

Llamadas asíncronas (Parte II)

El protocolo SMTP

Competencia

- Reconocer el uso del protocolo SMTP para el envío de correos electrónicos

Introducción

Hoy en día no podemos concebir un mundo sin correos electrónicos, es el medio de comunicación más formal en la industria y con el cual respaldamos cualquier información importante. El protocolo SMTP es usado para el envío simple de correos electrónicos y utilizado por diferentes tecnologías para enviar correos de forma programática.

En este capítulo, aprenderás de qué se trata este protocolo para prepararte y desarrollar próximamente una aplicación con Node que lo ocupe para enviar un correo electrónico.

El protocolo SMTP

Por sus siglas, Simple Mail Transfer Protocol (Protocolo simple de transferencia de correo) es un protocolo usado para el envío de correos electrónicos, bajo simples parámetros o propiedades que indican declarativamente la información expuesta en el correo, la información correspondiente al origen y destino del correo. Este protocolo por supuesto utiliza el internet como medio de transmisión, por lo que no podría ejecutarse si no se está conectado de forma online.

Los correos enviados bajo este protocolo tienen la particularidad de ser reconocidos como correos seguros, es decir, no deberán ser considerados SPAM. ¿Por qué un correo puede ser considerado SPAM? Un correo es considerado SPAM por varios motivos, pero el más frecuente es la dudosa procedencia, es decir, la IP o el DNS que fue utilizado para el envío es extraño o incluso su envío no fue solicitado, por lo que se considera de primera instancia como posible correo basura.

Ahora hablemos de Node, como bien sabes es nuestro entorno de desarrollo con JavaScript, el cual puede acceder a funcionalidades a nivel de sistema operativo, más allá de la programación de eventos que usamos en el desarrollo frontend.

Entonces, ¿Cómo podemos programar una función que utilice el protocolo SMTP, para enviar correos electrónicos desde nuestras aplicaciones? No es tan complejo como parece, lo único que necesitaremos de forma indispensable será un proveedor de correos electrónicos.

Hoy en día contamos con una importante herramienta ofrecida por Gmail, para representarnos como nuestro proveedor de correos, gracias a un servicio que ofrecen que posibilita la opción de usar nuestro correo como emisor, simulando de esta manera un proceso común de envío de mail desde el propio sitio de Gmail ¿Genial cierto? Esta programación la veremos un poco más adelante.

Es importante entender cómo funciona el protocolo SMTP y para esto te muestro el proceso diagramado en la siguiente imagen:

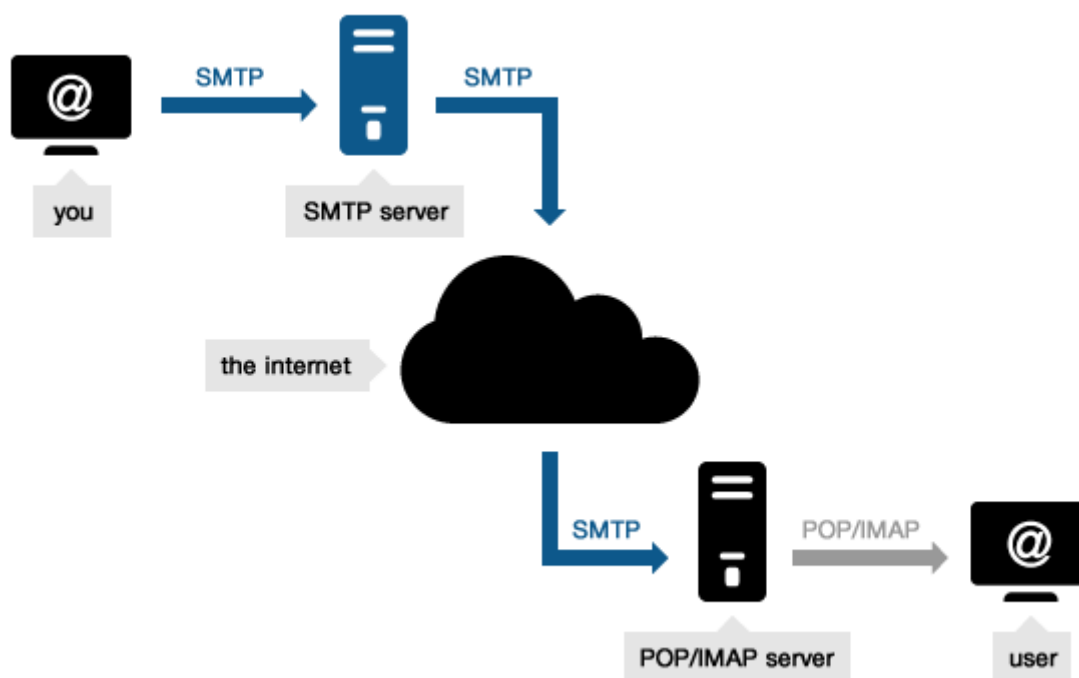


Imagen 1. Diagrama de protocolos para el envío de correos electrónicos.

Fuente: www.serversmtp.com

Observando el diagrama nos podemos dar cuenta que el proceso es sencillo, no obstante al final del flujo encontramos unas siglas nuevas POP/IMAP. ¿Qué son estas siglas? Para entenderlo en pocas palabras el protocolo SMTP nos sirve para el envío de correos, no obstante para la recepción, lectura, compatibilidad y descarga de éstos se realizan bajo estos protocolos. A continuación, te dejo la definición según Gmail expuesta en un artículo dentro de su sitio oficial [G Suite](#):

- **IMAP:** Almacena el correo en el servidor del cliente, por lo que se puede acceder a él desde cualquier dispositivo, ya sean teléfonos, ordenadores o tablets. Los correos electrónicos no se descargan si no haces clic en ellos y los archivos adjuntos no se descargan automáticamente, lo que te permite consultar tus mensajes rápidamente desde cualquier dispositivo.
Si accedes desde varios dispositivos y quieres que el correo electrónico y los archivos adjuntos se almacenen en el servidor del cliente, usa IMAP. (Fuente: [G Suite](#))
- **POP:** Se conecta al servidor del cliente, descarga todos los mensajes nuevos en un solo dispositivo y luego elimina el correo del servidor. Si lees los correos electrónicos en un único dispositivo y quieres que se eliminen del servidor después de descargarlos en tu dispositivo, usa POP. (Fuente: [G Suite](#))

El paquete Nodemailer

Competencias

- Identificar las propiedades básicas del paquete nodemailer para el envío de correos electrónicos
- Construir una aplicación Node que envíe correos electrónicos con el paquete Nodemailer

Introducción

El paquete Nodemailer es el paquete de NPM más popular para enviar correos electrónicos en una aplicación Node, gracias a su simple e intuitiva estructura, podrás en cuestión de minutos programar un servidor que realice el envío de uno o muchos correos electrónicos especificando el host como proveedor de correos.

En este capítulo, aprenderás las propiedades y métodos de este paquete, generarás el envío de un correo utilizando una cuenta de gmail como proveedor de correos, creada para motivos académicos de esta lectura, no obstante podrás ocupar también tu correo personal para hacer tus prácticas más personalizadas.

Nodemailer

Nodemailer es otro paquete muy conocido de NPM, nos sirve como herramienta para hacer envíos de correos electrónicos con Node y cuenta con un [sitio oficial](#), en el que encontrarás toda la información que necesites en caso de que quieras profundizar en su API y sus herramientas. El proyecto comenzó el 2010, cuando no había una opción sensata para enviar mensajes de correo electrónico, hoy es la solución a la que recurren la mayoría de los usuarios de Node de forma predeterminada.

Su uso es bastante intuitivo, por lo que con pocas líneas de código podremos hacer envío de correos electrónicos, no obstante, no tiene mucho sentido redactar los correos desde el servidor, por lo que ocuparemos una aplicación en el lado del cliente, que por medio de un formulario le ofrezca al usuario final poder especificar los datos básicos de un correo, como el asunto, el correo de él o de los destinatarios y el mensaje a enviar.

Propiedades de nodemailer

Nodemailer al ser importado en una variable nos ofrece diferentes propiedades y métodos, el que utilizaremos en esta lectura será el método **“createTransport”**, el cual recibe como argumento un objeto de configuración que especifica el servicio(host) y las credenciales de autenticación, pero ¿Qué sería un Transport? Sucede que SMTP es el medio de transporte principal en Nodemailer para entregar mensajes. SMTP es también el protocolo utilizado entre diferentes hosts de correo electrónico, por lo que es verdaderamente universal.

Casi todos los proveedores de entrega de correo electrónico admiten el envío basado en SMTP, incluso si impulsan principalmente su envío basado en API. Las API pueden tener más funciones, pero el uso de estas también significa el bloqueo del proveedor, mientras que en el caso de SMTP solo se necesita cambiar las opciones de configuración para reemplazar un proveedor por otro y listo.

Veamos como se ve esto con el siguiente código en donde te muestro una sintaxis básica de este método:

```
let transporter = nodemailer.createTransport({
  service: <proveedor>,
  auth: {
    user: <correo electrónico host>,
    pass: <contraseña del electrónico host>,
  },
})
```

Estas propiedades se describen de la siguiente manera:

- **service:** Proveedor de correos electrónicos. Este debe ser un servicio dedicado al envío de correos, en esta lectura ocuparemos a Gmail para esto.
- **auth:** En formato de objeto, representa las credenciales del usuario que está ejecutando el envío del correo, y recibe las siguientes propiedades:
 - **user:** La dirección del correo que servirá como host para el envío del correo electrónico
 - **pass:** La contraseña del correo que servirá como host para el envío del correo electrónico

Para poder utilizar tu propio correo electrónico gmail, debes activar la opción en el siguiente [link](#). Gmail permite utilizar tu correo como proveedor, pero necesita que lo especifiques con tu cuenta personal. La opción debe quedar como te muestro en la siguiente imagen.

← Acceso de aplicaciones poco seguras

Algunos dispositivos y aplicaciones utilizan una tecnología de inicio de sesión poco segura, lo que aumenta la vulnerabilidad de tu cuenta. Te recomendamos que desactives el acceso de estas aplicaciones, aunque también puedes activarlo si quieres usarlas a pesar de los riesgos que conllevan. Desactivaremos este ajuste de forma automática si no lo utilizas. [Más información](#)

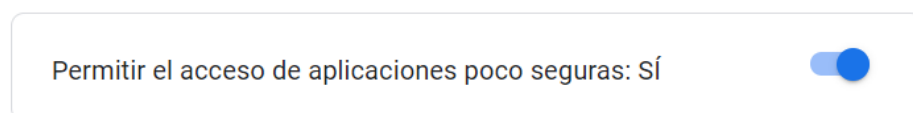


Imagen 2. Habilitar acceso de aplicaciones poco seguras en Gmail
Fuente: Desafío Latam

Por motivos de esta lectura se creó un correo electrónico Gmail, de manera que si no quieres usar tu correo personal y no tienes uno alterno, puedas utilizarlo sin problema.

Las credenciales de este correo electrónico son las siguientes:

- **Correo:** nodemailerADL@gmail.com
- **Contraseña:** desafiolatam

El proveedor de correos será el mismo Gmail, por lo que el código expuesto anteriormente quedará de la siguiente manera:

```
let transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'nodemailerADL@gmail.com',
    pass: "desafiolatam",
  },
})
```

Aunque no es indispensable, guardamos en una variable la llamada de este método para tener el código más limpio y seguir las buenas prácticas.

Cabe destacar que el usuario de autenticación para el envío de correo electrónico sirve solo como host del envío, es decir, no será reconocido como el emisor en sí del correo, este dato se toma para hacer referencia al representante del acto a nivel funcional, no obstante se puede especificar en las propiedades que veremos a continuación, que el correo fue enviado por otra dirección de correo electrónico. ¿No queda del todo claro? Te lo explico con un ejemplo: Cuando pides comida a domicilio, quien representa el envío de la comida, ¿tú o el restaurante que está enviando al repartidor? Por supuesto, sería el restaurante, no obstante fuiste tu quien emitió la orden y pagó la comida.

En síntesis, este usuario corresponde a la cuenta que tu proveedor de correos te asigne y ya que estamos usando Gmail, corresponderá a las credenciales del correo directamente.

Lo siguiente es ocupar un método de nuestra variable “transporter” llamado “sendMail”, el cual tiene la siguiente sintaxis:

```
transporter.sendMail(<opciones_correo>,<callback>)
```

Como puedes ver, necesitamos pasarle las opciones del correo electrónico que queremos enviar, este será un objeto que podemos crear en otra variable y tiene las siguientes propiedades básicas:

```
let mailOptions = {
  from: <correo electrónico que envía el mensaje>,
  to: <el o los correos a los que se quiere mandar el mensaje>,
  subject: <asunto del correo>,
  text: <contenido del correo>,
}
```

Tenemos como opciones ó propiedades principales las siguientes:

- **from:** Correo de origen, este dato debe ser tipo String.
- **to:** Destinatario o destinatarios, en caso de ser solo 1 correo, se puede especificar como String, pero si quisiéramos enviar el correo a varias personas para hacer un envío de correos electrónicos masivos, podemos pasar como vale un arreglo de Strings especificando todos los correos de destino.
- **subject:** Asunto del correo, este dato debe ser tipo String
- **text:** Mensaje en formato de texto plano.

Cabe destacar que la propiedad "text" puedes cambiarla por "html" y de esta manera enviar código HTML.

Para probar por primera vez Nodemailer, llenaremos los datos de la siguiente manera:

```
let mailOptions = {  
  from: 'nodemailerADL@gmail.com',  
  to: 'nodemailerADL@gmail.com',  
  subject: 'Nodemailer Test',  
  text: 'Probando... 1,2,3...',  
}
```

El segundo parámetro del método `sendMail` es un callback, el cual recibe como parámetro el error y el mensaje de éxito en ese mismo orden, por lo que podemos capturar cualquiera de estos escenarios y mandarlo por consola.

Ejercicio guiado: Mi primer envío de correo electrónico en Node

Desarrollar una aplicación básica que envíe un correo electrónico con el paquete nodemailer, pero antes deberás instalar el paquete, así que abre la consola y escribe el siguiente comando y luego sigue los siguientes pasos para llegar a la solución:

```
npm i nodemailer
```

Con esto tenemos todo listo para realizar nuestra primera prueba, sigue los pasos para hacer el envío de tu primer correo electrónico con nodemailer.

- **Paso 1:** Importar el paquete Nodemailer a una constante.
- **Paso 2:** Guardar en una variable “transporter” el llamado al método createTransport pasándole como configuración las siguientes propiedades:
 - service: gmail
 - auth:
 - user: nodemailerADL@gmail.com
 - pass: desaflatam
- **Paso 3:** Crear una variable “mailOptions” para definir las opciones del correo electrónico de prueba, las cuales serán las siguientes:
 - from: nodemailerADL@gmail.com
 - to: nodemailerADL@gmail.com
 - subject: Nodemailer Test
 - text: Probando... 1,2,3...
- **Paso 4:** Invoca el método sendMail de la instancia transporter pasándole como argumento las opciones del correo y en su segundo parámetro recibirás un callback, con el error y la data correspondiente al envío.

- **Paso 5:** Usa condicionales para imprimir por consola el error y la data correspondiente al envío en caso de existir.

```
// Paso 1
const nodemailer = require('nodemailer')

// Paso 2
let transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'nodemailerADL@gmail.com',
    pass: 'desafiolatam',
  },
})

// Paso 3
let mailOptions = {
  from: 'nodemailerADL@gmail.com',
  to: 'nodemailerADL@gmail.com',
  subject: 'Nodemailer Test',
  text: 'Probando... 1,2,3...',
}

// Paso 4
transporter.sendMail(mailOptions, (err, data) => {
  // Paso 5
  if (err) console.log(err)
  if (data) console.log(data)
})
```

Ahora si abres la terminal y levantas la aplicación deberás obtener lo que se muestra en la siguiente imagen:

```
→ ejerciciolectura git:(master) x node index.js
{
  accepted: [ 'nodemailerADL@gmail.com' ],
  rejected: [],
  envelopeTime: 617,
  messageTime: 723,
  messageSize: 325,
  response: '250 2.0.0 OK 1600801593 b28sm8355161qka.117 - gsmtpt',
  envelope: {
    from: 'nodemailerADL@gmail.com',
    to: [ 'nodemailerADL@gmail.com' ]
  },
  messageId: '<9a161c1b-7fd5-5cef-512c-ca22ebcb96b9@gmail.com>'
}
```

Imagen 3. Mensaje por consola indicando que el correo fue enviado con éxito.
Fuente: Desafío Latam

Con este mensaje tenemos la certeza de que el correo fue enviado sin problemas.
¿Pero es así realmente? Para verificarlo abre la cuenta de Gmail y revisa la bandeja de “recibidos” y deberás tener lo que te muestro en la siguiente imagen.

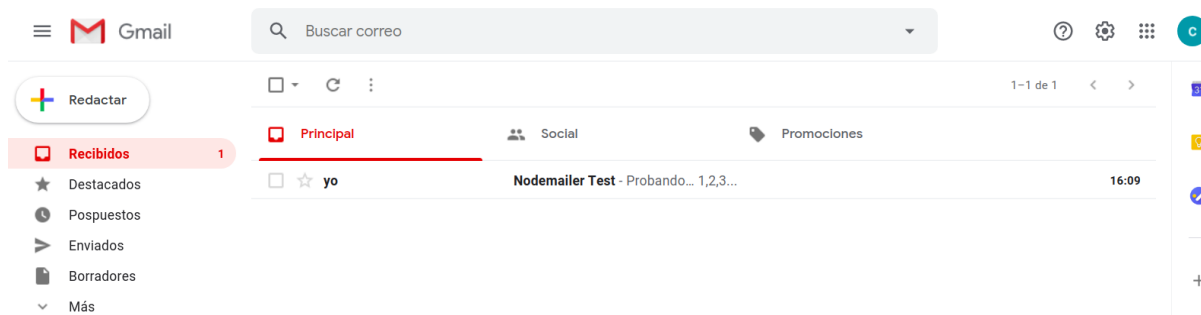


Imagen 4. Bandeja de entrada indicando que se recibió el correo electrónico.
Fuente: Desafío Latam

¡Maravilloso! Hemos logrado enviar un correo electrónico en simples pasos con el paquete nodemailer.

Ejercicio propuesto (1)

Basado en el ejercicio “Mi primer envío de correo electrónico en Node”, desarrolla una aplicación que al ser ejecutada envíe el siguiente HTML:

```
<h1>Desafío</h1>
<h2>LATAM</h1>
```

Envío de un correo electrónico a partir de una consulta HTTP

Competencia

- Construir un servidor que procese el envío de un correo electrónico a partir de una consulta HTTP

Introducción

Los servicios de correos electrónicos han evolucionado con el pasar del tiempo, hasta convertirse en parte de un sistema de multiservicios interconectados; para el caso de Gmail tenemos todo el mundo Drive que comparte funcionalidades, accesos variados para documentos y archivos de una manera bastante eficiente.

En este capítulo, aprenderás a crear un servidor capaz de enviar correos electrónicos, a través del protocolo SMTP usando a Gmail como host y proveedor de correos. De esta manera podrás agregar esta funcionalidad a tu stack de servicios y a tus aplicaciones de servidor, las cuales podrás conectar con tus aplicaciones en el lado del cliente por medio del protocolo HTTP, teniendo como resultado un sistema para la mensajería editable de correos electrónicos.

Servidor para correos electrónicos

Ahora que sabemos que podemos enviar correos electrónicos, lo siguiente será mezclar esta nueva habilidad con lo aprendido en sesiones anteriores, específicamente en la creación de servidores que disponibilizan rutas.

Ejercicio guiado: Servidor para correos electrónicos

Crear un servidor que reciba la información que se enviará en un correo desde el cliente. Inicialmente por medio de una consulta que realizaremos en el navegador con una URL que incluya los parámetros que capturaremos en el servidor, esta consulta próximamente será emitida por un formulario HTML, el cual recordemos por defecto emite una consulta GET con los parámetros escritos por un usuario en los inputs.

Al tener mezclada la lógica del servidor con la lógica de nodemailer podríamos encontrarnos con la incomodidad de tener todo en 1, por lo que ha llegado el momento de dividir en 2 archivos nuestra aplicación e importar en el principal (servidor), una función preparada para recibir los parámetros y asignarlos en las propiedades correspondientes. Es decir, necesitaremos tener 2 archivos, los cuales serán: `mailer.js` e `index.js`, estos archivos guardarán la lógica de nodemailer y la del servidor respectivamente.

Ya que tenemos la lógica para el envío de correos, pasa ese código a un archivo llamado `mailer.js` y sigue los siguientes pasos para hacerle las modificaciones pertinentes:

- **Paso 1:** Crear una función llamada "enviar" que reciba como parámetro los valores de "to", "subject" y "text".
- **Paso 2:** Exportar la función enviar como un módulo.

```
const nodemailer = require('nodemailer')

// Paso 1
function enviar(to, subject, text) {
  let transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'nodemailerADL@gmail.com',
      pass: 'desafiolatam',
    },
  })

  let mailOptions = {
    from: 'nodemailerADL@gmail.com',
    to,
    subject,
    text,
  }

  transporter.sendMail(mailOptions, (err, data) => {
    if (err) console.log(err)
    if (data) console.log(data)
  })
}

// Paso 2
module.exports = enviar
```

Teniendo entonces este archivo listo, sigue los siguientes pasos para la lógica del servidor en el archivo index.js

- **Paso 1:** Importar la función enviar del archivo mailer.js.
- **Paso 2:** Ya que la idea es recibir los valores desde el cliente en una consulta HTTP, importa el módulo url para poder extraer los parámetros de la query strings.
- **Paso 3:** Importar el módulo http para crear el servidor.
- **Paso 4:** Crear el servidor con el método createServer del módulo http.
- **Paso 5:** Guardar en variables los parámetros “para”, “asunto” y “contenido”.

- **Paso 6:** Crear la ruta raíz del servidor.
- **Paso 7:** Ejecutar la función enviar importada del archivo mailer.js pasándole como argumento los parámetros recibidos de la consulta

```
// Paso 1
const enviar = require('./mailer')
// Paso 2
const url = require('url')
// Paso 3
const http = require('http')
// Paso 4
http
  .createServer(function (req, res) {
    // Paso 5
    let { para, asunto, contenido } = url.parse(req.url, true).query
    // Paso 6
    if (req.url.startsWith('/')) {
      // Paso 7
      enviar(para, asunto, contenido)
    }
  })
  .listen(3000)
```

Ahora levanta el servidor y consúltalo con el siguiente link en donde tendremos como parámetro el mismo correo electrónico, pero nuestro asunto será "Servidor" y el contenido "ADL"

<http://localhost:3000/?para=nodemaileradl@gmail.com&asunto=Servidor&contenido=ADL>

En el navegador no veremos nada por ahora, pero en la consola debiste haber recibido el mensaje de éxito que te mostré en la imagen 3 y si revisas la bandeja de entrada verás lo que te muestro a continuación.

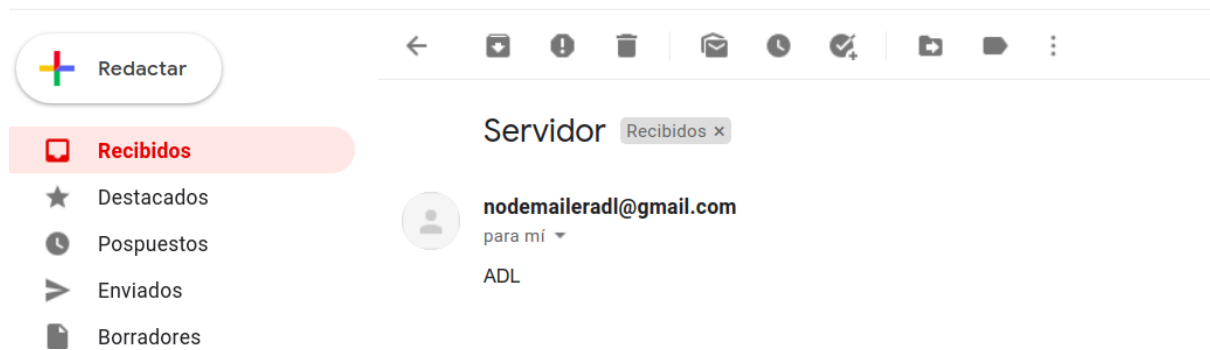


Imagen 5. Bandeja de entrada del correo electrónico
Fuente: Desafío Latam

Como puedes ver, todo está funcionando sin problemas, y ahora pudiendo ofrecer la función de nodemailer en nuestros servidores se nos abre una puerta enorme de posibilidades y funcionalidades nuevas a nuestras aplicaciones full stack.

Ejercicio propuesto (2)

Basado en el ejercicio de envío de correo con una consulta HTTP, desarrolla un servidor que disponibilice una ruta “enviar” que envíe un correo electrónico basado en los parámetros recibidos en la consulta y le devuelva al cliente un mensaje diciendo “Se procedió a intentar mandar el correo electrónico... revise su bandeja de entrada para confirmar que se haya enviado.”

Envío de correos electrónicos masivos

Competencia

- Construir un servidor que procese el envío de correos electrónicos masivos a partir de una consulta HTTP

Introducción

Con la práctica del capítulo anterior aprendiste a realizar el envío de un correo electrónico con valores estáticos y arbitrarios escritos en una URL, no obstante en el desarrollo laboral estos datos provienen de un formulario llenado desde una aplicación cliente.

En este capítulo aprenderás a enviar correos electrónicos masivos a partir de una formulario HTML que envíe en un input un conjunto de correos electrónicos separados por comas, teniendo en el lado del servidor la lógica para identificar cada dirección y emitir el envío del correo a todos.

Con esta habilidad adquirida, tu perfil como full stack developer se verá beneficiado y podrás construir sistemas más completos, con más funcionalidades que se pueden comunicar con tus usuarios a través del medio informativo más formal, los correos electrónicos.

Generando envío de correos electrónicos masivos

Es posible que te encuentres con la necesidad de mandar el mismo correo electrónico a varias personas al mismo tiempo, en vez de estar ejecutando varias veces la función en el servidor y cambiando las direcciones de destino, nodemailer nos permite pasarle en la propiedad "to" un arreglo con las direcciones que deseamos enviar el correo.

La lista de correos electrónicos en los sistemas de correos más populares se genera en un campo de texto separando con comas (,) las direcciones, es decir si quisiéramos mandar un mismo correo electrónico a 2 direcciones diferentes habría que escribir algo como esto: "primera_direccion@gmail.com, segunda_dirección@gmail.com".

Esto se escribiría dentro de un formulario a nivel de cliente, en el backend podríamos separar ese string con el método split de JavaScript, especificando que la separación se realizará basada en el carácter ",".

Preparando formulario

Para esto procedemos a preparar un archivo index.html básico que contenga un formulario incluyendo los 3 campos que necesitamos enviarle al servidor.

Puedes ocupar el siguiente código o generar tu propio formulario si deseas.

```
<form action="http://localhost:3000/correos">
  <label>Para: </label> <input name="para" /><br />
  <label>Asunto: </label> <input name="asunto" /><br />
  <label>Contenido: </label><textarea name="contenido"></textarea><br />
  <button>Enviar correo</button>
</form>
```

Como puedes notar, el formulario redireccionará los datos a una ruta en el servidor llamada "correos", esta ruta aún no está creada por lo que habrá que desarrollarla, dejándole la ruta principal al formulario HTML.

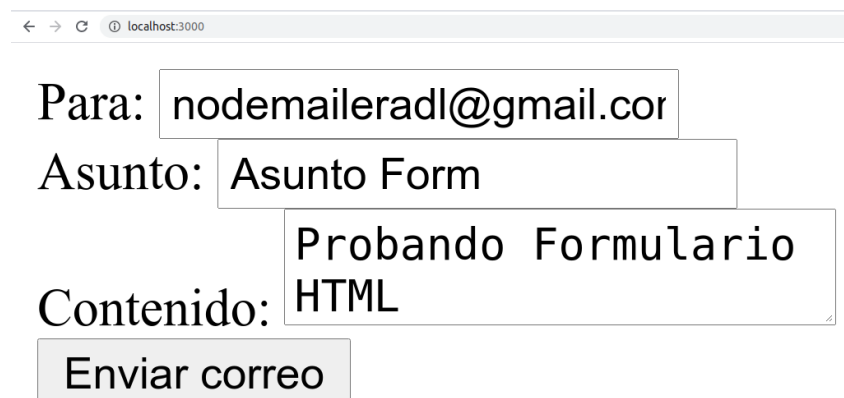
Ya que necesitaremos devolver este archivo HTML desde el servidor, necesitaremos por supuesto importar el módulo File System. Prosigue con los siguientes pasos para modificar la lógica en nuestro servidor:

- **Paso 1:** Importar el módulo File System.
- **Paso 2:** Especificar en la ruta raíz que se devolverá contenido HTML con la cabecera correspondiente.

- **Paso 3:** Devolver el contenido del archivo index.html en la ruta raíz.
- **Paso 4:** Crear la ruta “correos”, la cual será la encargada de ejecutar la función “enviar” importada del archivo mailer.js

```
const enviar = require('./mailer')
const url = require('url')
const http = require('http')
// Paso 1
const fs = require('fs')
http
  .createServer(function (req, res) {
    let { para, asunto, contenido } = url.parse(req.url, true).query
    if (req.url == '/') {
      // Paso 2
      res.setHeader('content-type', 'text/html')
      // Paso 3
      fs.readFile('index.html', 'utf8', (err, data) => {
        res.end(data)
      })
    }
    // Paso 4
    if (req.url.startsWith('/correos')) {
      enviar(para, asunto, contenido)
    }
  })
  .listen(3000)
```

Ahora intenta consultar al servidor desde tu navegador y obtendrás lo que te muestro en la siguiente imagen:



← → 🔄 📄 localhost:3000

Para:

Asunto:

Contenido:

Imagen 6. Formulario devuelto por el servidor para el envío de un correo electrónico
Fuente: Desafío Latam

Si llenas el formulario con la información que te muestro en la imagen 6 y presionas el botón “Enviar correo”, deberás recibir en la bandeja de entrada lo que te muestro en la siguiente imagen:

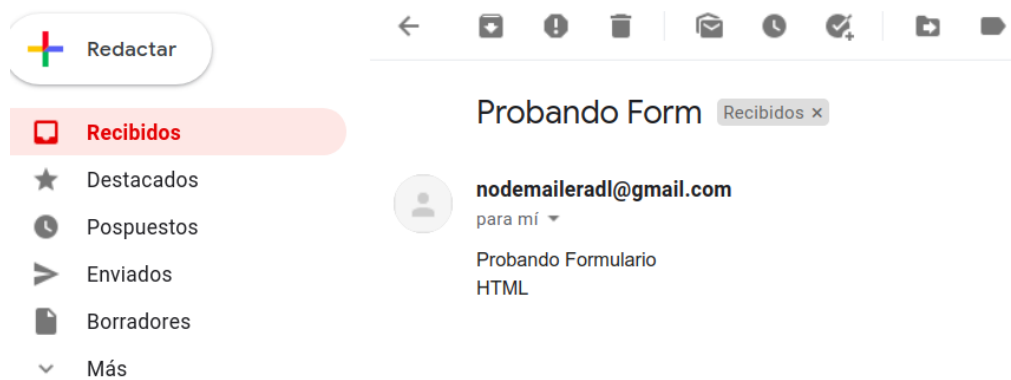


Imagen 7. Correo recibido con los datos escritos desde un formulario HTML
Fuente: Desafío Latam

Ejercicio propuesto (3)

Basado en el ejercicio donde probamos el formulario HTML en el navegador, crea un servidor que envíe el correo electrónico sólo si los 3 campos son distintos a un String vacío, respondiendo con un mensaje indicando que se deben llenar todos los campos.

Envío de correos masivos

Ahora que probamos con éxito el formulario HTML devuelto en la ruta raíz del servidor, solo falta procesar el String recibido en el parámetro “para”, con el objetivo de dividir en un arreglo los correos escritos separados por “,”.

Para esto deberás seguir el siguiente paso:

- **Paso 1:** Enviar el parámetro “para” aplicado con el método split, especificando como argumento el carácter “,”

```
const enviar = require('./mailer')
const url = require('url')
const http = require('http')
const fs = require('fs')
http
  .createServer(function (req, res) {
    let { para, asunto, contenido } = url.parse(req.url, true).query
    if (req.url === '/') {
      res.setHeader('content-type', 'text/html')
      fs.readFile('index.html', 'utf8', (err, data) => {
        res.end(data)
      })
    }
    if (req.url.startsWith('/correos')) {
      // Paso 1
      enviar(para.split(','), asunto, contenido)
    }
  })
  .listen(3000)
```

Ahora con esto solo tendrás que escribir las direcciones que desees separadas por “,” y se enviará el correo electrónico a todos.

Ejercicio propuesto (4)

Basado en el ejercicio donde enviamos correos electrónicos masivos separados por comas, devuelve un mensaje de error al cliente en caso de intentar mandar un string en el campo “Para” que no incluya la coma.

Para que compruebes efectivamente el envío masivo de los correos, pídele a tus compañeros sus direcciones de correo electrónico para usarlos en tu código y pregúntales luego si les llegó el correo que enviaste.

Resumen

A lo largo de esta lectura cubrimos:

- El protocolo SMTP para el envío de correos electrónicos.
- El paquete nodemailer, herramienta para hacer envíos de correos electrónicos con Node.
- Propiedades del paquete nodemailer para el envío de correos electrónicos.
- Desarrollo de una aplicación Node que al ser ejecutada envía un correo electrónico.
- Desarrollo de un servidor con Node que disponibiliza una ruta para el envío de correos electrónicos masivos.

Solución de los ejercicios propuestos

1. Basado en el ejercicio “Mi primer envío de correo electrónico en Node”, desarrolla una aplicación que al ser ejecutada envíe el siguiente HTML:

```
<h1>Desafío</h1>
<h2>LATAM</h1>
```

```
const nodemailer = require('nodemailer')

let transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'nodemailerADL@gmail.com',
    pass: 'desafiolatam',
  },
})

let mailOptions = {
  from: 'nodemailerADL@gmail.com',
  to: 'nodemailerADL@gmail.com',
  subject: 'Nodemailer Test',
  html: `
    <h1>Desafío</h1>
    <h2>LATAM</h1>
  `,
}

transporter.sendMail(mailOptions, (err, data) => {
  if (err) console.log(err)
  if (data) console.log(data)
})
```

2. Basado en el ejercicio de envío de correo con una consulta HTTP, desarrolla un servidor que disponibilice una ruta “enviar” que envíe un correo electrónico basado en los parámetros recibidos en la consulta y le devuelva al cliente un mensaje diciendo “Se procedió a intentar mandar el correo electrónico... revise su bandeja de entrada para confirmar que se haya enviado.”

```
const enviar = require('./mailer')
const url = require('url')
const http = require('http')
http
  .createServer(function (req, res) {
    let { para, asunto, contenido } = url.parse(req.url, true).query

    if (req.url.startsWith('/enviar')) {
      enviar(para, asunto, contenido)
      res.end(
        'Se procedió a intentar mandar el correo electrónico... revise su bandeja de entrada para confirmar que se haya enviado.'
      )
    }
  })
  .listen(3000)
```

3. Basado en el ejercicio donde probamos el formulario HTML en el navegador, crea un servidor que envíe el correo electrónico sólo si los 3 campos son distintos a un String vacío, respondiendo con un mensaje indicando que se deben llenar todos los campos.

```
const enviar = require('./mailer')
const url = require('url')
const http = require('http')
const fs = require('fs')
http
  .createServer(function (req, res) {
    let { para, asunto, contenido } = url.parse(req.url, true).query
    if (req.url === '/') {
      res.setHeader('content-type', 'text/html')
      fs.readFile('index.html', 'utf8', (err, data) => {
        res.end(data)
      })
    }
    if (req.url.startsWith('/correos')) {
      para !== '' && asunto !== '' && contenido !== ''
        ? enviar(para.split(','), asunto, contenido)
        : res.write('Faltan campos por llenar')
    }
  })
  .listen(3000)
```


- Basado en el ejercicio donde enviamos correos electrónicos masivos separados por comas, devuelve un mensaje de error al cliente en caso de intentar mandar un string en el campo "Para" que no incluya la coma.

```
const enviar = require('./mailer')
const url = require('url')
const http = require('http')
const fs = require('fs')
http
  .createServer(function (req, res) {
    let { para, asunto, contenido } = url.parse(req.url, true).query
    if (req.url === '/') {
      res.setHeader('content-type', 'text/html')
      fs.readFile('index.html', 'utf8', (err, data) => {
        res.end(data)
      })
    }
    if (req.url.startsWith('/correos')) {
      para !== '' && asunto !== '' && contenido !== '' &&
      para.includes(',')
        ? enviar(para.split(','), asunto, contenido)
        : res.write('Faltan campos por llenar o se está intentando un
correo con solo 1 dirección')
    }
  })
  .listen(3000)
```