

# PRÁCTICA 2: SERVICIO DE MENSAJERÍA P2P

DESARROLLO E IMPLEMENTACIÓN POR:  
ALEJANDRO PALENCIA PALOMO Y LUCÍA MONEDERO GRIFO

## INTRODUCCIÓN

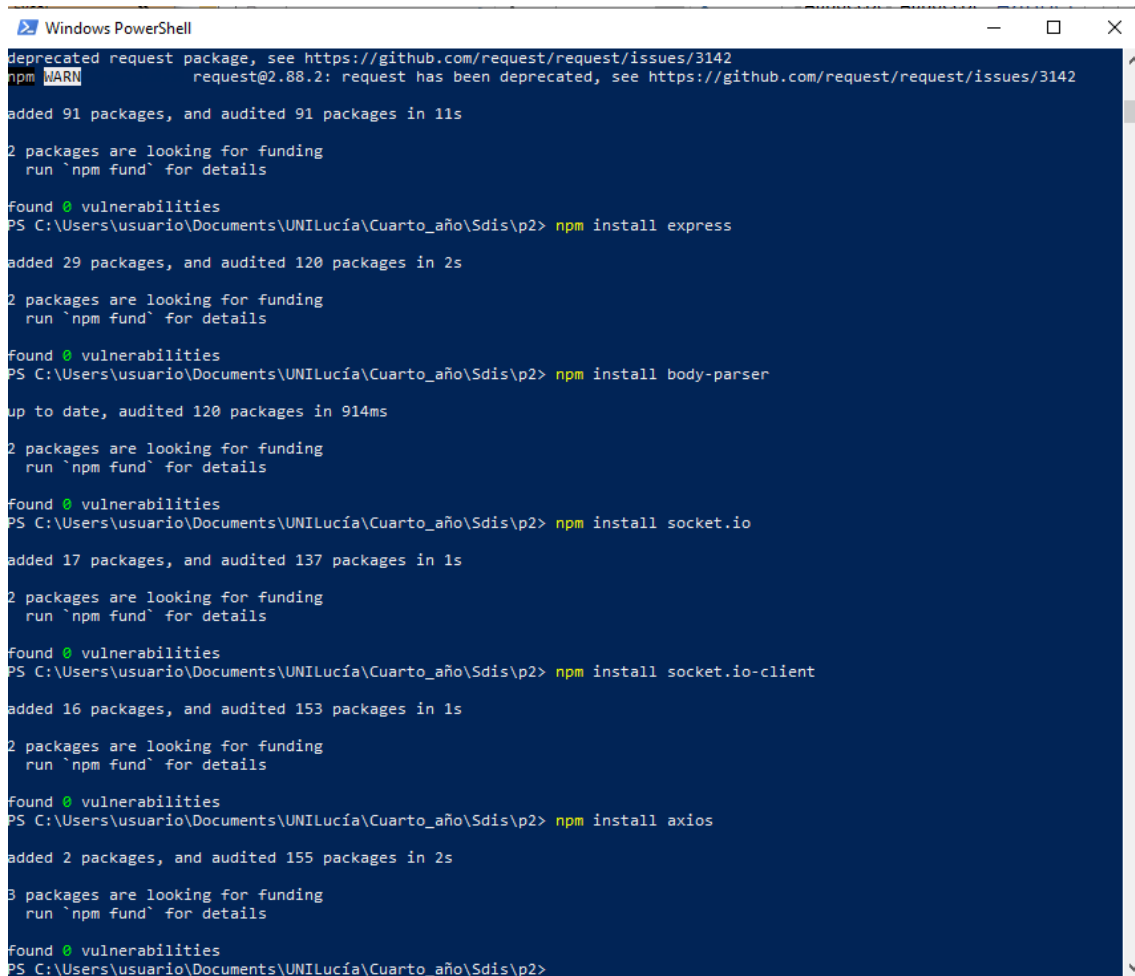
Para la realización de la práctica, vamos a emplear un servicio Cloud, para que los nodos sean visibles de forma pública. Al arrancar el servicio, nos va a dar una URL y cuando nos conectemos a dicha URL, nos estaremos conectando al servidor local que tenemos en localhost.

Vamos a usar conexiones de todos los nodos con todos los nodos, para no tener que emplear “flooding” y evitar así llenar la red de excesivos mensajes.

## DESARROLLO DE LA PRÁCTICA

### 1.INSTALACIÓN

En primer lugar, hacemos mediante *npm install* la instalación de todas las librerías necesarias para la realización de la práctica:



```
Windows PowerShell
deprecated request package, see https://github.com/request/request/issues/3142
npm WARN request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
added 91 packages, and audited 91 packages in 11s
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2> npm install express
added 29 packages, and audited 120 packages in 2s
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2> npm install body-parser
up to date, audited 120 packages in 914ms
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2> npm install socket.io
added 17 packages, and audited 137 packages in 1s
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2> npm install socket.io-client
added 16 packages, and audited 153 packages in 1s
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2> npm install axios
added 2 packages, and audited 155 packages in 2s
3 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2>
```

Aquí se puede ver más de cerca la instalación de una de ellas en concreto (*axios*):

```
C:\Users\Usuario>npm install axios
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\Usuario\package.json'
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\Usuario\package.json'
npm WARN Usuario No description
npm WARN Usuario No repository field.
npm WARN Usuario No README data
npm WARN Usuario No license field.

+ axios@0.20.0
added 2 packages from 4 contributors and audited 266 packages in 1.698s

3 packages are looking for funding
  run `npm fund` for details

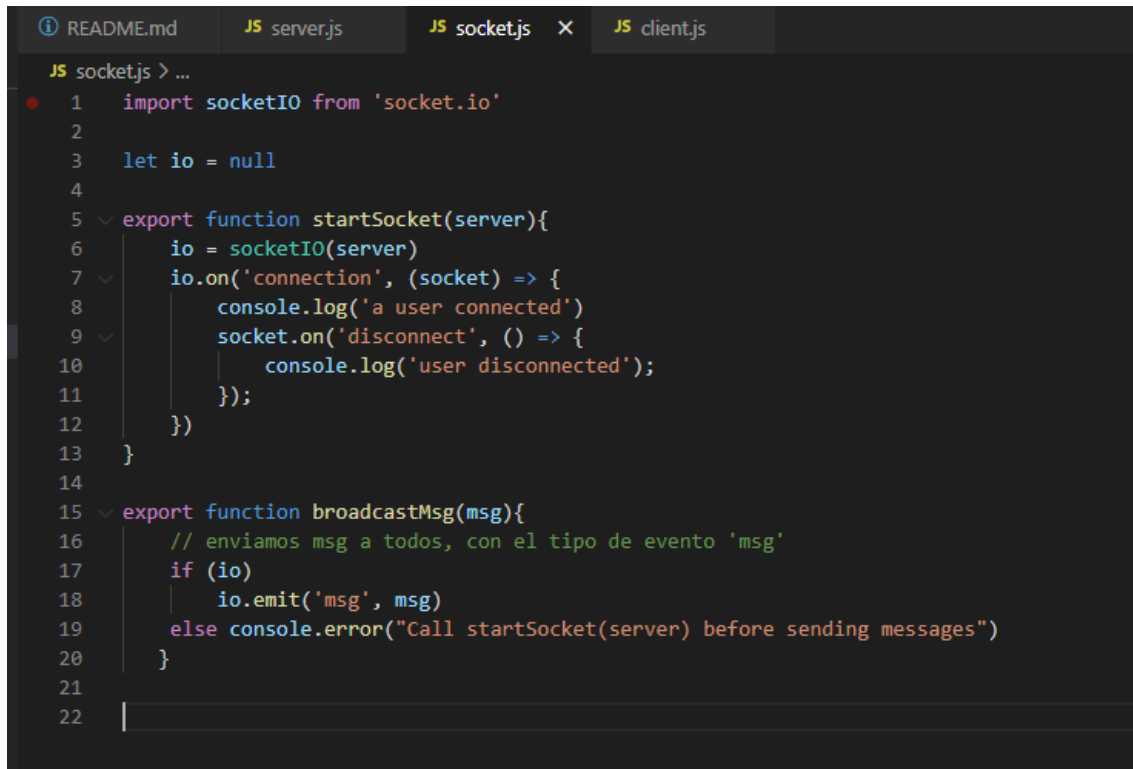
found 0 vulnerabilities
```

## 2.SERVIDOR DE MENSAJERÍA, EN MODO ECO

Creamos el fichero server.js como se pide:

```
JS server.js > ...
1  import express from 'express'
2  import {startSocket} from './socket.js'
3  import {broadcastMsg} from './socket.js'
4  const PORT = 8080
5  const app = express()
6  app.get('/', (req, res) => {
7    res.send('Hello World I am running locally')}
8  const server = app.listen(PORT, () => console.log("listening at localhost:"+PORT))
9  startSocket(server)
10
11  app.put('/msg', (req, res) => {
12    console.log('msg received')
13    const msg = req.body
14    res.send('OK')
15    // send the message back (eco)
16    broadcastMsg(msg)
17  })
18
```

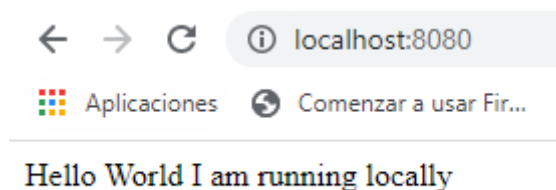
Para arrancar también el servidor de WebSocket, creamos la función *startSocket* en el fichero *socket.js*. Además, creamos la función *broadcastMsg* para que envíe un mensaje a todos los usuarios conectados a través de websocket, como se aprecia en la captura:



```
JS socketjs > ...
1 import socketIO from 'socket.io'
2
3 let io = null
4
5 export function startSocket(server){
6   io = socketIO(server)
7   io.on('connection', (socket) => {
8     console.log('a user connected')
9     socket.on('disconnect', () => {
10       console.log('user disconnected');
11     });
12   })
13 }
14
15 export function broadcastMsg(msg){
16   // enviamos msg a todos, con el tipo de evento 'msg'
17   if (io)
18     io.emit('msg', msg)
19   else console.error("Call startSocket(server) before sending messages")
20 }
21
22
```

Para comprobar que lo que hemos hecho funciona, vamos a conectarnos a *localhost:8080* desde un navegador, tras iniciar el servidor con *'node ./server.js'*, y si el mensaje *'Hello World I am running locally'* aparece, significará que todos los pasos están correctos hasta el momento.

Esto se puede apreciar en la siguiente imagen:



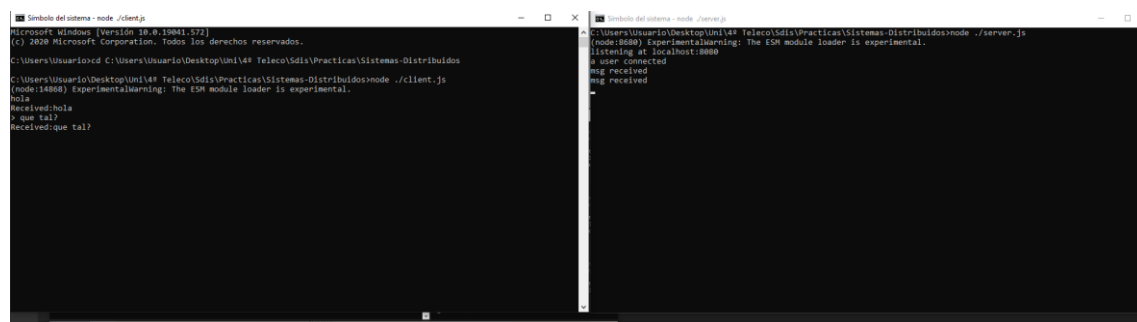
Después, hemos tenido que crear el fichero `client.js` para poder llamar al método `put`, el cual hemos añadido anteriormente para que la función del api *RESTful* que recibe mensajes retransmita con `broadcastMsg` lo que le llegue mediante este `put /msg` ya mencionado.

De esta manera, podremos probar nuestro servidor de ECO y ver que funciona como se espera:

```
JS client.js > ...
1  import axios from 'axios'
2  import io from 'socket.io-client';
3  import readline from 'readline'
4  const server = 'http://localhost:8080'
5  const socket = io(server)
6  socket.on('msg', function (msg) {
7    console.log(msg)
8  })
9
10 async function sendMsg(msg){
11   try {
12     await axios.put(server + '/msg', msg)
13   }
14   catch (error){
15     console.log(error)
16   }
17 }
18 const rl = readline.createInterface(process.stdin, process.stdout)
19 rl.on('line', function (line) {
20   sendMsg(line)
21 });
22
```

A continuación, se muestra una captura de un ejemplo de ejecución de cliente y servidor:

(Ampliar para ver en más detalle)



The image shows two terminal windows side-by-side. The left window is titled 'Símbolo del sistema - node ./client.js' and shows the execution of the client script. It displays the prompt 'holo' and the response 'que tal?'. The right window is titled 'Símbolo del sistema - node ./server.js' and shows the execution of the server script. It displays the message 'Listening at localhost:8080' and three instances of 'msg received'.

### 3.COMPLETANDO LA RED P2P

En esta parte de la práctica, vamos a hacer una serie de modificaciones para que los nodos puedan, a parte de recibir mensajes desde localhost, recibirlos desde otras máquinas.

Para esto, vamos a usar ngrok, que es un servicio de internet que nos sirve para crear un túnel cifrado que nos permita conectar los nodos desde una IP pública, ya que a menudo, cuando estamos creando una red P2P nos encontramos con problemas con los cortafuegos y con el uso de IPs privadas, que nos obligan a conectarnos a internet a través de NAT. Estos problemas se solucionan al emplear ngrok, el cual nos va a permitir crear de forma gratuita 4 túneles, recibiendo hasta 40 conexiones por minuto.

Para implementar todo esto, creamos el fichero ngrok.js, como se ve a continuación:

```
JS ngrok.js > [?] startNGrok
1  import ngrok from 'ngrok'
2
3  let ngURL = ''
4
5  export const startNGrok = async function(PORT) {
6      const url = await ngrok.connect({
7          proto : 'http',
8          addr : PORT,
9      }, )
10     ngURL = url
11     console.log (url)
12     return url
13 }
```

Por otro lado, ha habido que hacer modificaciones en el fichero server.js para adaptarlo a la nueva funcionalidad que queremos implementar(para peticiones get y put):

```
JS server.js > [?] start
1  import express from 'express'
2  import {startSocket, broadcastMsg} from './socket.js'
3  import bodyParser from 'body-parser'
4  import ngrok from 'ngrok'
5  import axios from 'axios'
6
7  import {startNGrok} from './ngrok.js'
8  //import {fasync, awaitAll} from './utils.js'
9
10 //const PORT = 8080
11 async function start(ID, PORT, CN_URL){
12
13
14     // Ngrok para obtener la url del servidor
15     let ngURL = ''
16     try {
17         ngURL = await startNGrok(PORT)
18     }
19     catch(error){
20         throw new Error(error)
21     }
22
23     let n = {url:ngURL,id:ID}
24     let lista_nodos = new Array(n)
25
26     // Inicialización del objeto app
27     const app = express()
28
29     // Parsear el body de las peticiones
30     app.use(bodyParser.json())
31     app.use(bodyParser.urlencoded({ extended: false }))
32
33     //Configuraciones de servidor
34     app.get('/', (req, res) => {
35         res.send('Hello World I am running locally')
36     })
37 }
```

```

6
7 app.get('/nodes', (req, res) => {
8   res.send(lista_nodos)
9 })
10
11
12 app.get('/nodes/:id?', (req, res) => {
13   res.send(lista_nodos.find(Node => Node.id == req.params.id))
14 })
15
16
17 app.put('/msg', (req, res) => {
18   console.log('msg received')
19   const msg = req.body
20   res.send('OK')
21   // send the message back (eco)
22   broadcastMsg(msg)
23 })
24
25 app.put('/nodes/:id?', (req, res) => {
26   let object_JSON = {id: req.params.id, url: req.body}
27   lista_nodos = [...lista_nodos, object_JSON]
28 })
29
30 process.on('SIGINT', async (code) => {
31   await console.log('Process before Exit event with code: ', code) //Funcionalidad para cerrar servidor con ctrl+c
32   process.exit()
33 }) //Aquí iría todo el código para configurar con Express los servidores
34
35
36 //let longitud_nodos = lista_nodos.length
37 const url_server = ngURL
38 let lista = ''
39 if (args.length === 3){
40   try{
41     const lista = await axios.get(CN_URL+'/nodes')
42     await Promise.all(lista.data.map(x => axios.put(x.url+'/nodes/'+ID, {url:url_server})))
43     console.log(lista.data)
44   }
45   catch(error){
46     console.log(error)
47   }
48 }

```

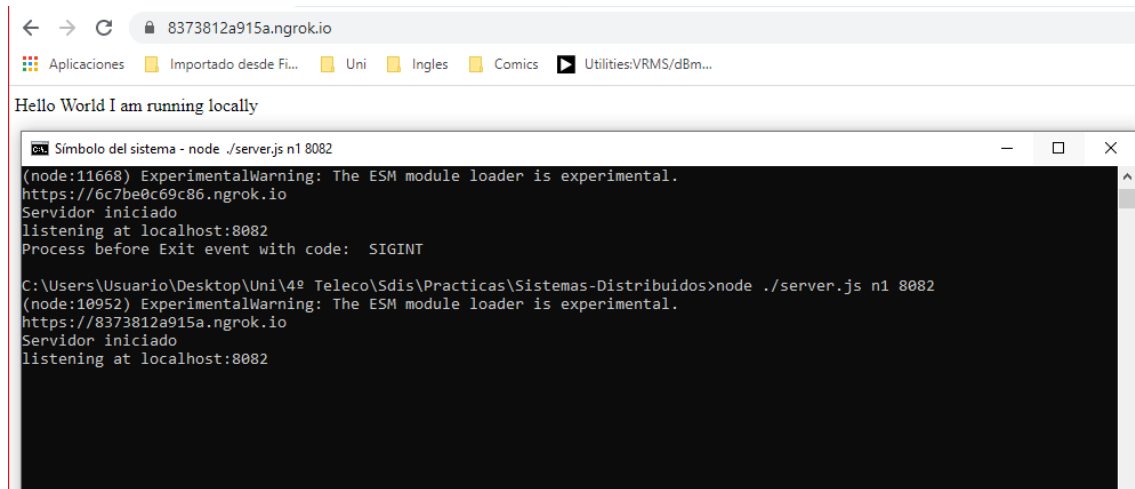
```

75   }
76   catch(error){
77     console.log(error)
78   }
79 }
80
81
82 // Una vez configurado el servidor, se pone a la escucha
83 const server = app.listen(PORT, () => console.log("listening at localhost:"+PORT))
84 startSocket(server)
85
86 }
87
88 // Código añadido para coger puerto por línea de comandos
89 const args = (method) Console.log(message?: any, ...optionalParams: any[]): void (+2 overloads)
90 if (args.l
91   console Prints to stdout with newline.
92   console.log("Por defecto se escucha en el puerto 8080 y no se conecta a ningún nodo")
93 }
94 else {
95   const ID = args.length > 0 ? args[0] : 'NOID' //Pasar ID del nodo como primer argumento
96   const PORT = args.length > 1 ? parseInt(args[1]) : 8080 // Puerto por defecto en el segundo argumento
97   const Node = args.length > 2 ? args[2] : '' // La dirección del nodo al que conectar es el tercer argumento
98   // Fin lectura argumentos
99
100   start(ID, PORT, Node).then(()=>console.log("Servidor iniciado"))
101 }
102
103

```

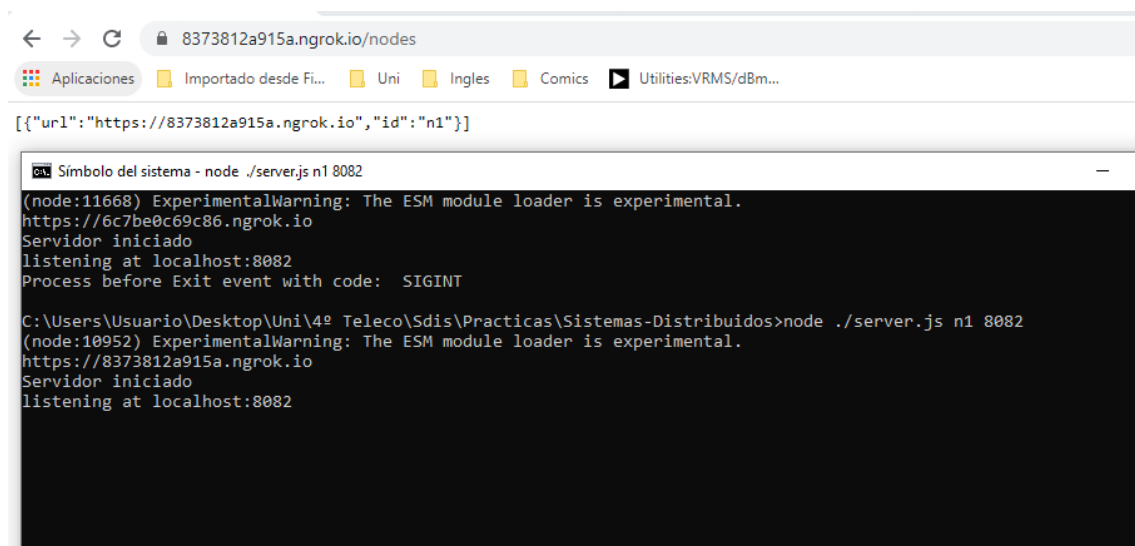
Para ejecutarlo, vamos a la terminal tecleando “*cmd*”. Después, vamos a dirigirnos a la carpeta en la que estemos realizando nuestro proyecto (con el comando “*cd*”), y posteriormente escribimos “*node ./server.js n1 8082*” para ejecutar el server con el nodo *n1* en el puerto *8082* (podríamos elegir otro puerto).

De esta manera, tendremos el servidor iniciado, y *ngrok* nos dará una URL que tendremos que poner en el navegador que estemos usando.



Así, conseguiremos el primer objetivo, es decir, que aparezca el mensaje “Hello World I am running locally”.

Para cumplir con lo siguiente que se nos pide, si ahora escribimos */nodes* a continuación de esta URL, veremos que nos aparece el siguiente mensaje con los datos de la URL y el ID de nuestro nodo.





El siguiente objetivo es que, a partir de los pasos que hemos hecho anteriormente, en lugar de conectarnos únicamente con 1 nodo, nos conectemos con 2 nodos y que se muestre en el browser, es decir, una lista de los nodos conectados.

Para hacer esto, procedemos de manera similar a como lo hemos hecho antes:

Ejecutamos un server para el nodo n1 en el puerto 8081:

```
Símbolo del sistema - node ./server.js n1 8081
Microsoft Windows [Versión 10.0.19042.630]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.
C:\Users\usuario>cd C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2
C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2>node ./server.js n1 8081
https://4e4dab341096.ngrok.io
Servidor iniciado
listening at localhost:8081
```

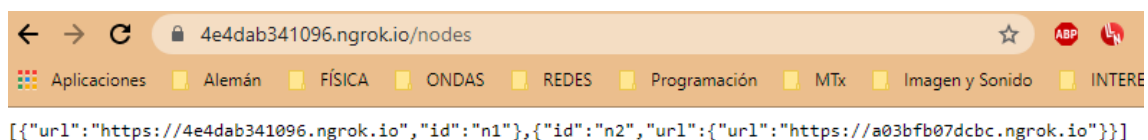
Copiamos la URL dada en el browser y nos aparecerá de nuevo el mensaje de “Hello World I am running locally”. Después, hacemos un /nodes y nos aparece el mensaje superior con la URL y el id del primer nodo. Hasta aquí, todo es igual al paso anterior.

El cambio surge ahora, cuando abrimos otra terminal y ejecutamos el server de la siguiente forma: “node ./server.js n2 8082 https://4e4dab341096.ngrok.io” es decir, añadiendo como argumento la URL dada al lanzar el primer server para poder conectarnos con el nuevo nodo a la dirección del otro nodo.

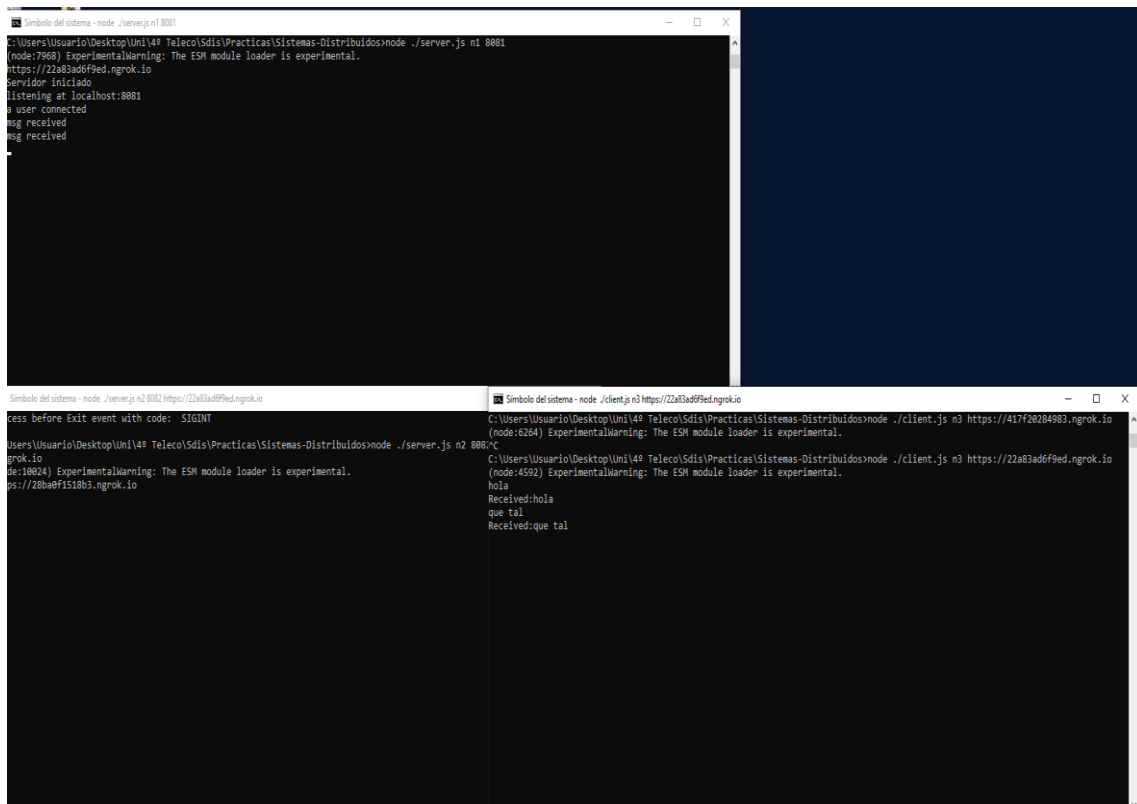
```
Símbolo del sistema - node ./server.js n1 8081
Microsoft Windows [Versión 10.0.19042.630]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.
C:\Users\usuario>cd C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2
C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2>node ./server.js n1 8081
https://4e4dab341096.ngrok.io
Servidor iniciado
listening at localhost:8081

Símbolo del sistema - node ./server.js n2 8082 https://4e4dab341096.ngrok.io
Microsoft Windows [Versión 10.0.19042.630]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.
C:\Users\usuario>cd C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2
C:\Users\usuario\Documents\UNILucía\Cuarto_año\Sdis\p2>node ./server.js n2 8082 https://4e4dab341096.ngrok.io
https://a03bfb07dcbc.ngrok.io
```

Si recargamos la página en la que anteriormente sólo nos mostraba una URL y un id de un nodo, podremos ver que se nos añade una segunda URL y un segundo id del nuevo nodo que se ha conectado al primero, como se puede ver a continuación:



Además, si conectamos un cliente con el puerto del servidor principal, se envían y se reciben de manera correcta los mensajes:



The image shows two Windows command prompt windows. The top window is titled 'Símbolo del sistema - node ./server.js n1 8081'. It shows the execution of 'node ./server.js n1 8081', a warning about the ESM module loader, and the server starting and listening on localhost:8081. It then shows a client connecting and sending two messages: 'hola' and 'que tal', both of which are received correctly. The bottom window is titled 'Símbolo del sistema - node ./client.js n3 https://22a83ad6f9ed.ngrok.io'. It shows the execution of 'node ./client.js n3 https://22a83ad6f9ed.ngrok.io', a warning about the ESM module loader, and the client sending two messages: 'hola' and 'que tal', both of which are received correctly.

```
Símbolo del sistema - node ./server.js n1 8081
C:\Users\Usuario\Desktop\Uni\4º Teleco\Sdis\Practicas\Sistemas-Distribuidos>node ./server.js n1 8081
(node:7968) ExperimentalWarning: The ESM module loader is experimental.
https://22a83ad6f9ed.ngrok.io
Servidor iniciado
Listening at localhost:8081
a user connected
msg received
msg received

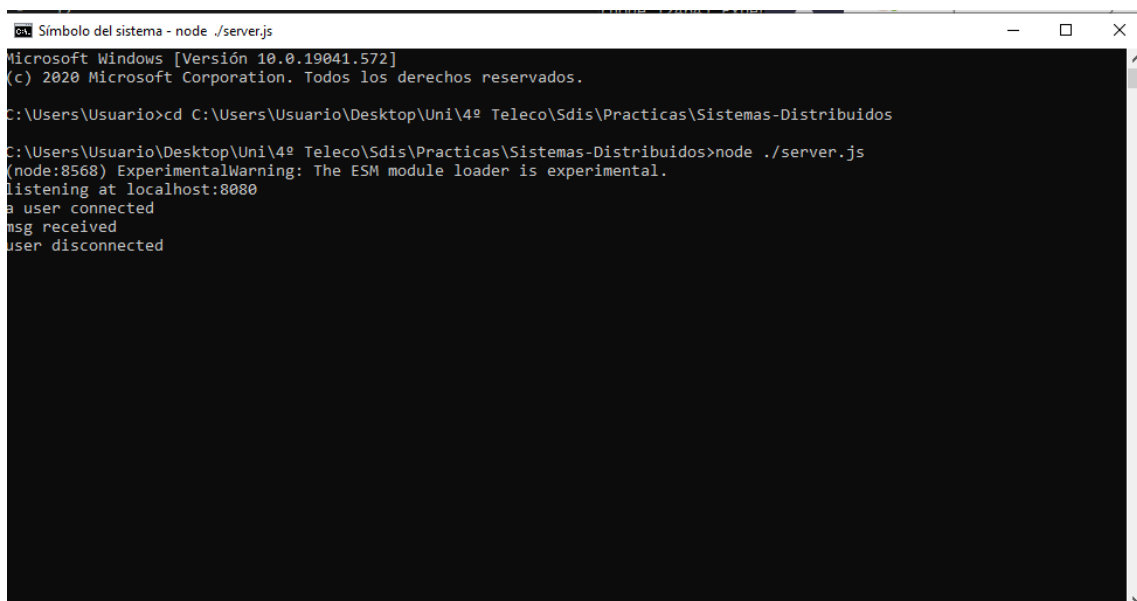
Símbolo del sistema - node ./client.js n3 https://22a83ad6f9ed.ngrok.io
C:\Users\Usuario\Desktop\Uni\4º Teleco\Sdis\Practicas\Sistemas-Distribuidos>node ./client.js n3 https://22a83ad6f9ed.ngrok.io
(node:6264) ExperimentalWarning: The ESM module loader is experimental.
hola
Received:hola
que tal
Received:que tal

Símbolo del sistema - node ./server.js n2 8080
C:\Users\Usuario\Desktop\Uni\4º Teleco\Sdis\Practicas\Sistemas-Distribuidos>node ./server.js n2 8080
(node:8568) ExperimentalWarning: The ESM module loader is experimental.
Listening at localhost:8080
a user connected
msg received
user disconnected
```

Por tanto, se cumple así el objetivo pedido en esta parte de la práctica.

#### 4.ERRORES ENCONTRADOS

Sin embargo, hubo algunos errores mientras escribíamos el código que, tras realizar algunas consultas al profesor, pudimos solucionar para que se produjese esta correcta ejecución. En las siguientes imágenes se ven algunos de nuestros errores principales:



The image shows a Windows command prompt window titled 'Símbolo del sistema - node ./server.js'. It shows the execution of 'node ./server.js', a warning about the ESM module loader, and the server starting and listening on localhost:8080. It then shows a client connecting and sending a message 'hola', which is received correctly. However, the client then disconnects, and the server logs 'user disconnected'.

```
Símbolo del sistema - node ./server.js
Microsoft Windows [Versión 10.0.19041.572]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Usuario>cd C:\Users\Usuario\Desktop\Uni\4º Teleco\Sdis\Practicas\Sistemas-Distribuidos
C:\Users\Usuario\Desktop\Uni\4º Teleco\Sdis\Practicas\Sistemas-Distribuidos>node ./server.js
(node:8568) ExperimentalWarning: The ESM module loader is experimental.
Listening at localhost:8080
a user connected
msg received
user disconnected
```

