Práctica 2. Servicio de mensajería P2P

La arquitectura Peer-to-Peer (P2P) se utiliza frecuentemente en servicios distribuidos cuando queremos un sistema escalable donde el número de nodos ni sus direcciones no es conocido a priori.

Podemos usar esta arquitectura para crear un servicio, de forma que cualquier nodo pueda responder a las consultas que se realicen, tenga este o no la información necesaria para ello. Cada nodo además permite que otros nodos descubran el resto de la red, o se añadan a la misma.

En esta práctica crearemos un servicio de mensajería instantánea y dos aplicaciones en Javascript con node.js. Una aplicación será el nodo P2P, y la otra un cliente que se conecta a un nodo cualquiera de la red P2P para publicar mensajes.

Un problema importante a la hora de crear una red P2P son los cortafuegos, y el uso de IPs privadas, que obligan a conectar a internet usando NAT, etc. Este problema lo solucionaremos usando un servicio de internet para crear un túnel cifrado que nos permita conectar a nuestros nodos desde una IP pública. Uno de los más sencillos de utilizar, y que no requiere registrarnos, es ngrok. De forma gratuita podremos crear con ngrok hasta 4 túneles, y recibir hasta 40 conexiones por minuto.

API RESTful

A continuación se describe el API RESTFul, indicando para cada operación el método http, la ruta, y la forma que tiene el argumento en formato JSON que se pasa al método, o el formato de la respuesta JSON (después de =>).

Este API sólo se utilizará para la comunicación entre los nodos de la red P2P. Nuestro sistema gestionará dos recursos, nodos, y mensajes. Para simplificar la práctica no implementaremos mecanismos de seguridad, replicación, control de fallos, etc., los cuales son imprescindibles en sistemas que están en producción, pero se escapan del ámbito de esta práctica.

Gestión de nodos. Cada nodo tiene un “id” único, y una dirección (url) donde escucha peticiones. Para garantizar que los “id” son únicos, cada grupo de prácticas usará un código alfanumérico que comience por la letra del grupo de prácticas asignado en moodle, y evitará crear más de un nodo con el mismo id simultáneamente.

- Listar todos los nodos conocidos: get /nodes => [{id: ‘id1’, url: ‘url1’}, {id: ‘id2’, url: ‘url2’}, …, {id: ‘idn’, url: ‘urln’}]
    - Cada nodo guardará una lista en memoria de los nodos que conoce. Esa lista al comienzo solo contendrá al nodo actual, y se irá actualizando.

    - Al arrancar un nodo, si queremos que se una a una red existente, este tendrá URL de algún nodo de la red, al que le pedirá la lista del resto de nodos, para añadirse como nodo a todos ellos (con put /nodes/ID, ver más abajo).
- Obtener la URL de un nodo: get /nodes/ID => { url: 'https://….' }
    - Este método permite conocer la URL de un nodo, dado el ID, sin tener que preguntar la lista completa de nodos e IDs.
- Añadir un nodo: put /nodes/ID {url: 'https://…..'}
    - Este método lo llamarán los nodos al iniciarse, para pasar al resto de nodos de la red su propia dirección pública (url), e ID.
- Eliminar un nodo. delete /nodes/ID {url: 'https://…..'}
    - Este método lo llamarán los nodos al desconectarse de la red, de forma que notifiquen al resto de nodos que el nodo actual dejará de prestar servicio. Los nodos que reciban este mensaje solo eliminarán el nodo si coincide el ID y su URL.

Los usuarios finales no son nodos de la red, sino que se conectarán a un nodo para enviar mensajes a través de dicho nodo.

Puesto que queremos que los mensajes sean instantáneos, necesitamos un mecanismo de comunicación bidireccional entre usuarios y nodos de la red. Para ello utilizaremos WebSockets, y en particular la implementación que hace la librería socket.io. Este mecanismo evitará que los usuarios necesiten abrir un puerto para escuchar en una IP pública, o tengan que usar túneles, como en el caso de los nodos P2P.

Para simplificar, todos los usuarios conectados a un mismo nodo recibirán de forma instantánea los mensajes que lleguen a ese nodo, y no distinguiremos dentro del nodo a los distintos usuarios conectados.

Gestión de Mensajes. Los usuarios envían y reciben mensajes a través de websockets, pasando o recibiendo un objeto JSON con los siguientes campos { from: 'ID del nodo remitente', to: 'ID del nodo destino', msg: 'cuerpo del mensaje'}

Cuando un nodo recibe un mensaje de uno de sus usuarios conectados, este lo retransmite al nodo destino, que a su vez se lo pasará a sus usuarios conectados. El API para enviar mensajes solo tiene por lo tanto un método

- put /msg { from: 'ID del nodo remitente', to: 'ID del nodo destino', msg: 'cuerpo del mensaje'}
    - Nota: El objeto JSON enviado es el mismo que envió el usuario a través de websockets.

Paso 1. Preparación del proyecto e instalación de las librerías necesarias.

Primero prepara el proyecto tal como hiciste en la práctica 0. Asegurate de haber creado el fichero package.json (con npm init) y de añadir "type": "module", para indicar a node.js que utilizaremos módulos ES6, de forma que podamos usar import/export.

En esta práctica necesitarás instalar, con npm install, las siguientes librerías:

- ngrok: Para crear los túneles https y obtener una URL accesible de forma pública en internet.
- express y **body-parser**: Para crear el servicio RESTful
- socket.io y socket.io-client: Para las conexiones entre nodos y usuarios con websockets
- axios: Esta librería permite conectar a los servicios RESTful de una forma más sencilla

Paso 2. Servidor de mensajería, en modo ECO

Crea el fichero server.js con el siguiente código

```
import express from 'express'
import {startSocket} from './socket.js'
import bodyParser from 'body-parser'

const PORT = 8080
const app = express()
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: false }))

app.get('/', (req, res) => {
    res.send('Hello World I am running locally')
})
const server = app.listen(PORT, () => console.log("listening at localhost:"+PORT))
startSocket(server)
```

Este código iniciará express.js y conectará la operación get con la ruta '/' con al función lambda que muestra nuestro mensaje de hello world.

Para arrancar también el servidor de WebSocket, crearemos la función startSocket en el fichero socket.js, con el siguiente contenido.

```
import socketIO from 'socket.io'

let io = null

export function startSocket(server){
    io = socketIO(server)
    io.on('connection', (socket) => {
        console.log('a user connected')
        socket.on('disconnect', () => {
```

```
                console.log('user disconnected');
            });
        })
    }

    export function broadcastMsg(msg){
        // enviamos msg a todos, con el tipo de evento 'msg'
        if (io)
            io.emit('msg', msg)
         else console.error("Call startSocket(server) before sending
messages")
    }
```

También hemos creado la función broadcastMsg, que enviará un mensaje a todos los usuarios conectados a través de websocket.

Para usar esta función desde cualquier módulo, simplemente importala de forma similar a como se importa startSocket:

import {broadcastMsg} from ‘./socket.js’

Puedes probar que funciona el mensaje de HelloWord, conectando a localhost:8080 desde un navegador, tras iniciar el servidor con ‘node ./server.js’

Para que el servidor de ECO funcione, debemos añadir la función del api RESTful que recibe mensajes. Por ahora puedes hacer que esa función simplemente retransmita con broadcatMsg lo que llegue mediante put /msg

Por ejemplo:

```
    app.put('/msg', (req, res) => {
        console.log('msg received')
        const msg = req.body
        res.send('OK')
        // send the message back (eco)
        broadcastMsg(msg)
    })
```

Para llamar a este método put y probar nuestro servidor de ECO, podemos crear un cliente sencillo. Crea el fichero client.js con el siguiente código:

```
    import axios from 'axios'
    import io from 'socket.io-client'
    import readline from 'readline'

    const server = 'http://localhost:8080'
    const socket = io(server)
    socket.on('msg', function (msg.msg) {
```

```
        console.log(msg)
    })

    async function sendMsg(msg){
        try {
            await axios.put(server + '/msg', {msg})

        }
        catch (error){
            console.log(error)
        }
    }

    const rl = readline.createInterface(process.stdin,
process.stdout)

    rl.on('line', function (line) {
        sendMsg(line)
    });
```

La función sendMsg enviará las líneas que escribamos por entrada estándar, ejecutando a través de axios el método put del servidor. Por defecto conectaremos al nodo que hemos ejecutado en local.

Paso 3. Completando la red P2P

Para que los nodos puedan recibir mensajes desde otras máquinas, y no solo desde localhost, necesitamos obtener una dirección pública. Para ello utilizaremos el servicio gratuito de ngrok.

Puedes hacerlo creando el fichero ngrok.js, con el siguiente código

```
    import ngrok from 'ngrok'
    export const startNGrok = async function(PORT) {
       return ngrok.connect({
           proto : 'http',
           addr : PORT,
       }, )
    }
```

Podremos importar esta función desde cualquier módulo, y ejecutarla para obtener la URL pública, por ejemplo:

```
 import {startNGrok} from ‘./ngrok.js’
// desde una función asíncrona, obtenemos la URL
const url = await startNGrok(PORT)
```

Cuando necesitemos llamar de un nodo a otro, usaremos su dirección pública, y podremos llamar al API RESTful con Axios, de forma similar a lo que hicimos en el ejemplo de client.js

Duración y entrega de la práctica

La duración de esta práctica es de 3 semanas.

Crea un documento PDF incluyendo pantallazos con distintos ejemplos de ejecución, de los pasos principales para ejecutar y probar la práctica, comentando los problemas encontrando y cómo se han solucionado.

La entrega se realizará en un único fichero zip, que incluya el pdf de la memoria, y todos los archivos necesarios para ejecutar la práctica.