PRÁCTICA 2: SERVICIO DE MENSAJERÍA P2P

eps | Sistemas Distribuidos

desarrollo e implementación por:

alejandro Palencia palomo y lucía monedero grifo

2020

**INTRODUCCIÓN**

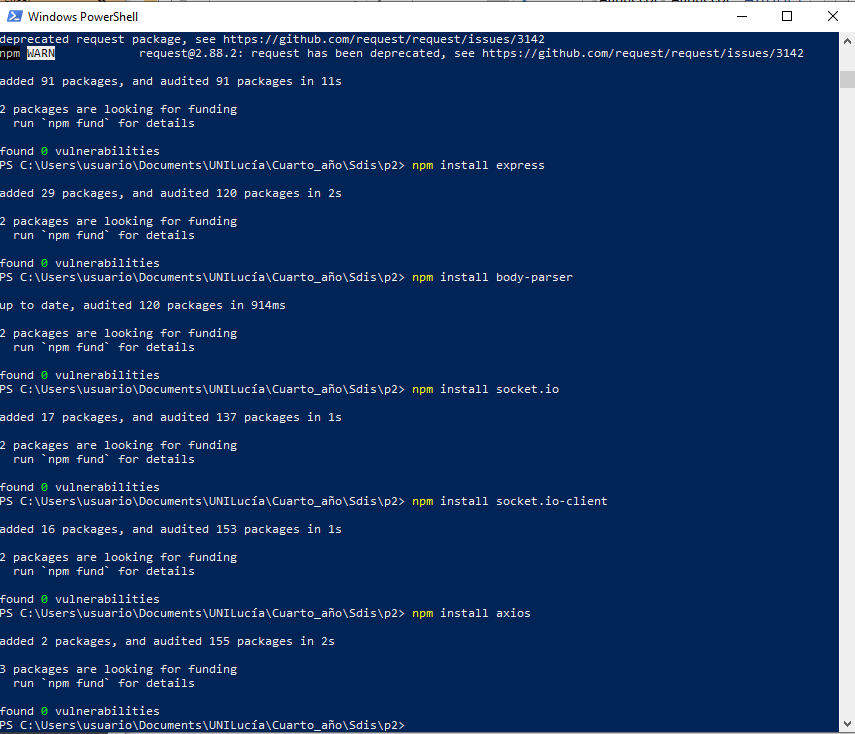
Para la realización de la práctica, vamos a emplear un servicio Cloud, para que los nodos sean visibles de forma pública. Al arrancar el servicio, nos va a dar una URL y cuando nos conectemos a dicha URL, nos estaremos conectando al servidor local que tenemos en localhost.

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.

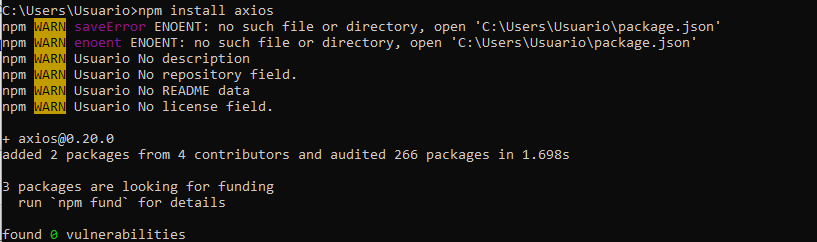
**DESARROLLO DE LA PRÁCTICA**

**1.INSTALACIÓN**

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

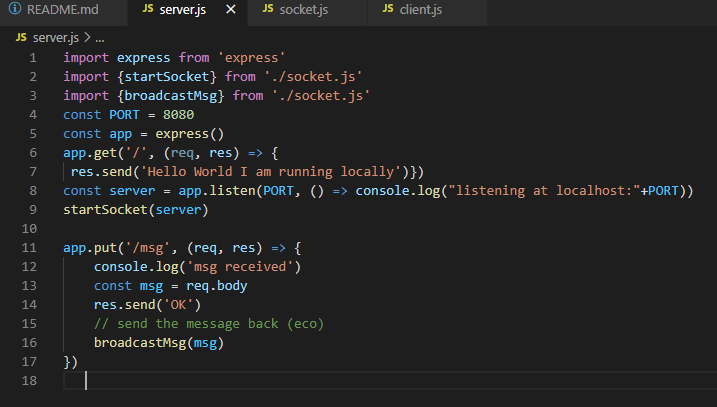


Aquí se puede ver más de cerca la instalación de una de ellas en concreto (*axios*):

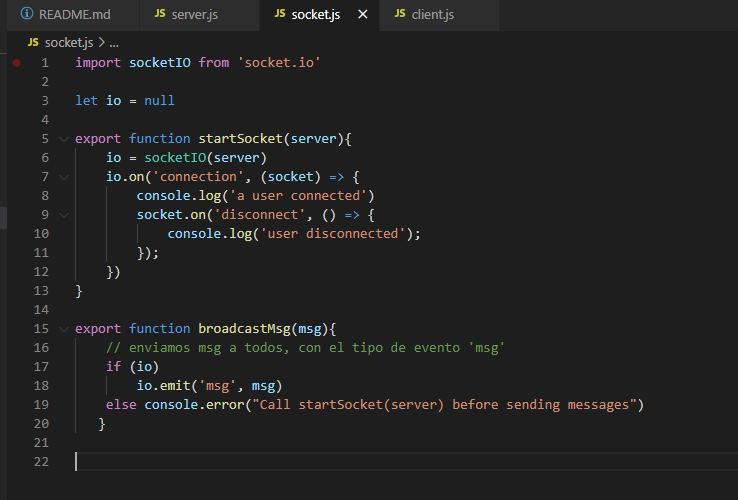


**2.SERVIDOR DE MENSAJERÍA, EN MODO ECO**

Creamos el fichero server.js como se pide:

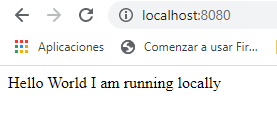


Para arrancar también el servidor de WebSocket, creamos la función *startSocket* en el fichero socket.js. Además, creamos la función *broadcastMsg* para que envíe un mensaje a todos los usuarios conectados a través de websocket, como se aprecia en la captura:



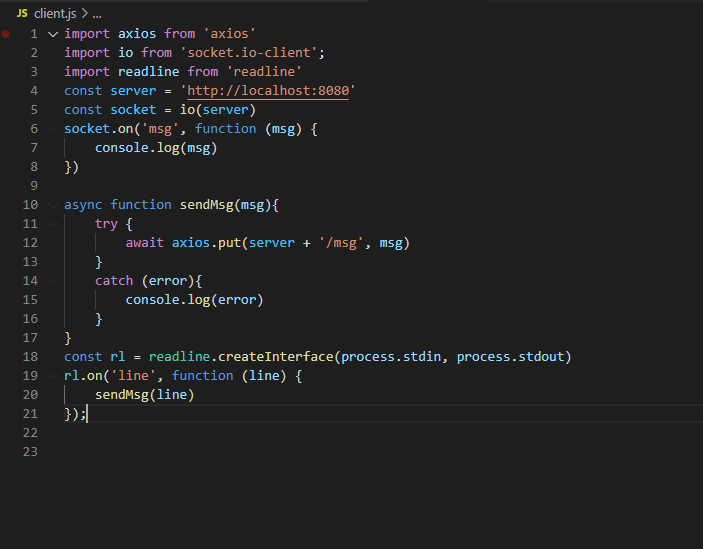
Para comprobar que lo que hemos hecho funciona, vamos a conectarnos a localhost:8080 desde un navegador, tras iniciar el servidor con ‘*node ./server.js’,* y si el mensaje ‘*Hello World I am running locally’* aparece, significará que todos los pasos están correctos hasta el momento.

Esto se puede apreciar en la siguiente imagen:



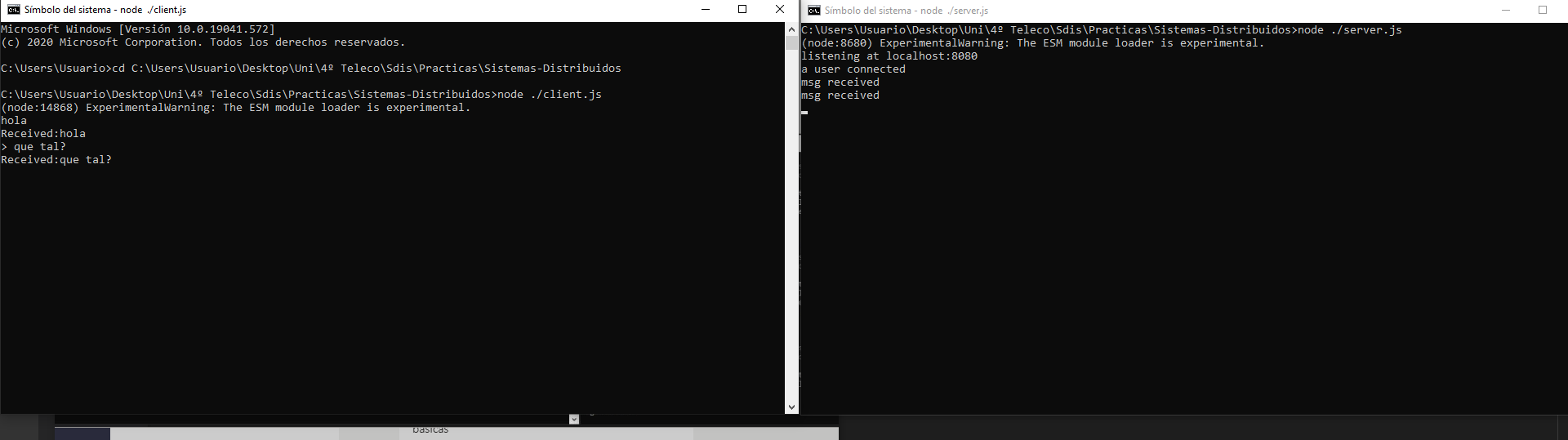
Después, hemos tenido que crear el fichero client.js para poder llamar al método *put*, el cual hemos añadido anteriormente para que la función del api *RESTful* que recibe mensajes retransmita con *broadcastMsg* lo que le llegue mediante este *put /msg* ya mencionado.

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



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

(Ampliar para ver en más detalle)

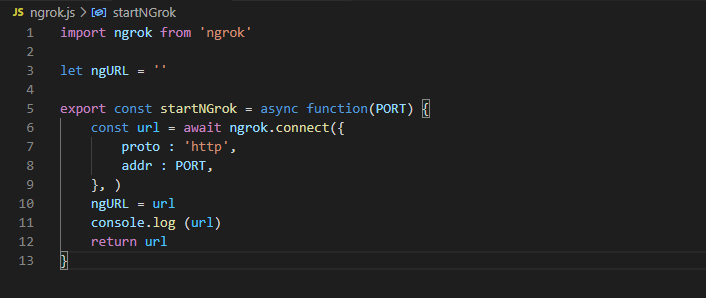


**3.COMPLETANDO LA RED P2P**

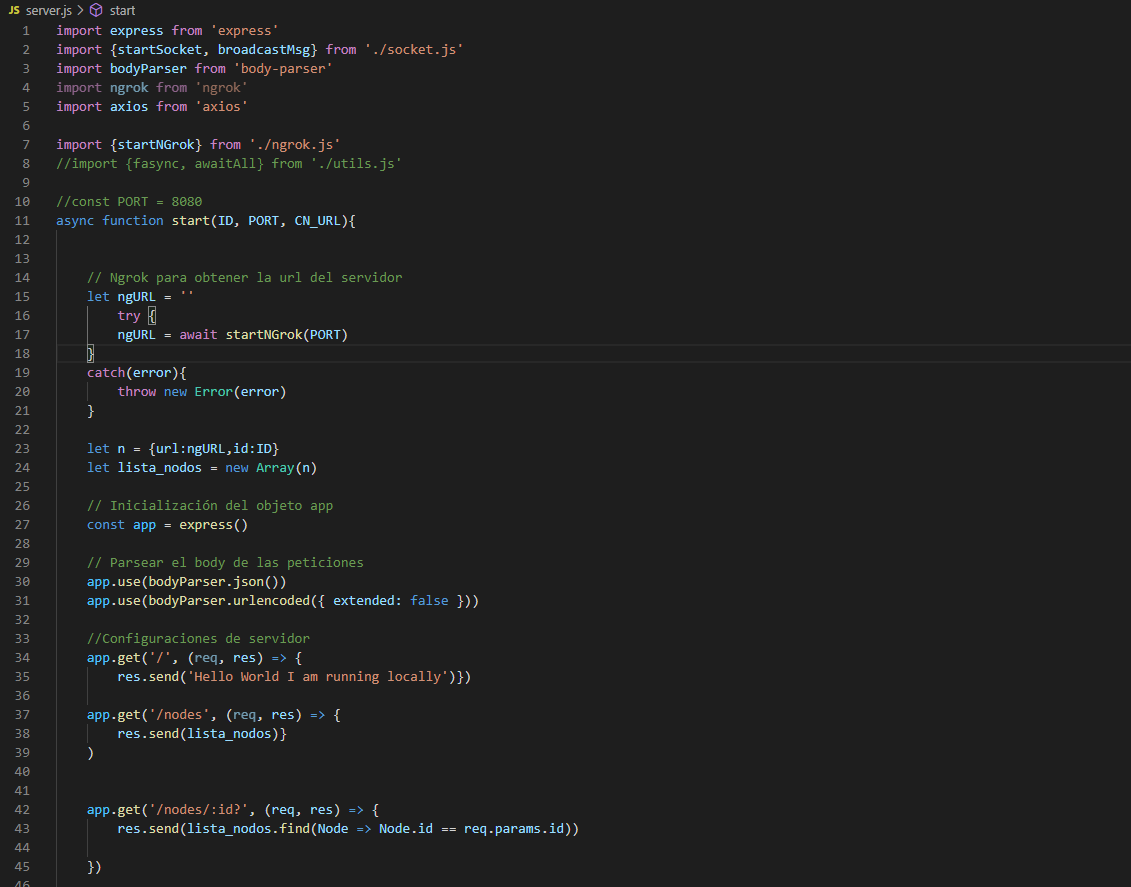
En esta parte de la práctica, vamos a hacer una serie de modificaciones para que los nodos puedan, a parte de recibir mensajes desde localhost, recibirlos desde otras máquinas.

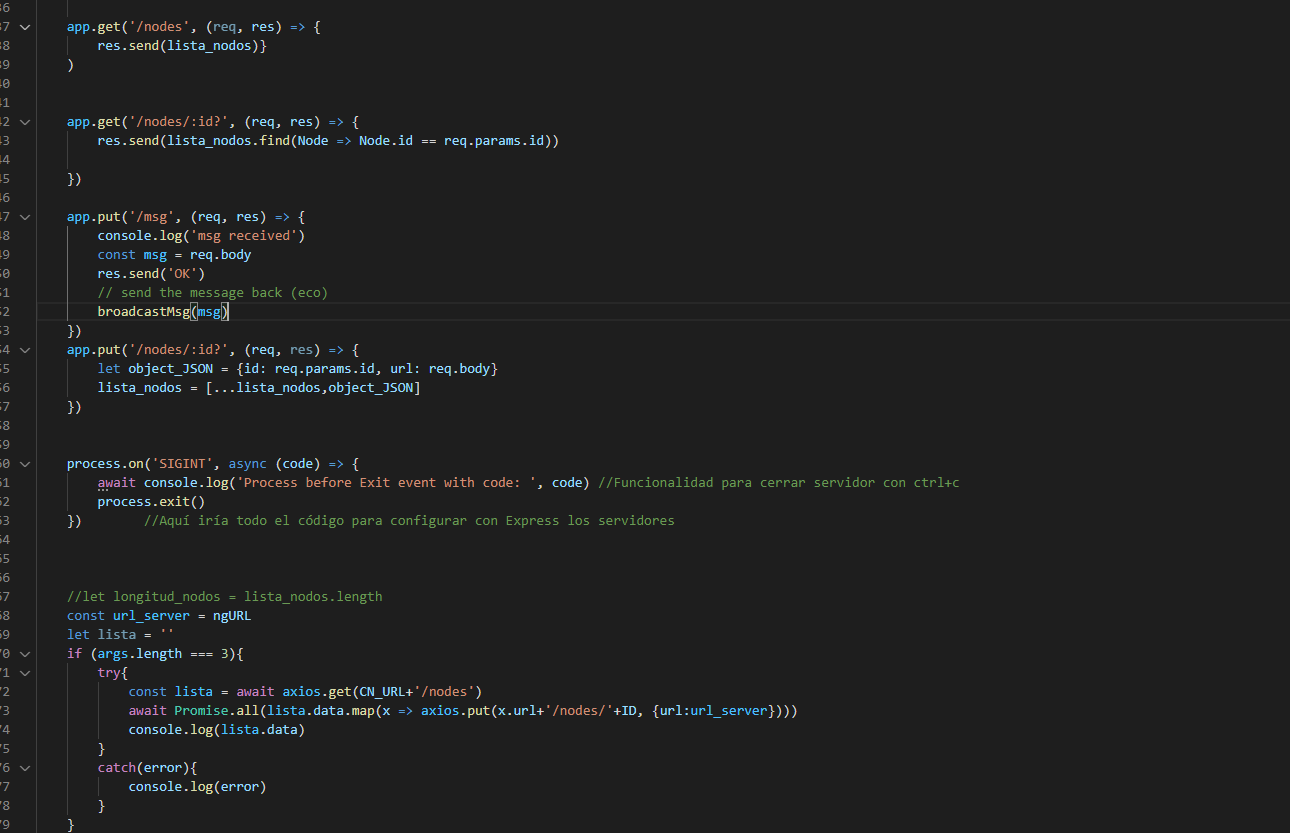
Para esto, vamos a usar ngrok, que es un servicio de internet que nos sirve para crear un túnel cifrado que nos permita conectar los nodos desde una IP pública, ya que a menudo, cuando estamos creando una red P2P nos encontramos con problemas con los cortafuegos y con el uso de IPs privadas, que nos 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.

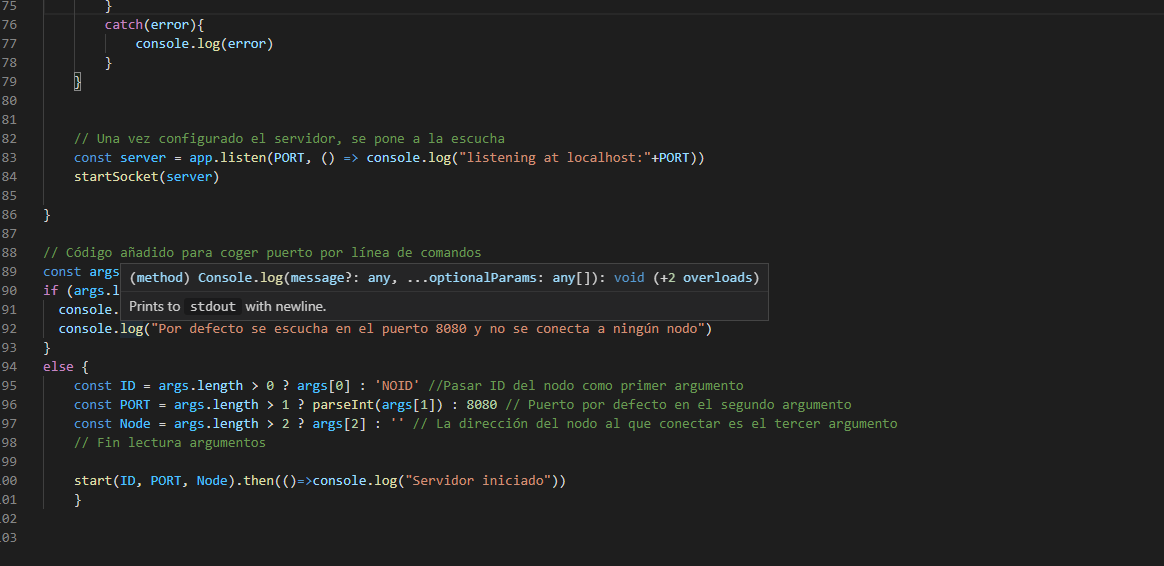
Para implementar todo esto, creamos el fichero ngrok.js, como se ve a continuación:



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):

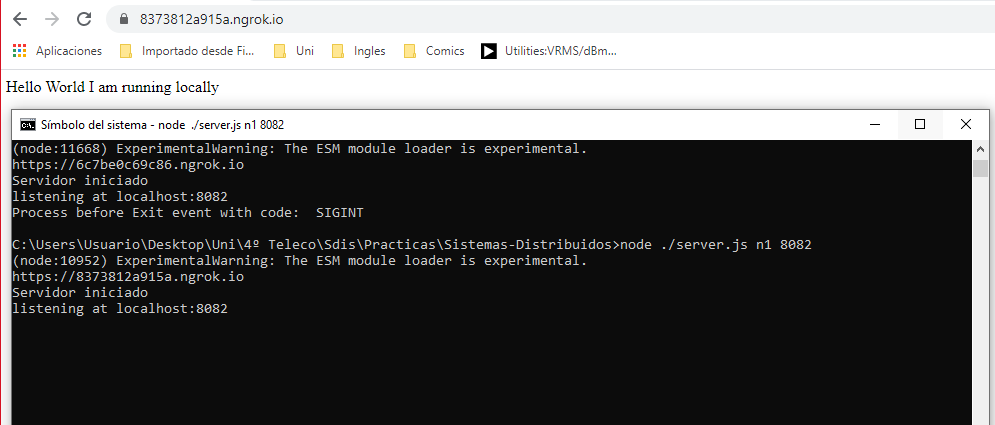






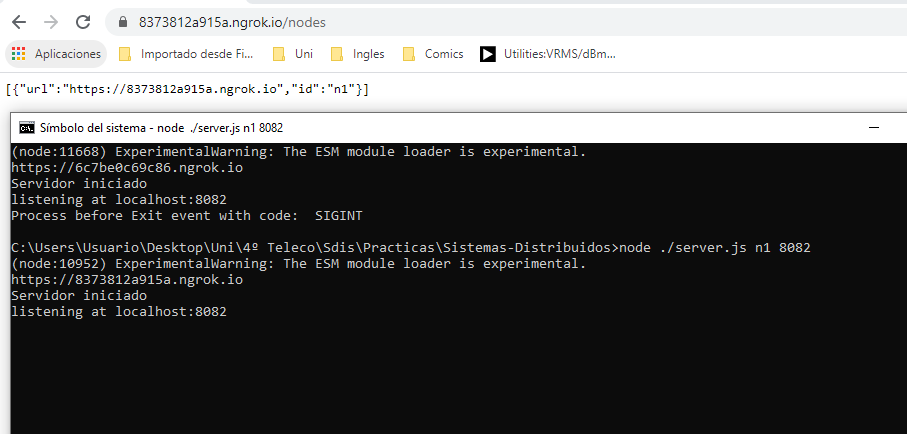
Para ejecutarlo, vamos a la terminal tecleando “*cmd*”. Después, vamos a dirigirnos a la carpeta en la que estemos realizando nuestro proyecto (con el comando “cd”), y posteriormente escribimos “*node ./server.js n1 8082*” para ejecutar el server con el nodo *n1* en el puerto *8082* (podríamos elegir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

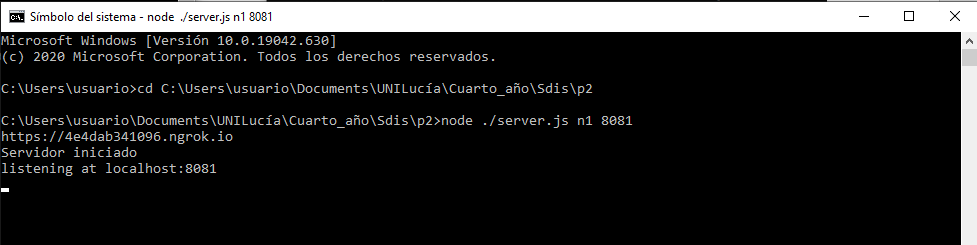
Para cumplir con lo siguiente que se nos pide, si ahora escribimos */nodes* a continuación de esta URL, veremos que nos aparece el siguiente mensaje con los datos de la URL y el ID de nuestro nodo.



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.

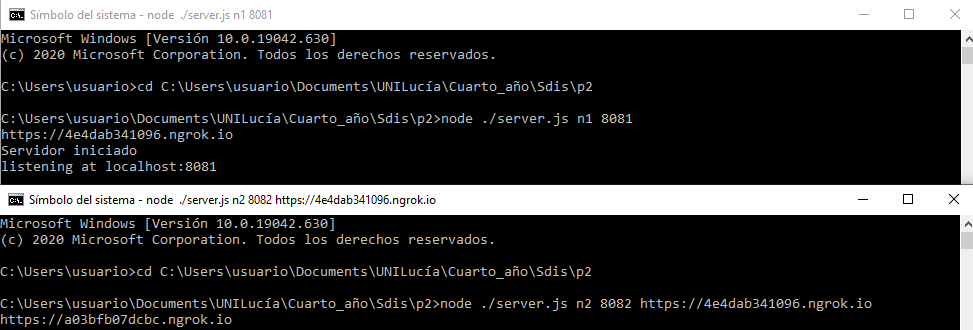
Para hacer esto, procedemos de manera similar a como lo hemos hecho antes:

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

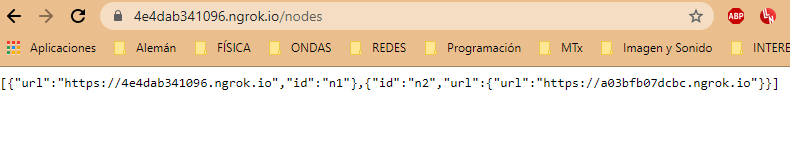


Copiamos la URL dada en el browser y nos aparecerá de nuevo el mensaje de “Hello World I am running locally”. Después, hacemos un /nodes y nos aparece el mensaje superior con la URL y el id del primer nodo. Hasta aquí, todo es igual al paso anterior.

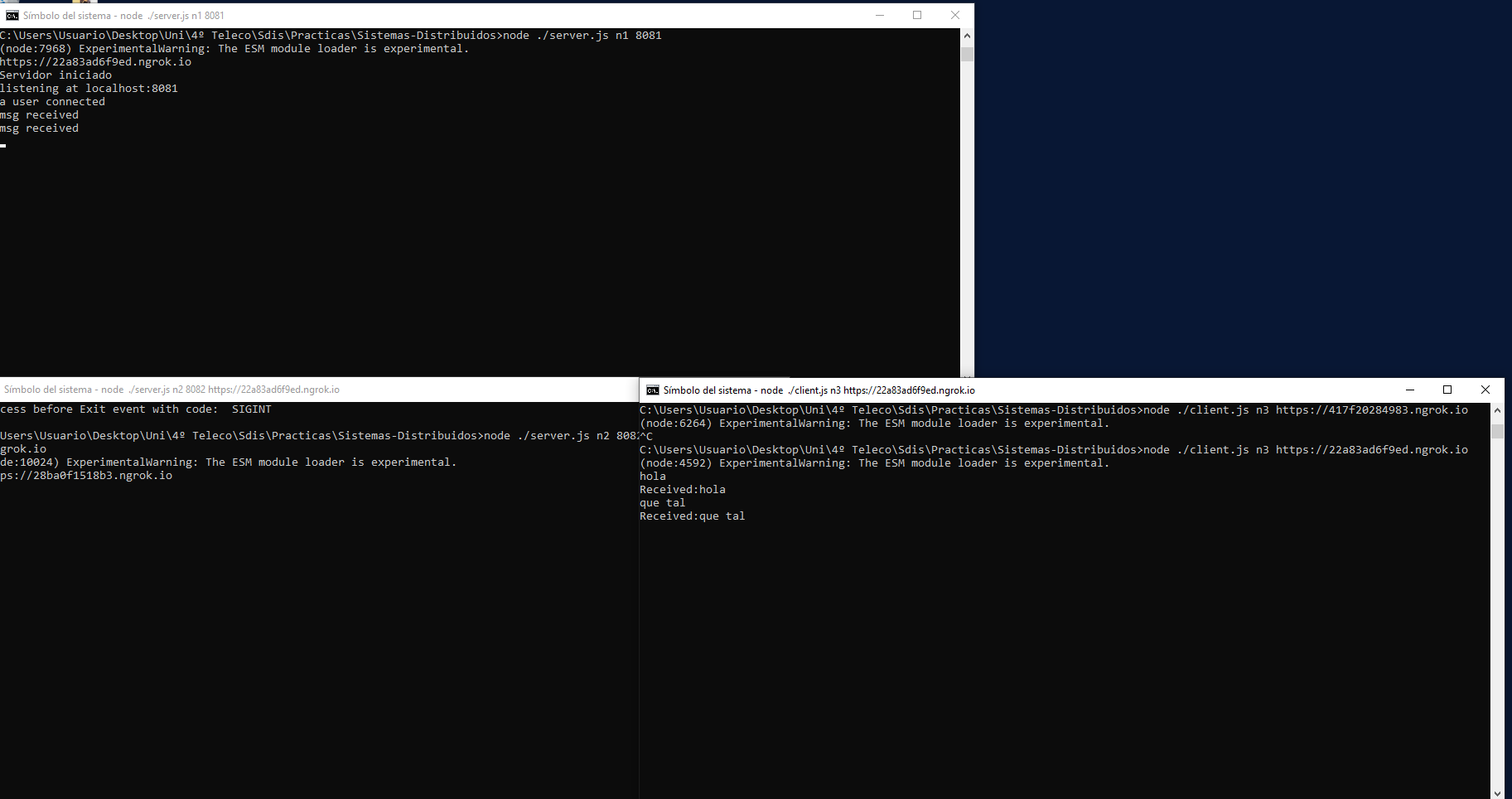
El cambio surge ahora, cuando abrimos otra terminal y ejecutamos el server de la siguiente forma: *“node ./server.js n2 8082 https://4e4dab341096.ngrok.io*” es decir, añadiendo como argumento la URL dada al lanzar el primer server para poder conectarnos con el nuevo nodo a la dirección del otro nodo.



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:



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:

