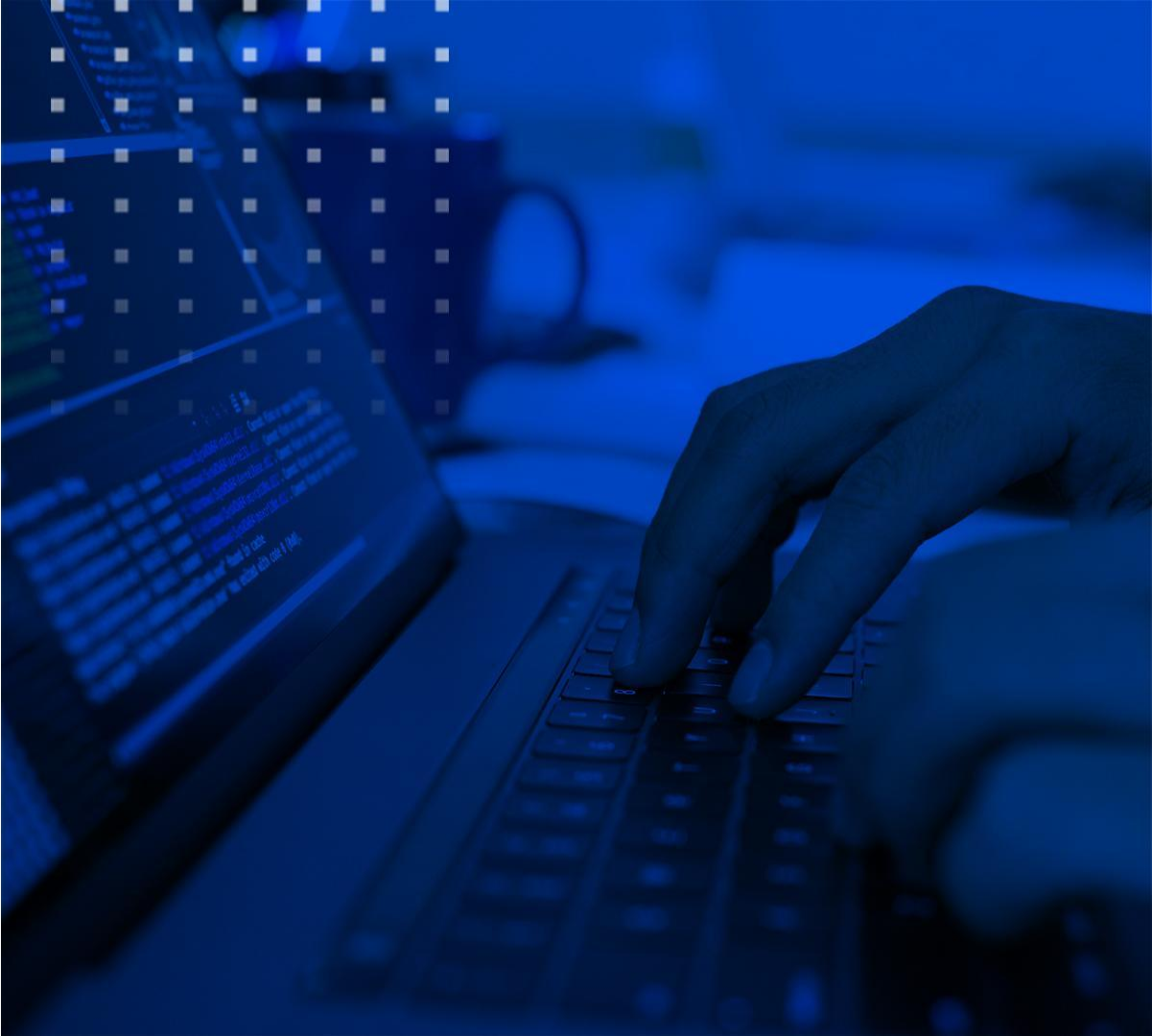


CURSO

FULL STACK DEVELOPER NIVEL INICIAL

UNIDAD 16
Envío de mails con Node JS



Este instructivo detalla los pasos a seguir para probar la funcionalidad de envío de correo desde un formulario. Utilizaremos un servicio gratuito que brinda la posibilidad de testeado, recibiendo el tráfico de correo saliente de nuestra aplicación.

Para recoger la información utilizaremos un formulario de contacto. Aquí mostramos como ejemplo el formulario contenido en nuestro archivo creado con Handlebars llamado `contact.hbs`, ubicado en la carpeta `views` (vistas).

```
<form class="form" action
```

```
<form class="form" action="/contact" method="post">
  <label for="nombre"></label>
  <input type="text" name="nombre" autofocus placeholder="Nombre" />
  <label for="apellido"></label>
  <input type="text" name="apellido" placeholder="Apellido" />
  <label for="email"></label>
  <input type="email" name="email" placeholder="E-Mail" />
  <label for="mensaje"></label>
  <textarea
    name="mensaje"
    cols="30"
    rows="6"
    placeholder="Su mensaje aquí..."
  ></textarea>
  <div class="btn">
    <input type="reset" value="Limpiar" />
    <input type="submit" value="Enviar" />
    <a href="/"><input type="button" value="Home" /></a>
  </div>
</form>
```

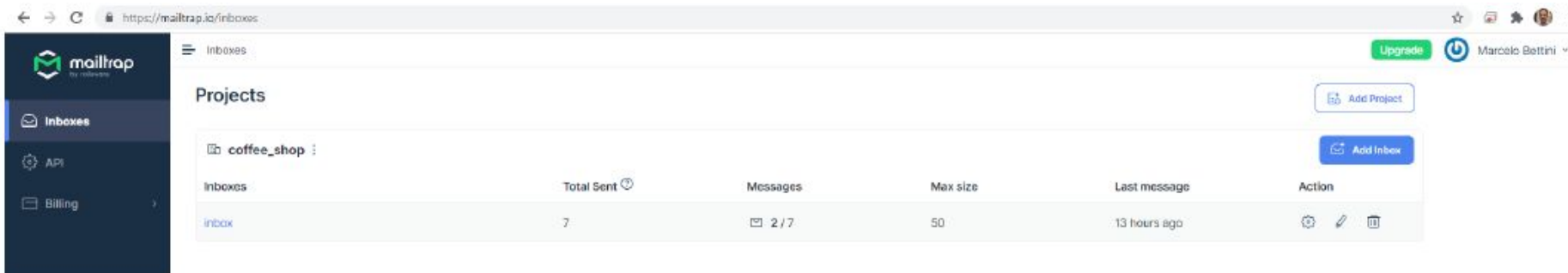
El atributo `action` indica la ruta que cargará el controlador cuando se envíe el formulario. En nuestro ejemplo hace referencia al controlador `contact.js`, ubicado en la carpeta `routes`. El atributo `method` indica que se trata de una petición de tipo `post`. Como recordará, el verbo `get` trae información del backend mientras que el verbo `post`, la envía.

Notará que hemos incluido una etiqueta `<label>` por cada `<input>`. Aunque mostramos el texto indicativo de cada `<input>` directamente en el campo de entrada con el atributo `placeholder`, las etiquetas `<label>` se incluyen por cuestiones de accesibilidad, note que están vacías de texto pero mantienen la referencia que las vincula con cada `<input>`.

El atributo `name` de cada `input` es el que permite identificar desde JavaScript (NodeJS, en este caso) el valor introducido por el usuario en cada campo luego de que se ejecuta el `submit` del formulario.

Ahora, crearemos una cuenta en el servicio Mailtrap:
<https://mailtrap.io/>

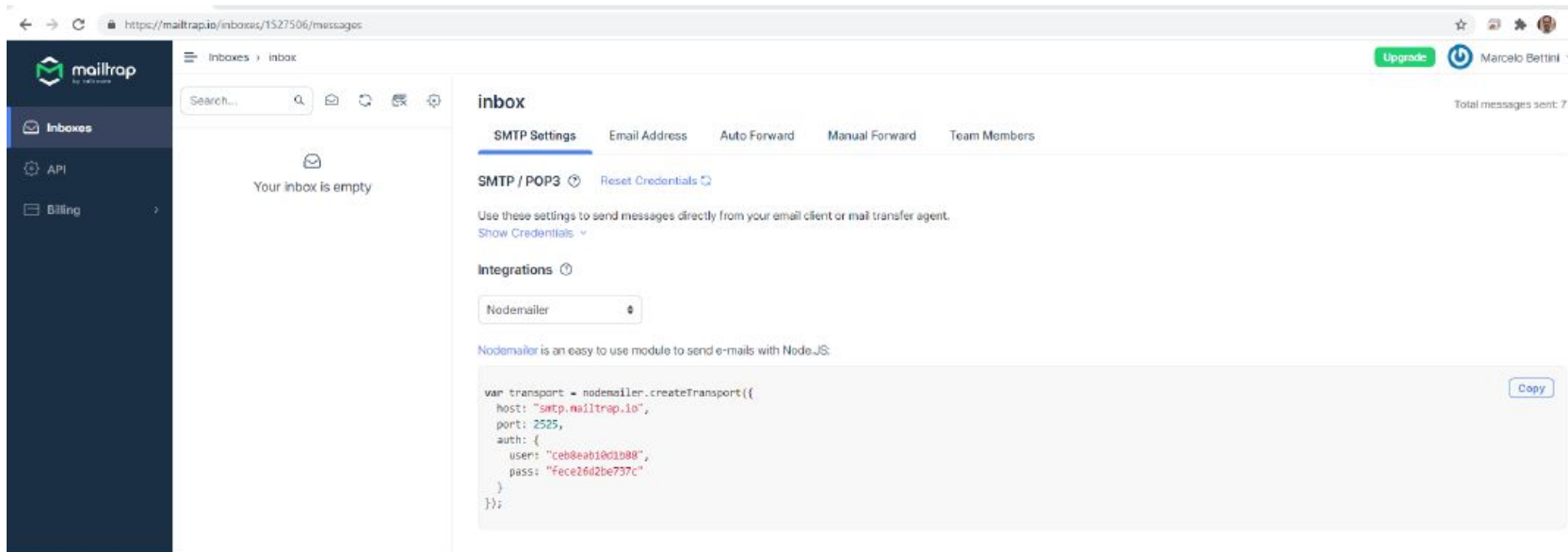
Luego agregaremos nuestro proyecto y nuestra bandeja de entrada (en la versión gratuita no podemos tener más que un proyecto y una bandeja; es suficiente para probar nuestro código).



The screenshot shows the Mailtrap web interface. The browser address bar displays <https://mailtrap.io/inboxes>. The user is logged in as 'Marcelo Bettini'. The interface includes a sidebar with navigation options: 'Inboxes', 'API', and 'Billing'. The main content area is titled 'Projects' and shows a table of inboxes for the project 'coffee_shop'.

Inboxes	Total Sent	Messages	Max size	Last message	Action
inbox	7	2 / 7	50	13 hours ago	Settings Edit Delete

Al dar clic en nuestro **inbox** veremos las credenciales que nos servirán para conectarnos y la forma de integrar el servicio con diferentes lenguajes de programación. Para **Node.js** verá listado el paquete **Nodemailer**. Cuando demos clic veremos las opciones de configuración que usaremos para integrar nuestro proyecto con **mailtrap**.



The screenshot shows the Mailtrap web interface. The left sidebar contains navigation links for 'Inboxes', 'API', and 'Billing'. The main content area is titled 'inbox' and shows 'SMTP Settings' as the active tab. Below this, there are links for 'Email Address', 'Auto Forward', 'Manual Forward', and 'Team Members'. The 'SMTP / POP3' section includes a 'Reset Credentials' link and a note: 'Use these settings to send messages directly from your email client or mail transfer agent. Show Credentials'. The 'Integrations' section has a dropdown menu set to 'Nodemailer'. Below this, a code block provides the Nodemailer configuration for Node.js:

```
var transport = nodemailer.createTransport({
  host: "smtp.mailtrap.io",
  port: 2525,
  auth: {
    user: "ceb8eab10d1088",
    pass: "feca26d2be737c"
  }
});
```

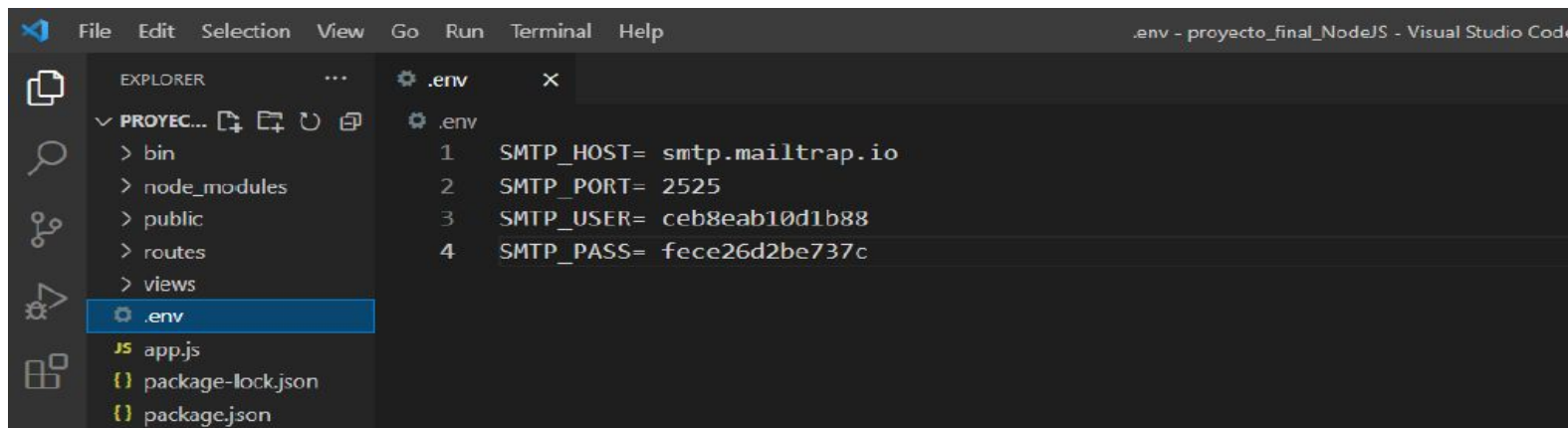
A 'Copy' button is located to the right of the code block. The interface also shows 'Your inbox is empty' and 'Total messages sent: 7'.

Vayamos a Node e instalemos en la carpeta de nuestro proyecto los módulos necesarios para completar la tarea:

- **Nodemailer** <https://www.npmjs.com/package/nodemailer>
npm i nodemailer
- **dotenv** <https://www.npmjs.com/package/dotenv>
npm i dotenv

Crearemos en el nivel superior de nuestra carpeta de proyecto el archivo `.env`. Aquí trabajaremos con la dependencia **dotenv**, creando las variables de entorno con datos de configuración que estarán disponibles en toda nuestra aplicación.

Es el archivo en el que copiaremos los datos de configuración provistos por **mailtrap** que vimos en la captura de pantalla anterior. Note que eliminamos las comillas.



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left and the Editor window on the right. The Explorer sidebar shows a project structure with folders: bin, node_modules, public, routes, and views. Below these are files: app.js, package-lock.json, and package.json. The .env file is selected in the Explorer. The Editor window shows the content of the .env file:

```
.env
1 SMTP_HOST= smtp.mailtrap.io
2 SMTP_PORT= 2525
3 SMTP_USER= ceb8eab10d1b88
4 SMTP_PASS= fece26d2be737c
```

ATENCIÓN

NO debe copiar los datos que ve en estas capturas de pantalla. Usted tendrá datos diferentes para user y pass, provistos por mailtrap. Si por cuestiones de seguridad reinicia sus credenciales en mailtrap (*Reset Credentials, en la pestaña SMTP Settings*), luego deberá actualizar esos datos de user y pass en su archivo .env

Finalmente, vamos a importar (*require*) **dotenv** en nuestro archivo principal **app.js**

Podemos hacerlo bien arriba, entre las primeras líneas de nuestro código:

```
//REQUERIMOS LAS VARIABLES DE ENTORNO, DONDE HEMOS DEFINIDO  
//LOS DATOS PARA ACCEDER A MAILTRAP CON NODEMAILER  
require("dotenv").config();
```

Creando el método post para enviar el mensaje al servidor de correo

Hasta aquí, nuestro archivo `contact.js` contiene solamente el controlador para mostrar el formulario (método `get`). Crearemos el controlador para el método `post`. Nuestro archivo quedará como se muestra en las siguientes capturas de código:

```
const express = require("express");
const router = express.Router();
const nodemailer = require("nodemailer");
//get muestra la ruta
router.get("/", (req, res, next) => {
  res.render("contact");
});
//post vincula con el submit del formulario y envía los datos
//al servidor de correo
router.post("/", async (req, res, next) => {
  /* asigno a estas constantes los datos que llegan desde cada
  campo de mi formulario */
  const nombre = req.body.nombre;
  const apellido = req.body.apellido;
  const email = req.body.email;
  const mensaje = req.body.mensaje;
```

```
/* y utilizo esa información para construir un objeto que
   contendrá el mensaje saliente*/
let emailMsg = {
  to: "coffeehouse@info.com",
  from: email,
  subject: "Mensaje desde formulario de contacto",
  html: `Mensaje de ${nombre} ${apellido} : ${mensaje} `
};

/*Asignamos a transport los datos de configuración de mailtrap,
   contenidos en el archivo .env*/
```

```
let transport = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: process.env.SMTP_PORT,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS,
  },
});
/*Finalmente, enviamos el objeto con el mensaje usando el método
transport.sendMail() Para hacerlo, seguimos la sintaxis propuesta
en la documentación de nodemailer: https://nodemailer.com/about/ */
let info = await transport.sendMail(emailMsg);
```

```
/*una vez que el mensaje se envía, volvemos a renderizar nuestra
vista del formulario pasándole una propiedad con un mensaje*/
  await res.render("contact", {
    message: "Mensaje Enviado",
  });
});

module.exports = router;
```


Para que ese “Mensaje Enviado” se muestre en nuestro formulario, tendremos que agregar la referencia a la propiedad **message** en nuestra vista **contact.hbs**.

Antes de la etiqueta que cierra el formulario, la incluiremos entre las dobles llaves que le indican a Handlebars que lo que se encuentra allí no es texto sino código JavaScript (*en este caso, la variable o propiedad que pasamos al llamar al nuevo render de la vista luego de enviar el mensaje*).

```
<p>{{message}}</p>  
</form>
```

Nuestra integración está terminada. Si corremos el proyecto con `npm start` y enviamos un mensaje desde el formulario, lo veremos aparecer en nuestro inbox del servicio mailtrap.



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES