Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Ingeniería inversa de un chat Libp2p.

Daniel Sanchidrián Jimeno

Tutor: Juan José Álvarez Sánchez

Curso: 2023-2024

ÍNDICE DE CONTENIDOS

1. DESCRIPCIÓN GENERAL DEL PROYECTO.

1.1. OBJETIVOS Y CARACTERÍSTICAS DE LA APLICACIÓN.

El objetivo principal que queremos alcanzar con este proyecto es el desarrollo de una ingeniería inversa de una aplicación de chat descentralizado utilizando *libp2p* como su herramienta principal para explicar su funcionamiento. La ingeniería inversa consiste en el proceso de desmontar, analizar y comprender un sistema para determinar su diseño, características y funcionalidades.

Lo importante de este chat, y por ende su elección, es su privacidad entre los usuarios que se conecten y puedan tener una comunicación eficaz y segura. Para ello, cada uno de los usuarios dispondrá de una clave pública y privada, que serán únicas de cada uno de estos.

Este chat ha sido creado de una forma en la que cualquier usuario, conociendo su funcionamiento (en este aspecto entraríamos nosotros con la ingeniería inversa), pueda ser capaz de utilizarlo de una forma fácil y segura. De hecho, para utilizar el chat no hará falta registrarse, ya que, la información de cada uno de los nodos está guardada dentro de los nodos ejecutables de la aplicación.

Los principales objetivos que queremos obtener con esta aplicación son: una conexión descentralizada que permita a los usuarios conectarse directamente entre sí sin depender de un servidor centralizado, confidencialidad, integridad, autenticación robustez y facilidad de uso, que hagan que esta aplicación sea la opción principal para los usuarios. Además, gracias al empleo de *libp2p*, gozará de mayor interoperabilidad y se podrá conectar a otros nodos que utilicen tecnologías o plataformas diferentes.

Para el funcionamiento de esta aplicación serán necesarios dos elementos: un sistema Windows para poder acceder a la consola de comandos (*CMD*), y una librería *libp2p* combinada con cualquier tipo de código (*java, javascript, typescript*…)

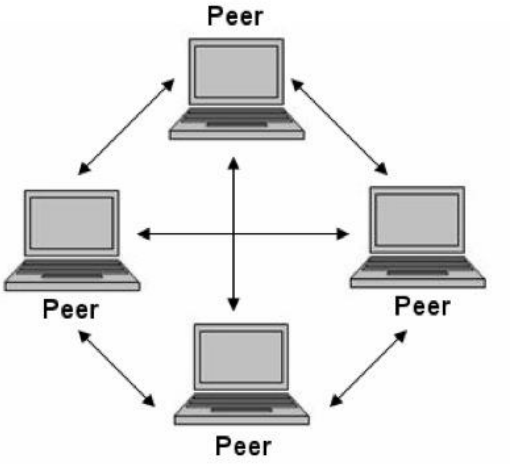

Ilustración 1. Esquema red p2p.

Libp2p es un sistema modular de protocolos, especificaciones y bibliotecas que permiten la realización de aplicaciones de red del tipo *peer-to-peer* (*p2p*). Su objetivo será integrar ser un conjunto de herramientas modular y de uso general para cualquier aplicación del tipo *peer-to-peer* (*Libp2p Documentation Portal*, s.f.)

Ilustración 2. Componentes libp2p.

Libp2p es una biblioteca modular y extensible que está diseñada para poder facilitar la comunicación *peer-to-peer* en distintos entornos y aplicaciones. Según el Portal de Documentación de *Libp2p*, las características principales que lo definen son las siguientes:

- **Modularidad:** Con ello nos referimos a que los desarrolladores que trabajan con *libp2p* pueden combinar diferentes componentes para poder satisfacer las necesidades que necesite la aplicación. Esto provoca que la personalización del conjunto de redes pueda adaptarse a cualquier requisito especifico de cualquier aplicación *p2p*.

- **Flexibilidad en la programación en la capa de transportede la pila de protocolos *tcp/ip*:** *Libp2p* proporciona un conjunto de especificaciones que pueden adaptarse para admitir varios protocolos de transporte, lo que provoca que las aplicaciones de *libp2p* operen en diversos entornos de tiempo de ejecución y redes.
- **Versatilidad:** Esta biblioteca modular ofrece una gran variedad de mecanismos de descubrimiento, patrones de almacenamiento y recuperación de datos. También está implementado en muchos lenguajes de programación, lo que facilita a los desarrolladores una gran flexibilidad a la hora de construir aplicaciones *p2p*.
- **Seguridad:** Algunas características de seguridad pueden ser la verificación de identidad de pares mediante criptografía de clave publica o la comunicación cifrada entre pares utilizando algoritmos criptográficos modernos.
- **Robustez:** Gracias a sus características y elecciones de diseño, *Libp2p* puede funcionar de manera efectiva y eficiente en una amplia gama de entornos y recuperarse rápidamente de interrupciones o fallos.
- **Resiliencia:** Las redes *p2p* suelen ser más resistentes, ya que no existe un único punto de fallo como en las tradicionales. *Libp2p* incluye características como pueden ser el descubrimiento de pares y enrutamiento de contenido que ayudan a garantizar que la red permanezca disponible y accesible incluso si algunos pares están desconectados.
- **Eficiencia:** En cuanto a la utilización de recursos, las redes *p2p* distribuyen los datos entre múltiples pares en lugar de almacenarse en un servidor central. Al incluir varios patrones de almacenamiento y recuperación, los desarrolladores pueden distribuir datos de manera eficiente en la red, lo que permite almacenar y recuperar datos de manera rentable y escalable.
- **Superación de barreras *Network Address Translation (NAT)*:** *Libp2p* permite la comunicación *P2P* entre pares, incluso cuando están detrás de dispositivos *NAT* o *firewalls*. Esto provoca que se pueda mantener la conectividad de la red y asegurar que siga siendo accesible a pesar de la presencia de la *NAT* o de los *firewalls*.
- **Distribución y difusión de mensajes:** Este sistema tiene un patrón de publicación/suscripción (*pubsub*) que permite a un remitente enviar el mismo mensaje a varios destinatarios sin que este tenga que conocer quiénes son los

usuarios receptores. *Libp2p* implementa *pubsub* a través de protocolos como puede ser *gossipsub*, proporcionando a los desarrolladores un medio flexible y eficiente de intercambio de datos y mensajes dentro de las aplicaciones *P2P*.

- **Interoperabilidad:** *Libp2p* puede ser programado en distintos lenguajes y versiones, pero, gracias a la interoperabilidad, los distintos ecosistemas de lenguaje se pueden comunicar sin problemas.
- **Descentralización**: Las redes *p2p* son redes de naturaleza descentralizada, lo que les permite operar sin ninguna autoridad central. *Libp2p* está diseñado específicamente para facilitar la comunicación descentralizada entre pares, lo que provoca que se puedan construir aplicaciones *P2P* resistentes a la censura y más resilientes ante las interrupciones de red.

Por lo tanto, observamos que *libp2p* es una herramienta flexible, a la vez que segura, para su implementación en los sistemas de red *peer-to-peer*. Todas estas características presentadas son más beneficiosas que desfavorables, por lo que este proyecto se ve favorecido mediante el uso de esta librería modular.

1.2. MOTIVACIÓN.

Últimamente, las conversaciones que tenemos a través de chats que han tenido problemas de seguridad, atacando la intimidad de las personas y viendo sus datos filtrados han generado mucho de qué hablar. Aunque estas aplicaciones de mensajería han revolucionado las comunicaciones y nos han facilitado la vida a muchos niveles, debemos ser conscientes de los riesgos asociados a nuestra privacidad que conllevan, como el robo de datos personales, de contacto, conversaciones privadas o información de contenido sensible.

Estas aplicaciones de mensajería están continuamente expuestas a amenazas y ataques cibernéticos, por lo que, según el Servicio de Salud de las Islas Baleares (2024), Por ello debemos utilizar opciones de mensajería que incluyan el cifrado de extremo a extremo para mejorar la seguridad de nuestras comunicaciones.El objetivo de este trabajo es aplicar la ingeniería inversa para comprender el funcionamiento de cada una de las partes componentes del chat. Esto facilitaría su conocimiento desde su bajo nivel (funcionamiento de la interfaz del chat) hasta el más alto (conocimiento de la red *p2p*, protocolo de conexión, etc.).

Procederemos entonces a buscar una aplicación en fase beta que consista en un chat cifrado, para que dicha conexión entre los dos usuarios esté bajo el control del usuario. Este chat consiste básicamente en dos consolas de Windows (*CMD*), en la que uno inicia el *dialer* y otro el *listener* para poder así establecer una conexión *TCP* entre ellos, con una clave pública y otra privada, que provoca que la conexión sea mutua entre los dos pares.

Así, para evitar ser espiados, utilizaremos *libp2p* para establecer esta conexión descentralizada, con el objetivo de establecer la conexión entre dos pares sin tener que depender de las grandes multinacionales que controlan el mercado, ya sean Facebook, Telegram, o cualquier otra. Esto provoca que la conversación privada que tengan dos usuarios utilizando esta aplicación sea segura, ya que es controlada por los propios usuarios y no por entes externos.

Hemos escogido *Libp2p* porque es una biblioteca que contiene todo lo necesario para poder crear redes *p2p* de una forma sencilla, y esta misma se encarga de crear todas las necesidades de la red como puede ser el enrutamiento, la detección de nodos y la transmisión de datos, haciendo que sea la herramienta idónea para nuestro proyecto.

2. CUESTIONES METODOLÓGICAS.

La metodología utilizada en nuestro caso es *SCRUM*, ya que es la que mejor se adapta al formato y tipo de trabajo.

SCRUM hace referencia a un marco de trabajo ágil a través del cual las personas abordan problemas complejos adaptativos a la vez que se entregan productos de forma creativa y eficiente con el máximo valor posible. *Scrum* es una metodología que se encarga de ayudar a los equipos a colaborar y a realizar un trabajo eficaz y de alto impacto. La metodología *SCRUM* proporciona valores, roles y pautas para ayudar al equipo a centrarse en la iteración y la mejora continua en proyectos complejos (Martins, 2024)

A continuación, procederemos a definir los elementos que actúan en la metodología SCRUM y que utilizaremos en dicho proyecto.

En el apartado personal, dentro de los **roles del equipo**, distinguimos entre:

- ***Product Owner***: Este rol hace referencia a la persona que va a definir los requisitos del chat *libp2p*. En este caso, somos nosotros mismos, ya que estamos realizando una ingeniería inversa de este proyecto.
- ***Scrum Master***: Este rol se encarga de hacer seguir a todo el equipo de desarrollo todos los principios y prácticas de SCRUM, como pueden ser las *dailies* o las retrospectivas del proyecto. De nuevo, ante la autogestión de este trabajo, somos nosotros mismos.
- **Equipo de desarrollo**: Es el responsable de implementar, diseñar y probar el chat. En este caso, el *product owner* y el *scrum master* son la misma persona.

Por un lado, los ***sprints*** son las reuniones del equipo propio, es decir, entre el alumno y tutor. En ellas se establecen las diferentes formas de trabajar y de afrontar los puntos a tratar dentro del proyecto. En nuestro contexto particular, los *sprints* se realizarán cada dos semanas aproximadamente, de forma tanto presencial como telemática (ante la imposibilidad de la primera opción).

Por otro lado, las **reuniones *SCRUM*** son las reuniones con los cargos superiores, es decir, con el *SCRUM master*. Podemos diferenciar entre:

- ***Sprint planning*:** Al comienzo de cada *sprint* las personas reunidas decidirán cómo y cuáles van a ser los aspectos que afrontar durante la duración del *sprint*.

- ***Daily*:** Son reuniones diarias de corta duración, las cuales vamos a utilizar para compartir el proceso que llevamos con las tareas, si estamos atascados o si hemos solventado todos los problemas de una forma eficaz.
- **Retrospectivas:** Al final de cada *sprint* todos los integrantes del proyecto se reunirán para darle el *feedback* correspondiente al *Scrum master* y comprobar que todas las tareas y las funcionalidades han sido implementadas. A su vez, también se planificará cómo vamos a afrontar el trabajo en el siguiente sprint y cuáles van a ser los próximos pasos a seguir y objetivos a conseguir.

Uno de los objetivos principales de *SCRUM* es fomentar la entrega continua de software funcional, verificando y comprobando al final de cada sprint que todas las especificaciones han sido implementadas.

3. PLANIFICACIÓN Y PRESUPUESTO

3.1. ESTIMACIÓN

La estimación de costes se realizará a través del método conocido como *COCOMO* (*COnstructive COst MOdel*), pues al margen de ser el modelo algorítmico de estimación de costo en proyectos de software más conocido. Según Boehm et al. (2000), como se citó en Garita-González y Liziano-Madriz (2018), este tipo de modelo es empleado para ayudar en la gestión de riesgos, la planificación y control del trabajo, y un mejor análisis de inversión del proyecto de software.

COCOMO es un modelo de formulación matemática con un fuerte componente de base empírica, principalmente utilizado para estimación de costos en los proyectos de software (Garita, 2014). El modelo *COCOMO* está orientado a la magnitud del producto final, está basado en estimaciones matemáticas y, por ende, mide el tamaño del proyecto y utiliza las líneas de código como unidad de medida.

Lo primero que debemos saber es qué modelo vamos a utilizar dentro de *COCOMO* para nuestro proyecto. Tenemos tres modelos los cuales vamos a ver a continuación:

Tabla 1: Valores constantes por modo de desarrollo (Boehm, 1984)

	Básico		**Intermedio**	
Modo	a_i	b_i	a_i	b_i
Orgánico	2.4	1.05	3.2	1.05
Semiencajado	3.0	1.12	3.0	1.12
Empotrado	3.6	1.2	2.8	1.2

En nuestro caso, al ser un proyecto pequeño de menos de 50 KLDC, utilizaremos el modelo orgánico para la realización de la estimación de este.

Tabla 2: Niveles de influencia para cada característica.

Comunicación de datos	4	Funciones distribuidas	3
Rendimiento	4	Gran carga de trabajo	4
Frecuencia de transacciones	3	Alta Concurrencia	4
Requisitos de manejo del usuario final	4	Facilidad de instalación	5
Procesos complejos	3	Utilización con otros sistemas	0

Facilidad de mantenimiento	3	Facilidad de operación	4
Instalación en múltiples lugares	0	Facilidad de cambio	4

La suma total de todos los puntos es: 45.

Calculamos el factor de ajuste (**FA**) con su fórmula:

FA = (Nivel de influencia * 0,01) + 0,65

FA = (45 * 0,01) + 0,65 = 1,1

Ahora vamos a calcular los puntos de función no ajustados:

- Entrada Externa: 5
- Salida Externa: 5
- Consulta Externa: 0
- Archivos Lógicos Internos: 2
- Archivos de Interfaz Externos: 0

De este modo, los puntos de función no ajustados (**PFNA**) serían:

PFNA = (5 x 3) + (5 x 4) + (0 x 0) + (2 x 15) + (0 x 0) = 65

Ahora vamos a obtener los puntos de función ajustados:

Puntos de función = FC * PFNA = 1,1 * 65 = 71,5

Como en PHP son 53 líneas de código (aproximadamente) por cada punto de función, entonces:

KLDC = Líneas/PF * PF = 53 * 71,5 = 3789,5 aproximadamente 4KLDC

Vamos a realizar COCOMO ahora:

Tabla 3: Valores constantes por modo de desarrollo (Boehm, 1984)

	Básico		**Intermedio**	
Modo	**A**	**B**	**C**	**D**
Orgánico	2.4	1.05	3.2	1.05
Semiencajado	3.0	1.12	3.0	1.12
Empotrado	3.6	1.2	2.8	1.2

Vamos a calcular el Esfuerzo nominal (**PM**) y el Factor de Ajuste de Esfuerzo (**FAE**):

$$\mathbf{PM} = A*(KLDC)^{B}$$

$$\mathbf{PM} = 2.4 * (4)^{1,05} = 10,2890 \sim 10 \text{ personas/mes}$$

$$\mathbf{FAE}=1,15*1,00*0,85*1,11*1,00*1,00*1,07*0,86*0,82*0,70*1,00*0,95*1,00*0,91*1,08 = 0,53508480$$

Ahora vamos a realizar el cálculo del esfuerzo:

$$\mathbf{Esfuerzo} = PM * FAE = 10 * 0,53508480 = 5,35 \text{ persona/mes}$$

A continuación, vamos a calcular cuánto costaría el proyecto y su duración estimada:

$$\mathbf{Coste} = 5,35 * 1200(€/mes) = 6420€$$

$$\mathbf{Tiempo} = 5,35/2 = 2,675 \text{ meses} \sim 3 \text{ meses}$$

Por lo que, con dos personas, durante 3 meses, trabajando el proyecto saldría adelante. Teniendo en cuenta que las personas con las que trabajamos son individuos con experiencia, todo el proyecto se podría cumplir en el plazo de tiempo estimado.

3.2. PRESUPUESTO

En este apartado vamos a realizar un estudio económico del proyecto, tanto de hardware, como de software, y el gasto del personal contratado.

Según una estimación realizada, este proyecto ha sido creado en 300 horas de trabajo, es decir, un total de 39 días de trabajo a 8 horas diarias cada día (aunque la jornada varíe de forma flexible, adaptándose a los horarios de oficina de la empresa). A continuación, vamos a ver una tabla con datos de horas empleadas para cada una de las tareas dentro del proyecto. Como utilizamos la metodología SCRUM, vamos a dividir las tareas en *sprints*.

Tabla 4: Distribución de horas de cada sprint.

Tarea	Duración (horas)
Sprint 1- Análisis del proyecto	60
Sprint 2 -Diseño	60
Sprint 3 – Código fuente	100
Sprint 4 – Pruebas de validación	80

Cada uno de los *sprints* han necesitado un reparto de horas distintos, por lo que se ha aplicado un reparto equivalente a la carga de trabajo de cada una de las partes especificadas.

3.2.1. Gastos Hardware

En este apartado analizaremos el coste del hardware empleado en el trabajo. En cuanto a los componentes empleados dentro del proyecto, distinguimos entre:

Tabla 5: Costes de recursos hardware.

Recursos	Coste	Unidades	Coste total	Coste/días
Ordenador portátil	799,99€	2	1599,98€	170,95€
Teclado	11,99€	2	23,98€	2,56€
Monitor	119,99€	2	239,98€	25,64€
Ratón	13,69€	2	27,38€	2,93€
				Total: 202,08

Para calcular el coste por días se ha procedido a calcular el coste total del producto por cada día en un año. Como la duración del proyecto son 300 horas, esto equivale a 39 días. la fórmula utilizada ha sido:

(Coste Total/365 días) * 39 días.

Los precios son los que hay ahora mismo en el mercado[1] con la suficiente potencia como para poder llevar a cabo el proyecto sin tener ningún tipo de problema.

En este caso estas son las descripciones de cada producto para tener un seguimiento de estos:

- Ordenador: HP 15-fd0045ns - Ordenador portátil de 15.6" Full HD (Intel Core i7-1355U, 16GB RAM, 512GB SSD, Intel Iris Xe Graphics, Windows 11 Home) Silver
- Teclado: Trust Taro Teclado - Disposición QWERTY Español
- Ratón: Logitech M185 Ratón Inalámbrico
- Monitor: HP V27ie G5- Monitor de 27" Full HD (1920 x 1080, 75Hz).

[1] Referencias tomadas del precio según la tienda online de la compañía Amazon (https://www.amazon.es/ref=nav_logo)

3.2.2. Gastos Software

En este caso específico, debemos tener en cuenta varias cosas. Por un lado, la librería *libp2p* es de uso público, por lo que no necesitamos licencia para trabajar con ella, al igual que *Visual Studio Code*, por lo que nos ahorraríamos el dinero de las licencias de ambos. Por otro lado, vamos a ver aquellos programas que sí necesitemos licencia para trabajar con ellos.

Tabla 6: Esquema de recursos software.

Recursos	**Coste**	**Unidades**	**Coste total**	**Coste/días**
Licencia Windows 11	119€	2	238€	25,43€
Licencia Microsoft Office	149€	2	298€	31,84€
Total: 57,27€				

3.2.3. Gastos de personal

En este apartado vamos a ver los gastos de personal empleados en nuestro proyecto.

Serán necesarios un programador y un líder de proyecto, que a su vez ocupará el cargo de *SCRUM master* del proyecto. Al utilizar SCRUM como metodología, vamos a dividir el personal en *Product Owner, Scrum Master* y equipo de desarrollo. Siendo un proyecto pequeño, solo vamos a necesitar a dos personas contratadas, y cada una de ellas tendrá el rol de *Scrum Master* y equipo de desarrollo, respectivamente. El *Product Owner,* en este contexto, será la propia empresa creadora del proyecto, ya que es un producto interno de esta.

Tabla 7. Recursos humanos del proyecto.

Recursos	**Salario/año**	**Salario/mes**	**Salario/día**	**Salario/hora**
Ingeniero informático	26.000 €	2.166,67€	108,33€	13,54€
Ingeniero Software	37.771€	3.147,58€	157,38€	19,67€

Estos datos han sido obtenidos a partir de los salarios que existen en la actualidad, basados en las ofertas que podemos encontrar en portales de trabajo como *LinkedIn*, *InfoJobs* o *Indeed* en el año 2024. Como los sueldos suelen mostrarse en términos de salario anual, lo hemos dividido en mes, día y hora.

Para calcular los costes de cada uno de los usuarios vamos a dividir el sueldo en las horas en las que va a trabajar cada uno de los trabajadores.

Tabla 8. Salario total según horas.

Recurso	Trabajo (Horas)	Coste (€)
Ingeniero informático	120	1624,8€
Ingeniero de Software	180	3540,6€
		Total:5165,4€

3.2.4. Presupuesto Total

En cuanto al resultado del presupuesto total del proyecto, teniendo en cuenta el total de los tres campos (Hardware, Software y recursos humanos), distinguimos:

Tabla 9. Gastos totales del proyecto.

Tipo de Gasto	Coste	Porcentaje (%)
Recurso Hardware	202,08 €	3,73%
Recurso Software	57,27€	1,2%
Recursos humanos	5165,4€	95,21%
		Total:5424,75€

En el gráfico a continuación distinguimos la relación entre los gastos del proyecto, y su porcentaje sobre el total.

Ilustración 3. Gráfico de relación de gastos totales.

4. DOCUMENTACIÓN TÉCNICA Y ANÁLISIS.

4.1. OBJETIVOS DEL SISTEMA.

En este apartado procederemos a explicar los objetivos principales que debe cumplir nuestro proyecto. Las variables de los requisitos de información han sido tomadas de las plantillas del Marco de Desarrollo de la Junta de Andalucía (MADEJA) (Durán y Bernárdez, 2002).

Tabla 10. Objetivo del sistema 1

OBJ - 1	**COMUNICACIÓN DESCENTRALIZADA**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Descripción	El sistema deberá conseguir una comunicación descentralizada entre los nodos de forma eficaz y segura.
Subobjetivos	OBJ - 1.1 Seguridad y privacidad OBJ - 1.2 Bajo consumo de recursos OBJ - 1.3 Escalabilidad OBJ – 1.4 Fácil Implementación OBJ – 1.5 Flexibilidad de conexión
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta

Tabla 11. Objetivo del sistema 1.1

OBJ - 1.1	**SEGURIDAD Y PRIVACIDAD**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Descripción	El sistema deberá garantizar la seguridad y la privacidad de las comunicaciones entre los nodos. En este caso utilizamos cifrado de extremo a extremo para proteger los mensajes entre nodos.
Importancia	Alta
Urgencia	Alta

Estado	Comprobado
Estabilidad	Alta

Tabla 12. Objetivo del sistema 1.2

OBJ - 1.2	**BAJO CONSUMO DE RECURSOS**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Descripción	El sistema deberá proporcionar de forma eficaz un bajo consumo de recursos dentro del sistema chat.
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta

Tabla 13. Objetivo del sistema 1.3

OBJ - 1.3	**ESCALABILIDAD**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Descripción	El sistema deberá ser escalable para poder provocar que el sistema chat pueda ser usado por varios nodos a la vez ampliando así el catálogo de usuarios conectados a la vez sin comprometer el rendimiento.
Importancia	Alta
Urgencia	Alta
Estado	Implementado con futura mejora
Estabilidad	Alta

Tabla 14. Objetivo del sistema 1.4

OBJ - 1.4	**FÁCIL IMPLEMENTACIÓN**

Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Descripción	El sistema deberá facilitar la implementación de funcionalidades de red dentro de nuestra aplicación.
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta

Tabla 15. Objetivo del sistema 1.5

OBJ - 1.5	**FLEXIBILIDAD DE CONEXIÓN**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Descripción	El sistema deberá permitir la comunicación entre las consolas incluso cuando la conectividad de estas a la red no sea constante, gracias a *libp2p* podremos conseguirlo.
Importancia	Media
Urgencia	Baja
Estado	Futura mejora
Estabilidad	Alta

4.2. REQUISITOS DE INFORMACIÓN.

Tabla 16. Requisitos de información 1

IRQ - 1	**INFORMACIÓN DE IP Y HOST**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Objetivos Asociados	
Requisitos Asociados	
Descripción	El sistema deberá recoger los datos de la *IP* y del *host* de

	cada nodo.
Datos específicos	Dirección IP *IP host* Protocolo de comunicación Clave Pública
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Máxima
Comentarios	

Tabla 17. Requisitos de información 2

IRQ - 2	**PUERTOS DE COMUNICACIÓN**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Objetivos Asociados	
Requisitos Asociados	
Descripción	El sistema deberá conocer los puertos de comunicación que están disponibles para evitar bloqueos de *firewalls* o restricciones de red
Datos específicos	Puertos de comunicación
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Máxima
Comentarios	

Tabla 18. Requisitos de información 3

IRQ - 3	**PROTOCOLOS DE TRANSPORTE**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno

Fuente	Daniel Sanchidrián Jimeno
Objetivos Asociados	
Requisitos Asociados	
Descripción	El sistema debe decidir el protocolo de transporte que se debe utilizar en para realizar la conexión entre los nodos
Datos específicos	Protocolo de transporte (*TCP*)
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Máxima
Comentarios	

Tabla 19. Requisitos de información 4

IRQ - 4	**IDENTIFICADORES DE NODOS**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Objetivos Asociados	
Requisitos Asociados	
Descripción	El sistema deberá conocer el identificador único de cada nodo (clave pública) para poder intercambiar esos identificadores para poder autenticarse los nodos entre sí.
Datos específicos	Clave Pública
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Máxima
Comentarios	

Tabla 20. Requisitos de información 5

IRQ - 5	**SEGURIDAD**
Versión	1.0

Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Objetivos Asociados	
Requisitos Asociados	
Descripción	El sistema deberá ser seguro para su uso utilizando un cifrado de clave privada y sea compatible con los dos nodos.
Datos específicos	Clave Privada
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Máxima
Comentarios	

Tabla 21. Requisitos de información 6

IRQ - 6	**LIBRERÍAS DE *LIBP2P***
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuente	Daniel Sanchidrián Jimeno
Objetivos Asociados	
Requisitos Asociados	
Descripción	El sistema deberá saber si las dos consolas tienen instaladas las librerías y herramientas necesarias para poder establecer la conexión entre los nodos
Datos específicos	Librerías *libp2p*
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Máxima
Comentarios	

4.3. RESTRICCIONES DE INFORMACIÓN

Tabla 22. Restricciones de información 1

CRQ - 1	PRIVACIDAD DE LOS MENSAJES
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	OBJ 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 04 – Identificadores de nodos IRQ 05 - Seguridad
Descripción	La información de los mensajes deberá ser privada entre los nodos y no puede ser accesible desde ningún nodo externo.
Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta
Comentarios	

Tabla 23. Restricciones de información 2

CRQ - 2	INTEGRIDAD DE LOS DATOS
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Obj 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 05 - Seguridad
Descripción	La información de los mensajes enviados no debe ser alterada o manipulada durante la transmisión.
Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta

Estado	Comprobado
Estabilidad	Alta
Comentarios	

Tabla 24. Restricciones de información 3

CRQ - 3	**AUTENTICACIÓN DE USUARIOS**
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	OBJ 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 05 - Seguridad
Descripción	La información que contiene cada usuario no debe ser alterada y solo aquellos que estén autorizados a participar en el chat lo podrán hacer
Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta
Comentarios	

Tabla 25. Restricciones de información 4

CRQ - 4	**CUMPLIMIENTO LEGAL**
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	OBJ 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 05 - Seguridad
Descripción	La información que se envía o sea almacenada debe cumplir con las leyes de privacidad y protección de datos.

Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta
Comentarios	

Tabla 26. Restricciones de información 5

CRQ - 5	**RETENCIÓN DE DATOS**
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	OBJ 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 05 - Seguridad
Descripción	La información de los mensajes enviados podrá ser mantenida durante un tiempo determinado antes de ser eliminados.
Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta
Comentarios	

Tabla 27. Restricciones de información 6

CRQ - 6	**PROTECCIÓN CONTRA ATAQUES**
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	OBJ 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 05 - Seguridad
Descripción	La información de los mensajes enviados

	debe estar protegida contra ataques malicioso como el *phishing*, por ejemplo.
Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta
Comentarios	

Tabla 28. Restricciones de información 7

CRQ - 7	**CONSENTIMIENTO DE USUARIOS**
Versión	1.0
Autores	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	OBJ 1.1 – Seguridad y privacidad
Requisitos asociados	IRQ 05 - Seguridad
Descripción	La información de los mensajes enviados debe de ser aceptada antes por el usuario consintiendo así su participación en el chat.
Datos específicos	Tipo
Importancia	Alta
Urgencia	Alta
Estado	Comprobado
Estabilidad	Alta
Comentarios	

4.4. REQUISITOS FUNCIONALES

Un requisito funcional es una forma de declarar cómo debe comportarse un sistema, es decir, se encarga de definir cómo debe trabajar dicho sistema para funcionar de la forma más correcta, satisfaciendo las necesidades del usuario de manera eficiente.

En lo referido al proyecto al que aplicamos ingeniería inversa, distinguimos entre distintos tipos de requisitos funcionales del sistema:

4.4.1. Definición de actores.

Un actor es una entidad externa al sistema que se encarga de interactuar con este. Los actores que hemos detectado en el sistema a estudiar son el usuario, la aplicación de chat, la Red *Libp2p* y la conexión segura.

Tabla 29. Definición primer actor

ACT - 1	**USUARIO**
Versión	1.0
Autores	Daniel Sanchidrian Jimeno
Fuentes	Daniel Sanchidrian Jimeno
Descripción	Este actor se encarga de interactuar con el sistema.
Comentarios	

Tabla 30. Definición segundo actor

ACT - 2	**APLICACIÓN CHAT**
Versión	1.0
Autores	Daniel Sanchidrian Jimeno
Fuentes	Daniel Sanchidrian Jimeno
Descripción	Este actor se encarga de establecer la conexión entre los dos pares.
Comentarios	

Tabla 31.Definición tercer actor

ACT - 3	**RED *LIBP2P***
Versión	1.0
Autores	Daniel Sanchidrian Jimeno
Fuentes	Daniel Sanchidrian Jimeno
Descripción	Este actor se encarga de establecer la infraestructura de red para facilitar la conexión *P2P*.
Comentarios	

Tabla 32. Definición cuarto actor

ACT - 4	CONEXIÓN SEGURA
Versión	1.0
Autores	Daniel Sanchidrian Jimeno
Fuentes	Daniel Sanchidrian Jimeno
Descripción	Este actor se encarga de que la conexión entre los pares sea segura.
Comentarios	

4.4.2. Diagrama de Interacción.

El Diagrama de Interacción describe cómo los actores interactúan entre sí. En este caso, contamos con los tres actores mencionados anteriormente: el usuario, la aplicación chat y la red *Libp2p*.

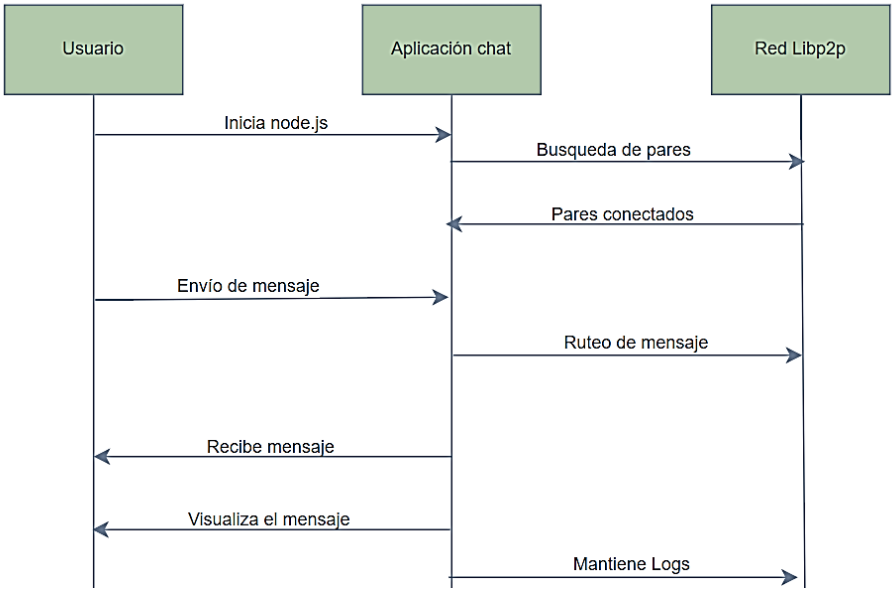

Ilustración 4. Diagrama de Interacción sobre el funcionamiento del chat

Mediante la ilustración anterior, podemos definir las relaciones y conceptos establecidos entre los actores para lograr una comunicación fluida y clara entre los dos *peers*.

- **Inicio node.js**: Los usuarios (cada uno en su consola) inician la aplicación utilizando el comando ***node dialer.js*** y ***node listener.js*** en cada uno de los casos para establecer una conexión inicial a la red *libp2p*.

- **Búsqueda de pares**: La aplicación busca, a través de la clave privada y pública en la red *libp2p*, el otro *peer* al que conectarse.
- **Pares conectados**: una vez encontrados los *peers*, el chat se encargará de establecer las conexiones seguras entre estos.
- **Envío de mensaje**: Uno de los *peers* envía un mensaje al otro a través de la aplicación, que el chat empaqueta y envía a través de la red *libp2p*.
- **Ruteo de mensaje:** La red libp2p se encarga de enrutar el mensaje al destinatario, en este caso el otro *peer*.
- **Recibe mensaje:** El *peer* destinatario recibe el mensaje y lo visualiza a través de la aplicación.
- **Mantiene logs**: La aplicación chat registra un historial de mensajes entre los *peers*.

4.4.3. Casos de uso

Podemos definir los casos de uso como la descripción textual de todas las maneras en las que los actores interactúan con el software. En este caso contamos con dos casos de uso diferentes: el caso de uso de usuario y el de la aplicación.

Caso de uso del usuario.

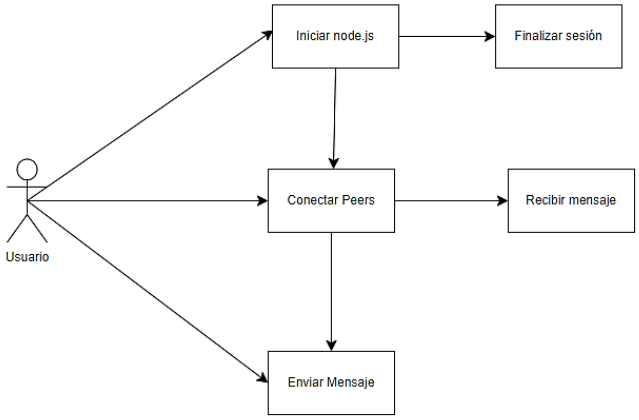

Ilustración 5. Esquema de interacción de usuario.

En esta ocasión, los actores son los dos usuarios, que pueden interactuar entre ellos mediante el sistema de chat libp2p. Para ello, deberán realizar ordenadamente las siguientes acciones:

1. **Iniciar node.js**. El usuario inicializa dentro de la consola de Windows el comando ***node listener.js*** o ***node dialer.js,*** según el usuario que sea, y establece la conexión con la red *libp2p*.
2. **Conectar *Peers*.** El sistema establece una conexión segura al *peer* (en este caso concreto, solo tenemos uno en la lista).
3. **Enviar mensaje.** El usuario escribe un mensaje, que la aplicación, a través de la red *libp2p,* envía al *peer* correspondiente.
4. **Recibir mensaje.** El usuario recibe un mensaje que ha sido enviado a través de la red *libp2p* desde otro *peer,* y lo muestra en la consola de este.
5. **Finalizar sesión.** El usuario decide terminar la sesión, ya sea por decisión propia, o por una caída de la red de internet. La aplicación cierra la conexión establecida con la red *libp2p*.

Tabla 33. Caso de uso 1

UC - 1	**INICIAR NODE.JS**	
Versión	1.0	
Autor	Daniel Sanchidrian Jimeno	
Fuente	Daniel Sanchidrian Jimeno	
Objetivos Asociados	OBJ – 1.4 – Fácil implementación.	
Requisitos Asociados	IRQ – 4 -Identificadores de nodos. CRQ -3 – Autentificación de usuarios. CRQ – 7 – Consentimiento de usuarios.	
Descripción	El actor iniciará la conexión con otro nodo a través de la red *libp2p*.	
Precondición	Ninguna	
Secuencia normal	Paso	Acción
	P1	El usuario accede a la consola de Windows.
	P2	El usuario se dirige a la ubicación en la que tiene los ficheros .js .
	P3	El usuario introduce el comando **node listener.js** o **node dialer.js.**
Postcondición	Usuario *logueado* dentro del chat.	
Comentarios		

Tabla 34.Caso de uso 2

<table>
<tr><td>UC - 2</td><td>CONECTAR PEERS</td></tr>
<tr><td>Versión</td><td>1.0</td></tr>
<tr><td>Autor</td><td>Daniel Sanchidrian Jimeno</td></tr>
<tr><td>Fuente</td><td>Daniel Sanchidrian Jimeno</td></tr>
<tr><td>Objetivos Asociados</td><td>OBJ- 1.5 - flexibilidad de conexión</td></tr>
<tr><td>Requisitos Asociados</td><td>IRQ -1 - información ip y host
IRQ -2 - Puertos de comunicación
IRQ -3 - Protocolos de transporte
CRQ -1 - Privacidad de los mensajes
CRQ -3 - Autenticación de usuarios</td></tr>
<tr><td>Descripción</td><td>El sistema conecta de forma automática los peers para establecer una conexión chat</td></tr>
<tr><td>Precondición</td><td>Haber iniciado ambos peers el comando específico para cada uno</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El sistema una vez los pares han ejecutado el comando busca el peer al que conectarse</td></tr><tr><td>P2</td><td>El sistema conecta ambos peers.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Ambos peers conectados</td></tr>
<tr><td>Comentarios</td><td></td></tr>
</table>

Tabla 35. Caso de uso 3

UC - 3	**ENVIAR MENSAJE**
Versión	1.0
Autor	Daniel Sanchidrian Jimeno
Fuente	Daniel Sanchidrian Jimeno
Objetivos Asociados	OBJ -1.1 - Seguridad y privacidad OBJ -1.5 - Flexibilidad de conexión
Requisitos Asociados	IRQ -2 - Puertos de comunicación CRQ -1 - Privacidad de los mensajes CRQ -2 -Integridad de los datos CRQ -5 -Retención de datos

Descripción	Uno de los *peers* envía un mensaje a otro *peer*.
Precondición	Haber inicializado la conexión del chat *libp2p*.
Secuencia normal	<table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario escribe un mensaje en la consola y lo envía al *peer* correspondiente</td></tr><tr><td>P2</td><td>El sistema muestra un historial del chat</td></tr></table>
Postcondición	Un mensaje nuevo enviado
Comentarios	

Tabla 36. Caso de uso 4

UC - 4	**RECIBIR MENSAJE**
Versión	1.0
Autor	Daniel Sanchidrian Jimeno
Fuente	Daniel Sanchidrian Jimeno
Objetivos Asociados	OBJ -1.1 - Seguridad y privacidad OBJ -1.5 - Flexibilidad de conexión
Requisitos Asociados	IRQ -2 - Puertos de comunicación CRQ -1 - Privacidad de los mensajes CRQ -2 -Integridad de los datos CRQ -5 -Retención de datos
Descripción	El usuario recibe en su chat un mensaje enviado por otro *peer*
Precondición	
Secuencia normal	<table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario recibe un mensaje en su chat</td></tr><tr><td>P2</td><td>El sistema muestra un historial del chat</td></tr></table>
Postcondición	Un mensaje recibido
Comentarios	

Tabla 37. Caso de uso 5

UC - 5	**FINALIZAR SESIÓN**
Versión	1.0
Autor	Daniel Sanchidrian Jimeno
Fuente	Daniel Sanchidrian Jimeno

<table>
<tr><td>Objetivos Asociados</td><td>OBJ -1 – Comunicación descentralizada
OBJ - 1.5 – Flexibilidad de conexión</td></tr>
<tr><td>Requisitos Asociados</td><td>IRQ - 4 – Identificadores de nodos
CRQ - 2 - Integridad de los datos
CRQ - 3 – Autenticación de usuarios
CRQ - 5 – Retención de datos</td></tr>
<tr><td>Descripción</td><td>El usuario cierra la consola o cancela la ejecución del comando de inicialización del chat.</td></tr>
<tr><td>Precondición</td><td>Usuario inicializado en el chat</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario cierra la consola o cancela el comando de uso del chat.</td></tr><tr><td>P2</td><td>El sistema finaliza la sesión.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Usuario cerró sesión y no recibirá más mensajes hasta volver a conectarse.</td></tr>
<tr><td>Comentarios</td><td></td></tr>
</table>

Caso de uso de la aplicación chat Libp2p.

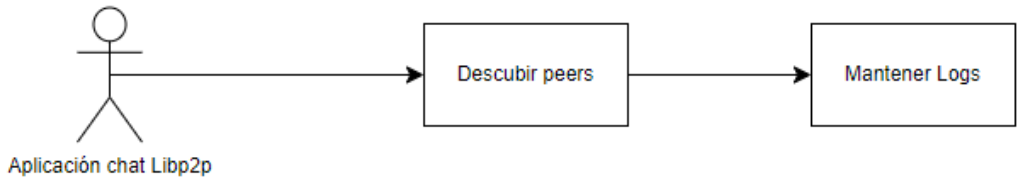

Ilustración 6. Esquema de interacción del chat Libp2p.

En esta ocasión, el actor principal es la aplicación del chat *libp2p*. Las acciones que debe seguir la misma son:

1. **Descubrir *Peers***. La aplicación conecta un *peer* con otro *peer* del listado (en este caso, solo tenemos la clave de uno, por lo que se conectará a ese).
2. **Mantener *Logs*.** La aplicación registrará un historial de mensajes intercambiados entre los *peers*. Una vez cerrada la sesión, se mantendrán hasta cerrar la consola.

Tabla 38. Caso de uso 6

UC - 6	**DESCUBIR *PEERS***
Versión	1.0
Autor	Daniel Sanchidrian Jimeno
Fuente	Daniel Sanchidrian Jimeno
Objetivos Asociados	OBJ - 1 – Comunicación descentralizada OBJ -1.1 – Seguridad y privacidad OBJ - 1.5 – Flexibilidad de conexión
Requisitos Asociados	IRQ - 1 – Información de *IP* y *HOST* IRQ - 2 – Puertos de comunicación IRQ - 3 – Protocolos de transporte IRQ - 4 – Identificadores de nodos CRQ - 2 – Integridad de los datos CRQ - 3 – Autenticación de usuarios CRQ - 5 – Retención de datos
Descripción	El sistema busca el *peer* al que tiene que conectarse
Precondición	No estar conectado al sistema
Secuencia normal	Paso: Acción P1: El usuario inicia el comando de ejecución del nodo. P2: El sistema busca en la red *libp2p* el *peer* al que debe conectarse.
Postcondición	*Peers conectados.*
Comentarios	

Tabla 39.Caso de uso 7

UC - 7	**MANTENER *LOGS***
Versión	1.0
Autor	Daniel Sanchidrian Jimeno
Fuente	Daniel Sanchidrian Jimeno
Objetivos Asociados	OBJ - 1 – Comunicación descentralizada OBJ -1.1 – Seguridad y privacidad OBJ - 1.5 – Flexibilidad de conexión
Requisitos Asociados	IRQ - 4 – Identificadores de nodos CRQ - 2 – Integridad de los datos

<table>
<tr><td></td><td>CRQ - 3 – Autenticación de usuarios
CRQ - 5 – Retención de datos</td></tr>
<tr><td>Descripción</td><td>El sistema mantiene un historial de los mensajes enviados entre los dos peers.</td></tr>
<tr><td>Precondición</td><td>Chat inicializado entre los dos peers.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario manda un mensaje a un peer.</td></tr><tr><td>P2</td><td>El usuario recibe un mensaje de un peer</td></tr><tr><td>P3</td><td>El sistema mantiene un historial de los mensajes enviados.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Un historial del chat entre los dos peers.</td></tr>
<tr><td>Comentarios</td><td></td></tr>
</table>

4.5. REQUISITOS NO FUNCIONALES.

Tabla 40. Requisito no funcional 1

NFR - 1	RENDIMIENTO
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de ser capaz de manejar un gran volumen de mensajes y usuarios simultáneos sin experimentar retardos en la respuesta del sistema
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 41.Requisito no funcional 2

NFR - 2	ESCALABILIDAD
Versión	1.0

Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de ser capaz de escalar para poder reaccionar a un aumento del número de usuarios y mensajes sin degradación
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 42.Requisito no funcional 3

NFR - 3	**DISPONIBILIDAD**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de estar disponible para su acceso la mayor parte del tiempo teniendo en cuenta el tiempo mínimo de inactividad por mantenimiento.
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 43. Requisito no funcional 4

NFR - 4	**FIABILIDAD**
Versión	1.0

Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de ser confiable y resistente a fallos con mecanismos de recuperación para manejar errores de red y seguridad en el envío de mensajes
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 44. Requisito no funcional 5

NFR - 5	**SEGURIDAD**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de tener medidas de seguridad sólidas para poder garantizar a los usuarios una integridad y privacidad de las comunicaciones, incluyendo la autenticación, cifrado de extremo a extremo y prevención de ataques maliciosos.
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 45.Requisito no funcional 6

NFR - 6	**INTEROPERABILIDAD**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema deberá ser compatible con otras implementaciones de *libp2p* para así poder permitir la comunicación con nodos que utilicen diferentes tecnologías y plataformas
Importancia	Media
Urgencia	Media
Estabilidad	Máxima
Comentarios	

Tabla 46. Requisito no funcional 7

NFR - 7	**PORTABILIDAD**
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de ser fácil de desplegar en diferentes entornos de Windows sin requerir modificaciones significativas
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 47. Requisito no funcional 8

NFR - 8	USABILIDAD
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema deberá tener una interfaz de usuario que sea intuitiva y fácil de usar para usuarios menos experimentados.
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 48. Requisito no funcional 9

NFR - 9	MANTENIBILIDAD
Versión	1.0
Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de ser fácil de mantener y actualizar.
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

Tabla 49. Requisito no funcional 10

NFR - 10	EFICIENCIA
Versión	1.0

Autor	Daniel Sanchidrián Jimeno
Fuentes	Daniel Sanchidrián Jimeno
Objetivos asociados	Todos
Requisitos Asociados	Todos
Descripción	El sistema debe de utilizar eficientemente los recursos del sistema como la memoria, el ancho de banda de red o el espacio en el disco duro, para así poder minimizar el impacto en el rendimiento del sistema.
Importancia	Alta
Urgencia	Alta
Estabilidad	Máxima
Comentarios	

5. INGENIERÍA INVERSA

5.1. INGENIERÍA INVERSA DEL DIALER.

En este apartado, definiremos el concepto de análisis estático, y su funcionamiento dentro del código seleccionado a analizar, y el de análisis dinámico.

5.1.1. Análisis estático.

El análisis estático se realiza sin ejecutar el código de la aplicación, es decir, procederemos a analizar el código de la aplicación para poder sacar fallas de este (si las hubiera) o una mejora del código del que disponemos. A primera vista podremos comprobar si existen o no errores, pero al complementarlo con el análisis dinámico, dispondremos de un estudio global, ya que los fallos que no encontremos en este análisis los podremos encontrar con este segundo análisis.

El código del *dialer* que vamos a analizar es el siguiente:

```
import { createFromJSON } from '@libp2p/peer-id-factory'
import { multiaddr } from '@multiformats/multiaddr'
import { createLibp2p } from './libp2p.js'
import peerIdDialerJson from './peer-id-dialer.js'
import peerIdListenerJson from './peer-id-listener.js'
import { stdinToStream, streamToConsole } from './stream.js'

async function run () {
            const [idDialer, idListener] = await Promise.all([
             createFromJSON(peerIdDialerJson),
             createFromJSON(peerIdListenerJson)
      ])

const nodeDialer = await createLibp2p({
            peerId: idDialer,
            addresses: {
      listen: ['/ip4/0.0.0.0/tcp/0']
       }
      })
console.log('Dialer ready, listening on:')
            nodeDialer.getMultiaddrs().forEach((ma) => {
            console.log(ma.toString())
      })
```

```
      constlistenerMa=multiaddr(`/ip4/127.0.0.1/tcp/10333/p2p/${idListene
r.toString()}`)
            const              stream              =              await
nodeDialer.dialProtocol(listenerMa,'/chat/1.0.0')

                  console.log('Dialer  dialed  to  listener  on  protocol:
/chat/1.0.0')
                  console.log('Type a message and see what happens')

             // Send stdin to the stream
             stdinToStream(stream)
            // Read the stream and output to console
            streamToConsole(stream)
      }
      run()
```

A continuación, procederemos a explicar cada una de las partes del código con el objetivo de comprender cada una de ellas de manera individual:

1. ***Imports***.

Lo primero que podemos comprobar es que importamos todas las funciones y módulos relacionados con el *peer-id*, direcciones *multiaddr* y la creación de la instancia de *libp2p*.

```
      import { createFromJSON } from '@libp2p/peer-id-factory'
      import { multiaddr } from '@multiformats/multiaddr'
      import { createLibp2p } from './libp2p.js'
      import peerIdDialerJson from './peer-id-dialer.js'
      import peerIdListenerJson from './peer-id-listener.js'
      import { stdinToStream, streamToConsole } from './stream.js'
```

2. **Función asíncrona**.

La función asíncrona, en este contexto, es denominada *run*, y puede contener el operador ***await,*** que se encarga de esperar el resultado de una expresión, en este caso, una promesa.

Por otro lado, una promesa es un objeto que representa un valor, el cual puede estar disponible ahora, en el futuro o nunca. En este caso concreto, nos es de utilidad, pues no sabemos cuándo va a responder uno de los nodos al otro.

Así, dentro de la función asíncrona con la que contamos vamos a tener la ***Promise.all,*** un conjunto de promesas esperando a ser resueltas al mismo tiempo.

Esta parte del código sugiere que ***peerIdDialerJson*** y ***peerIdListenerJson*** son datos en formato *JSON* que representan identificadores de pares, y se utilizan para crear instancias de estos identificadores de pares mediante la función ***createFromJSON***.

```
async function run () {
            const [idDialer, idListener] = await Promise.all([
             createFromJSON(peerIdDialerJson),
             createFromJSON(peerIdListenerJson)
      ])
```

3. ***NodeDialer***.

En esta parte del código se utiliza la función *createLibp2p* para crear una instancia de *libp2p* llamada *nodeDialer*.

Así pues, se configura *libp2p* para escuchar en todas las direcciones *IPv4* (/ip4/0.0.0.0) en un puerto *TCP* aleatorio (/tcp/0). Esto provoca que el nodo esté disponible para conexiones entrantes en todas las interfaces de red. Esta instancia de libp2p se utilizará para la parte de *dialing* en el contexto de la red libp2p.

```
const nodeDialer = await createLibp2p({
            peerId: idDialer,
            addresses: {
      listen: ['/ip4/0.0.0.0/tcp/0']
       }
      })
```

4. ***Console.log***.

Lo primero que salta a la vista cuando vemos el código es la utilización de Console.log, que sirve para mostrar al usuario por pantalla el mensaje "*dialer ready, listening on*".

Mediante el uso del método *getMultiaddrs* de la instancia de libp2p (*nodeDialer*), obtendremos un listado de las direcciones multiaddr a las que está escuchando el nodo. Luego, itera sobre estas direcciones y las imprime en la consola mediante *console.log(ma-toString()).*

```
console.log('Dialer ready, listening on:')
            nodeDialer.getMultiaddrs().forEach((ma) => {
            console.log(ma.toString())
      })
```

5. **Especificación de constantes**.

A continuación, procederemos a explicar las funciones o líneas de código para comprender el funcionamiento de cada una en la ingeniería inversa de este proyecto:

const listeneMa=multiaddr(/ip4/127.0.0.1/tcp/10333/p2p/${idListener.toString()}); Se crea una dirección *multiaddr* que representa la dirección del nodo remoto al que se desea marcar. En este caso, la dirección cuenta con la IP 127.0.0.1, el puerto 10333 y el identificador de par del *Listener*.

const stream = nodeDialer. dialProtocol(listenerMa, '/chat/1.0.0'); Se utiliza el método *dialProtocol* del nodo para establecer una conexión al nodo remoto utilizando la dirección *multiaddr* y el protocolo /chat/1.0.0. Esto devuelve un *stream* que representa la conexión.

console.log('Dialer dialed to listener on protocol: /chat/1.0.0'); imprime un mensaje en la consola indicando que el marcador se ha conectado al *listener* utilizando el protocolo /chat/1.0.0.

console.log('Type a message and see what happens'); imprime el mensaje "*Escribe un mensaje y mira qué ocurre*" en la consola, indicando que puedes escribir a un nodo un mensaje después de establecer la conexión.

stdinToStream(stream); y **stdinToConsole(stream);** Estas funciones están relacionadas con la manipulación de flujos de entrada/salida y están diseñadas para enviar la entrada estándar del flujo y mostrar el contenido de este en la consola.

run(); Finalmente, la función *run* se invoca para iniciar el proceso, lo que provoca que todas las operaciones descritas anteriormente sean ejecutadas.

```
      const                          listenerMa                          =
multiaddr(`/ip4/127.0.0.1/tcp/10333/p2p/${idListener.toString()}`)
         const   stream   =   await   nodeDialer.dialProtocol(listenerMa,
'/chat/1.0.0')

        console.log('Dialer dialed to listener on protocol: /chat/1.0.0')
        console.log('Type a message and see what happens')

        // Send stdin to the stream
        stdinToStream(stream)
        // Read the stream and output to console
        streamToConsole(stream)
      }

      run()
```

En resumen, podemos tener claro que el código analizado de forma estática realiza una conexión desde el nodo ***Dialer*** a un nodo remoto llamado ***Listener*** a través de *libp2p*, utilizando el protocolo **/chat/1.0.0.** Una vez inicializado, este se queda esperando la entrada del usuario para enviar un mensaje y muestra el flujo de la comunicación a través de la consola.

5.1.2. Análisis dinámico.

El análisis dinámico trata de analizar el código del proyecto una vez ejecutado, con miras a poder obtener alguna falla del mismo, sobre todo en la parte visual. En esta ocasión, vamos a analizar la parte funcional del *dialer* para ver cómo funciona.

Lo primero que debemos hacer es acceder a nuestra consola de Windows y movernos hasta el directorio en el que tenemos guardado el proyecto. En mi caso particular es la siguiente ruta:

dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src

Una vez desplazados en la ruta a través de la consola de comando deberemos ejecutar el comando "*node dialer.js*" para poder comenzar la ejecución. Pero primero

deberemos ejecutar el *listener* para poder realizar la conexión, ya que si no nos aparecerá el siguiente error:

```
C:\Users\dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src>node dialer.js
Dialer ready, listening on:
/ip4/192.168.0.28/tcp/52943/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
/ip4/127.0.0.1/tcp/52943/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
/ip4/172.19.144.1/tcp/52943/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
node:internal/process/promises:289
            triggerUncaughtException(err, true /* fromPromise */);
            ^

Error: connect ECONNREFUSED 127.0.0.1:10333
    at TCPConnectWrap.afterConnect [as oncomplete] (node:net:1595:16) {
  errno: -4078,
  code: 'ECONNREFUSED',
  syscall: 'connect',
  address: '127.0.0.1',
  port: 10333
}

Node.js v20.10.0

C:\Users\dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src>
```

Ilustración 7. Error por fallo de conexión dialer-listener.

El error principal que podemos encontrar al inicializar nuestra aplicación es el anteriormente mencionado, que básicamente muestra que el *dialer.js* no puede encontrar un *listener.js* para poder establecer la conexión con él de manera correcta. Ahora vamos a analizar una a una la traza de código que se muestra por pantalla:

- ***ECONNREFUSED*** indica que la conexión que se realiza al localhost(127.0.0.1) en el puerto 10333 fue rechazada.
- ***-4078*** es un código que corresponde a una conexión rechazada.
- ***TCPConnectWrap.afterConnect*** es una función que está relacionada con la conexión TCP después de un intento de conexión.

A continuación, procederemos a iniciar ambos nodos para poder establecer una conexión entre ellos y conocer el funcionamiento del código, partiendo de cada una de las líneas especificadas en la consola de comando:

```
C:\Users\dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src>node dialer.js
Dialer ready, listening on:
/ip4/192.168.0.28/tcp/53149/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
/ip4/127.0.0.1/tcp/53149/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
/ip4/172.19.144.1/tcp/53149/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
Dialer dialed to listener on protocol: /chat/1.0.0
Type a message and see what happens
```

Ilustración 8. Dialer conectado

/ip4/192.168.0.28/tcp/53149/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP.

Esta línea explica, en primer lugar, que se está utilizando el protocolo IPV4 en la dirección IP local 192.168.0.28. Si seguimos leyendo, podemos ver que se está utilizando el protocolo TCP en el puerto 10333. En la parte final hace referencia a la identificación del nodo por la clave publica, que, en el caso de libp2p, tiene el formato de Qm.

/ip4/127.0.0.1/tcp/53149/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP

Es exactamente igual a la del punto anterior, lo único que cambia la IP a 127.0.0., que hace referencia al *localhost*, es decir, se está conectando al propio nodo de la misma máquina.

/ip4/172.19.144.1/tcp/53149/p2p/Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP

Este último apartado sería igual que los anteriores, pero con la IP 172.19.144.1.

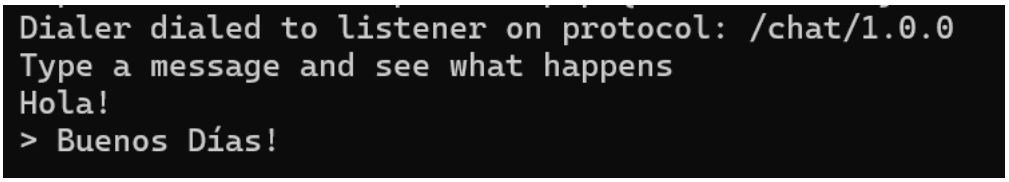

Ilustración 9. Prueba de funcionamiento del chat.

Una vez iniciados ambos nodos podemos mantener una conversación, indicando con anterioridad que nos hemos conectado a la clave pública que se encuentra en el *dialer*. El primer mensaje es enviado desde el *dialer* y el segundo desde el *listener*.

5.2. INGENIERÍA INVERSA DEL LISTENER.

En este apartado procederemos a analizar el código completo del *listener.json* para conocer su composición y, una vez estudiado, lo analizaremos por partes para llegar a un entendimiento claro de cómo funciona. Utilizaremos la misma técnica que en el apartado anterior, aplicando un análisis estático (para analizar el código) y un análisis dinámico para ver cómo se muestra por consola el código y su funcionamiento.

5.2.1. Análisis estático.

La explicación seguirá el modelo patrón anteriormente explicado, con las mismas fases. El código general del que extraeremos las diferentes partes se presentaría de la siguiente manera:

```
import { createFromJSON } from '@libp2p/peer-id-factory'
import { createLibp2p } from './libp2p.js'
import peerIdListenerJson from './peer-id-listener.js'
import { stdinToStream, streamToConsole } from './stream.js'

async function run () {
  const idListener = await createFromJSON(peerIdListenerJson)
  const nodeListener = await createLibp2p({
    peerId: idListener,
    addresses: {
      listen: ['/ip4/0.0.0.0/tcp/10333']
    }
  })

  nodeListener.addEventListener('peer:connect', (evt) => {
    const remotePeer = evt.detail
    console.log('connected to: ', remotePeer.toString())
  })

  await nodeListener.handle('/chat/1.0.0', async ({ stream }) => {
    stdinToStream(stream)
    streamToConsole(stream)
  })

  console.log('Listener ready, listening on:')
  nodeListener.getMultiaddrs().forEach((ma) => {
    console.log(ma.toString())
  })
}
run()
```

1. ***Imports***.

Lo primero que podemos comprobar es que importamos todas las funciones y módulos relacionados con el *peer-id*, direcciones *multiaddr* y la creación de la instancia de *libp2p*.

```
import { createFromJSON } from '@libp2p/peer-id-factory'
import { createLibp2p } from './libp2p.js'
import peerIdListenerJson from './peer-id-listener.js'
import { stdinToStream, streamToConsole } from './stream.js'
```

2. **Función asíncrona**.

La función asíncrona, de nuevo, es denominada *run*, y puede contener el operador ***await.*** Podemos comprobar que, dentro de este apartado, vienen definidas dos constantes: *idListener* (recoge el id del Listener al que queremos conectar) y *nodeListener* (es un objeto del tipo Listener que proporciona información sobre el mismo, asociándolo al id del Listener y a la ip de la que debe escuchar).

```
async function run () {
  // Create a new libp2p node with the given multi-address
  const idListener = await createFromJSON(peerIdListenerJson)
  const nodeListener = await createLibp2p({
    peerId: idListener,
    addresses: {
      listen: ['/ip4/0.0.0.0/tcp/10333']
    }
  })
```

3. ***NodeListener*.**

NodeListener se encarga, a través de *addEventListener*, de mostrar por pantalla al peer la información del par remoto al que se ha conectado.

```
nodeListener.addEventListener('peer:connect', (evt) => {
    const remotePeer = evt.detail
    console.log('connected to: ', remotePeer.toString())
  })
```

4. ***NodeListener.handle*.**

Utilizamos el método *handle* de *libp2p* para manejar todos los mensajes del protocolo /chat/1.0.0. Cuando recibimos un mensaje se envía la entrada estándar (*stdinToStream*) y lee el flujo, para luego ser mostrado por pantalla con *streamToConsole.*

```
await nodeListener.handle('/chat/1.0.0', async ({ stream }) => {
  // Send stdin to the stream
  stdinToStream(stream)
  // Read the stream and output to console
  streamToConsole(stream)
})
```

5. **Función *forEach()*.**

Primero, con *console.log* mostraremos, por pantalla, que el listener está listo para escuchar. El mismo se complementa a través del *forEach*, que recorre un listado de todas las direcciones *multiaddrs* de las que está escuchando. Finalmente, se invoca la función *run()* para iniciar el proceso de escucha.

```
console.log('Listener ready, listening on:')
 nodeListener.getMultiaddrs().forEach((ma) => {
  console.log(ma.toString())
 })
}
run()
```

5.2.2. Análisis dinámico.

Como anteriormente se ha mencionado, procederemos a realizar el análisis dinámico del *listener*, mediante la visualización de las trazas del código en la consola para comprender de una manera correcta lo mostrado.

Echando un primer vistazo observamos el arranque del *listener* y el acceso al mismo. En este caso, desde la consola debemos dirigirnos a la ruta en la que se encuentra el *listener* dentro de nuestro equipo. En mi caso particular sería:

\dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src

Una vez dentro de la ruta encontraremos varios archivos que configuran el código.

Nombre	Fecha de modificación	Tipo	Tamaño
al principio de este año			
stream.js	31/01/2024 9:38	JSFile	3 KB
peer-id-listener.js	31/01/2024 9:23	JSFile	3 KB
peer-id-dialer.js	31/01/2024 9:22	JSFile	3 KB
listener.js	31/01/2024 9:18	JSFile	2 KB
libp2p.js	31/01/2024 9:08	JSFile	3 KB
dialer.js	31/01/2024 9:01	JSFile	6 KB

Ilustración 10. Componentes Json

En este caso específico queremos arrancar el archivo llamado *listener.js* y se ejecuta con el siguiente comando: ***node listener.js***

```
C:\Users\dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src>node listener.js
Listener ready, listening on:
/ip4/192.168.0.28/tcp/10333/p2p/QmcrQZ6RJdpYuGvZqD5QEHAv6qX4BrQLJLQPQUrTrzdcgm
/ip4/127.0.0.1/tcp/10333/p2p/QmcrQZ6RJdpYuGvZqD5QEHAv6qX4BrQLJLQPQUrTrzdcgm
/ip4/172.19.144.1/tcp/10333/p2p/QmcrQZ6RJdpYuGvZqD5QEHAv6qX4BrQLJLQPQUrTrzdcgm
```

Ilustración 11. Prueba comando listener.js

Como hemos explicado anteriormente, en el dialer tenemos todos los datos de nuestro nodo, especificando la clave pública, los puertos y las IP a las que se realiza la conexión.

```
connected to:  Qma3GsJmB47xYuyahPZPSadh1avvxfyYQwk8R3UnFrQ6aP
> Hola!
Buenos Días!
```

Ilustración 12. Muestra de funcionamiento del chat.

Una vez iniciada la conexión, podemos comprobar que nos hemos conectado correctamente a la clave pública del otro nodo (en este caso, al del *dialer*). Uno de los problemas que podríamos arreglar con el análisis dinámico es la especificación de quién ha mandado cada uno de los mensajes, estableciendo por ejemplo un ID o *username* a cada uno de los *peers*.

Cuando cortamos la conexión de uno de los nodos nos saldrá el siguiente error. Este surge al desconectar uno de los nodos de forma inesperada, ya sea por perdida de conexión o por cerrar la consola en uno de los *peers* (se mostrará el error en el *peer* contrario).

Ilustración 13. Muestra de error de desconexión.

Como podemos observar, el error proviene de una promesa que no se estaba capturando. En este caso, es prácticamente lo mismo, cambiando que el código de error -4077 corresponde a una conexión restablecida.

6. PRUEBAS

6.1. PRUEBA DE INSTALACIÓN.

La prueba de instalación se ha llevado a cabo en dos ordenadores con diferente versión del sistema operativo para comprobar su correcta ejecución en ambos ámbitos. Primeramente, se ha llevado a cabo en los siguientes sistemas:

- Ordenador: Dell 13th Gen Intel(R) Core (TM) i7-1365U 1.80 GHz 16,0 GB RAM
- Sistema Operativo: Windows 11
- Resultado de la instalación del sistema chat: Correcto

En caso del otro sistema:

- Ordenador: Sobremesa con Ryzen 5 3600 (6 Núcleos y 12 hilos) y 16gb Ram
- Sistema operativo: Windows 10
- Resultado de la instalación del sistema chat: Correcto

6.2. PRUEBAS DEL SISTEMA

6.2.1. Prueba de rendimiento

Las pruebas de rendimiento dependen de la conexión a internet y del dispositivo en el que estemos trabajando. El tiempo de carga de la aplicación chat *libp2p* es rápido, ya que todas las operaciones se realizan en menos de un segundo. La operación que más se tarda en ejecutar es la búsqueda del *peer,* ya que este depende de que el otro usuario active su conexión con el sistema.

6.2.2. Prueba de robustez.

Si la conexión a internet cae, el sistema caerá y desconectará ambos *peers*. Si cerramos la consola de Windows, el sistema chat se cerrará y terminará la conexión, dando igual cuál haya sido el *peer* que perdió la conexión.

6.2.3. Prueba de seguridad

Libp2p cuenta con cifrado de transporte para poder asegurar la comunicación entre pares. El uso de la clave pública y privada provoca que solo los pares que estén conectados y verificados (cada uno tiene una identidad única) tengan la posibilidad de descifrar los mensajes.

Libp2p no almacena ninguno de los datos de usuarios y los mensajes enviados por el chat son efímeros, es decir, una vez acabada la conexión entre los pares ya no se vuelve a tener rastro de ellos.

6.3. PRUEBAS DE CAJA NEGRA

Las pruebas de Caja negra son un tipo de pruebas del sistema basado en la funcionalidad del mismo para un individuo que no conoce el funcionamiento de este. De hecho, sirven para comprobar el funcionamiento del sistema y si el mismo trabaja de la forma que se espera.

Ejemplo 1 – Inicialización del chat.

- Descripción del caso: Comprobar que ambos nodos pueden establecer una conexión entre sí para iniciar la comunicación.
- Precondiciones: Ambas consolas deben estar en ejecución y preparado el comando de inicio del chat.

Caso 1.1 – *Datos de entrada*: Un usuario introduce el comando de inicialización del chat mientras el otro usuario tiene el *listener* activo.

Resultado esperado: El sistema realiza la conexión de los dos nodos satisfactoriamente.

Caso 1.2 - *Datos de entrada*: Un usuario introduce el comando de inicialización del chat, mientras el otro usuario no tiene el *listener* activo.

Resultado esperado: El sistema muestra en pantalla un mensaje de error indicando que la conexión no puede ser posible en esos momentos.

Ejemplo 2 - Envío de mensaje de un peer a otro.

- Descripción del caso: Una vez realizada la conexión entre los dos *peers* , el sistema mantendrá una conexión entre estos para poder realizar una comunicación chat fluida.
- Precondiciones: Ambos nodos han sido conectados satisfactoriamente.

Caso 2.1 - *Datos de entrada*: Uno de los *peers* envía un mensaje al otro para comenzar la comunicación.

Resultado esperado: El sistema mostrará en la otra ventana del chat el mensaje enviado por el otro *peer,* manteniendo un historial de los mensajes.

Caso 2.2 - *Datos de Entrada*: El peer que recibió el mensaje visualiza este en su consola y lo responde.

Resultado esperado: El sistema mostrará un historial de los mensajes enviados entre estos.

Ejemplo 3 - Manejo de desconexión.

- Descripción del caso: Verificar que el sistema maneja de una forma correcta las desconexiones del sistema.
- Precondiciones: Ambos nodos han sido conectados satisfactoriamente.

Caso 3.1 - *Datos de entrada*: Uno de los nodos cierra la consola de comandos en la que inicializó el chat.

Resultado esperado: La consola de comandos del nodo que no se cerró recibirá un mensaje de error por pantalla de que sufrió una desconexión el nodo opuesto.

Caso 3.2 - *Datos de entrada*: Uno de los nodos pierde la conexión con la consola de comandos en la que inicializó el chat.

Resultado esperado: La consola de comandos del nodo que no se cerró recibirá un mensaje de error por pantalla de que sufrió una desconexión el nodo opuesto.

6.4. MANUAL DE INSTALACIÓN

La instalación del proyecto es muy sencilla, pues no necesitaremos instalar nada, ya que ejecutaremos los ficheros *json* que se encuentran dentro del proyecto en la consola de comandos que veremos en el siguiente apartado en el manual de uso.

Los únicos requisitos que necesitaremos para hacer funcionar el proyecto al que hemos aplicado la ingeniería inversa son: Windows (probado en las versiones 10 y 11) y un programa de edición de código (en este caso hemos utilizado *visual estudio code* para poder añadir extensiones).

Si quisiéramos modificar el código para hacerlo más comprensible para cualquiera, o incluso para añadir alguna implementación, deberíamos cargar el proyecto en *visual estudio* e instalar las extensiones necesarias. En mi caso tengo instalado:

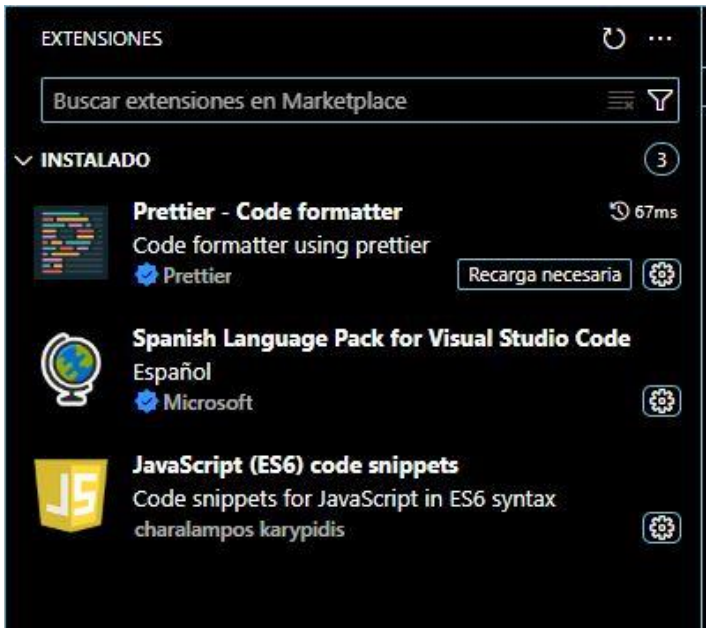

Ilustración 14. Extensiones Visual Studio Code

Prettier: Es una guía de estilo que se utiliza para permitir a los programadores usar un solo estilo en el código haciendo que este sea más fácil de leer y de mantener.

Spanish Language Pack for Visual Studio Code: Cambia el idioma de nuestro editor de texto al idioma deseado, en este caso el español.

JavaScript (ES6) code snippets: Ofrece una colección de fragmentos de código que pueden ahorrar tiempo y esfuerzo al escribir código *JavaScript*

Por otra parte, dentro del código a analizar, observamos distintas clases y paquetes que conforman el proyecto para poder ejecutarlo correctamente. El proyecto estudiado se compone de los siguientes paquetes y clases:

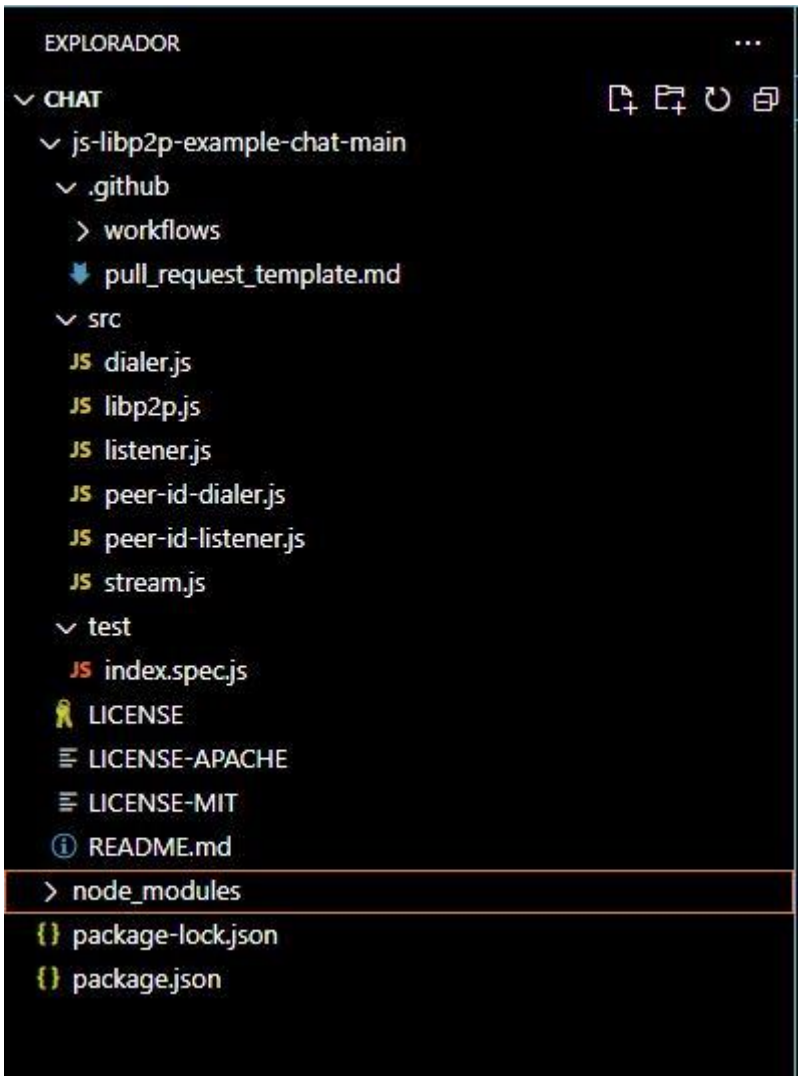

Ilustración 15: Paquetes del proyecto y clases

Podemos observar que dentro del paquete *src* se incluye toda la funcionalidad del chat (sobre todo los *Json* que ejecutaremos en nuestra consola de comandos). Mientras, en la carpeta principal del proyecto tenemos *package.json* , que nos ayudará a realizar un seguimiento más detallado de todos los paquetes instalados en el proyecto estudiado.

6.5. MANUAL DE USO

En este apartado procederemos a explicar de forma detallada como sería la forma correcta de utilizar la herramienta a analizar mediante ingeniería inversa. Este documento, más que ser un Trabajo de Fin de Frado, es una guía instructiva para conocer el manejo y modificación a nuestro antojo del comportamiento del proyecto.

Primeramente, y de manera sencilla, comenzaremos con la descarga del proyecto desde un repositorio *GitHub,* totalmente público, donde el creador ha dejado subido el

contenido necesario para ejecutar la aplicación chat. Dicho repositorio se encuentra en la siguiente ruta:

https://github.com/libp2p/js-libp2p-example-chat .

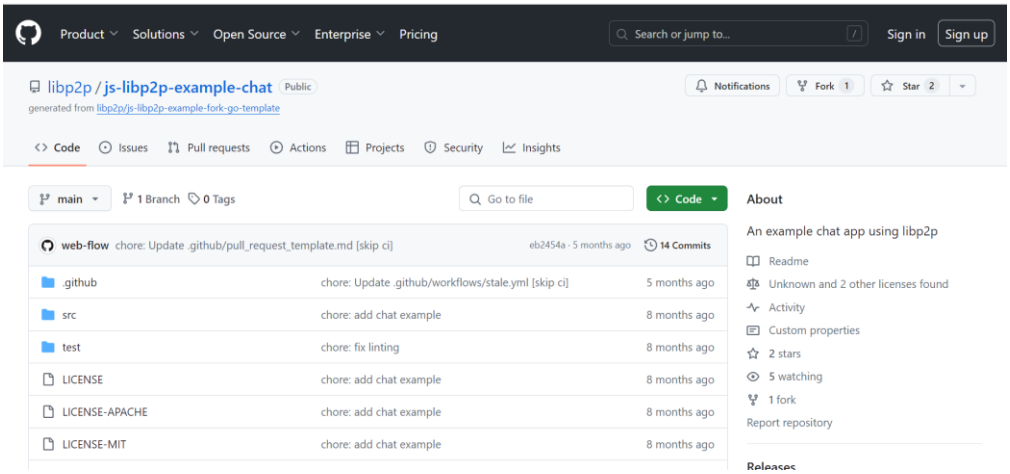

Ilustración 16. Repositorio github del proyecto

Una vez descargado el código, podemos trasladarlo a la carpeta que deseemos. En mi caso particular, la ruta en la que tengo guardado el proyecto es la siguiente:

C:\Users\dsanchidrian\Downloads\js-libp2p-example-chat-main

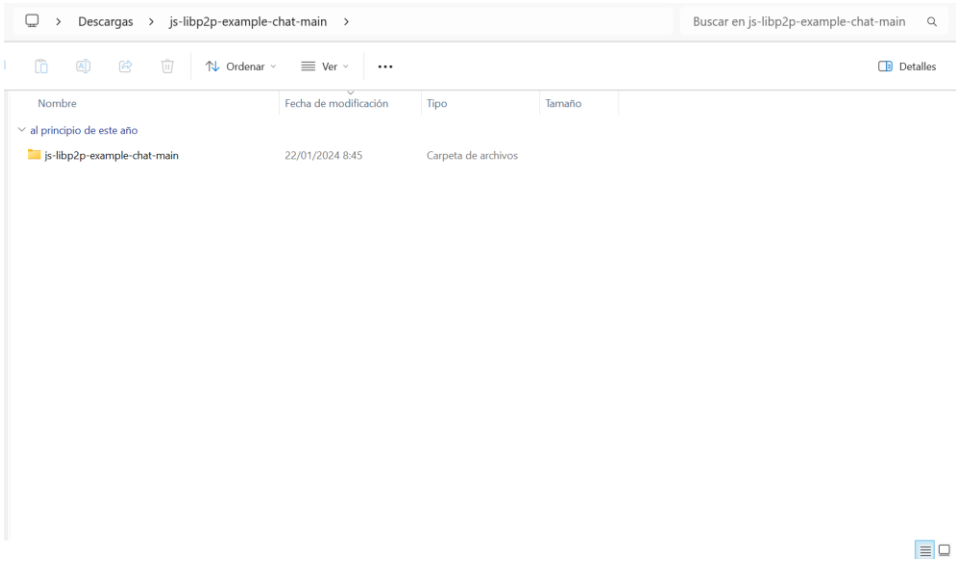

Ilustración 17. Carpeta de ubicación del archivo

Con el proyecto ya ubicado, debemos acceder a este para obtener la carpeta llamada *src*, pues contiene los ficheros que vamos a utilizar para arrancar la aplicación. En mi caso se encuentra en:

C:\Users\dsanchidrian\Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src

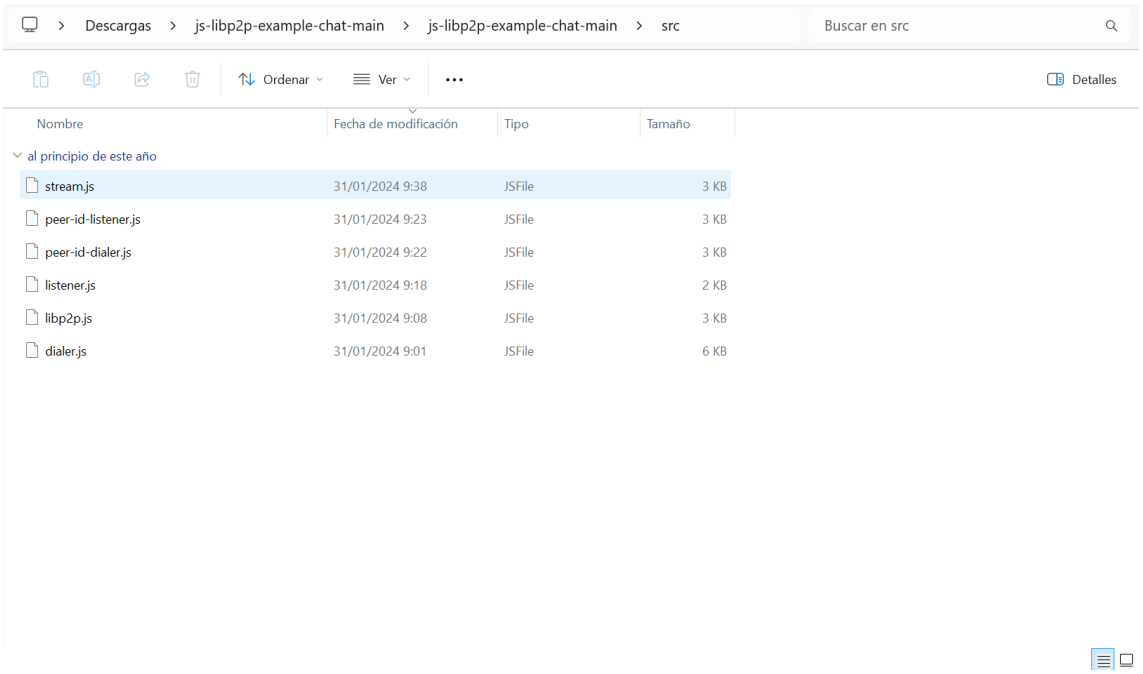

Ilustración 18: Componentes del proyecto

El siguiente paso sería abrir la consola de comandos para acceder a la carpeta *src* y arrancar por comando el fichero *Json*, tanto el *listener* como el *dialer* (cada uno de ellos debe de ser ejecutado en una consola distinta para establecer la conexión). Para acceder a la consola de comandos podemos hacerlo a través del buscador de Windows escribiendo ***cmd*** o en Windows 11 a través del siguiente icono:

Ilustración 19: Icono consola en la versión Windows 11.

Con ambas consolas abiertas, nuestra siguiente acción será acceder a la ubicación de la carpeta *src* en cada una de ellas. Una vez dentro, y a través del comando ***dir,*** podremos ver todos los ficheros que se encuentran dentro de este directorio. Para acceder a nuestra carpeta ***src*** deberíamos utilizar el siguiente comando:

cd Downloads\js-libp2p-example-chat-main\js-libp2p-example-chat-main\src

En mi caso particular es esta ruta, pero según la zona donde quede guardado tras su descargar será una ruta u otra.

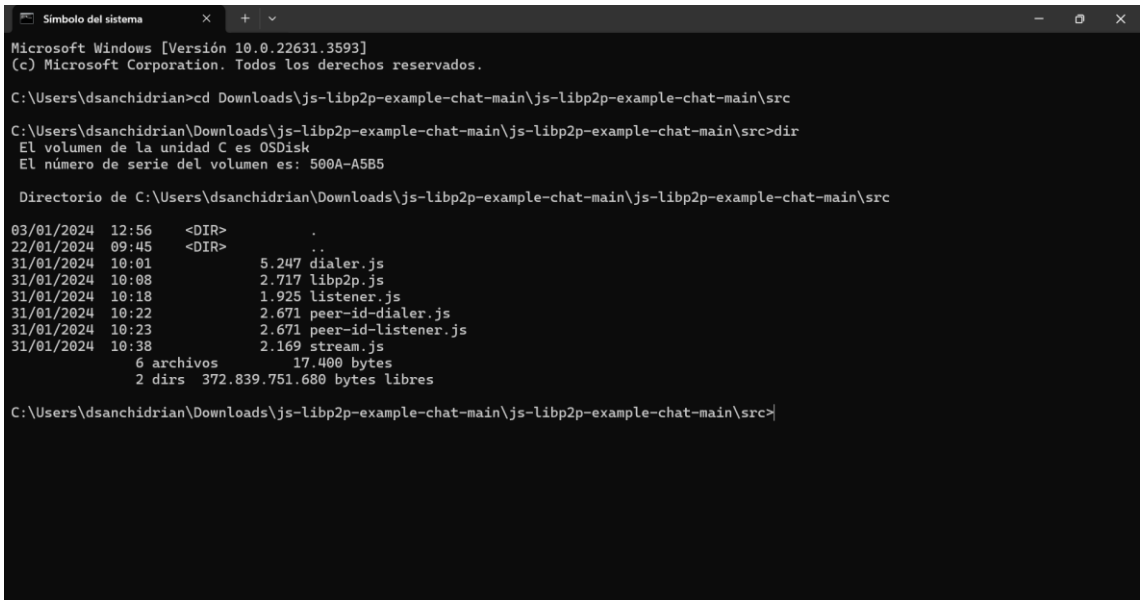

Ilustración 20: Comandos para movernos a la carpeta del proyecto

Ahora ejecutamos con el comando *node* los ficheros que necesitamos para abrir el chat entre las consolas. Si se da el caso de ejecutar primero el *dialer.js*, nos saldrá un error por pantalla de que no hay un *listener* escuchando, por lo que la comunicación no va a ser posible.

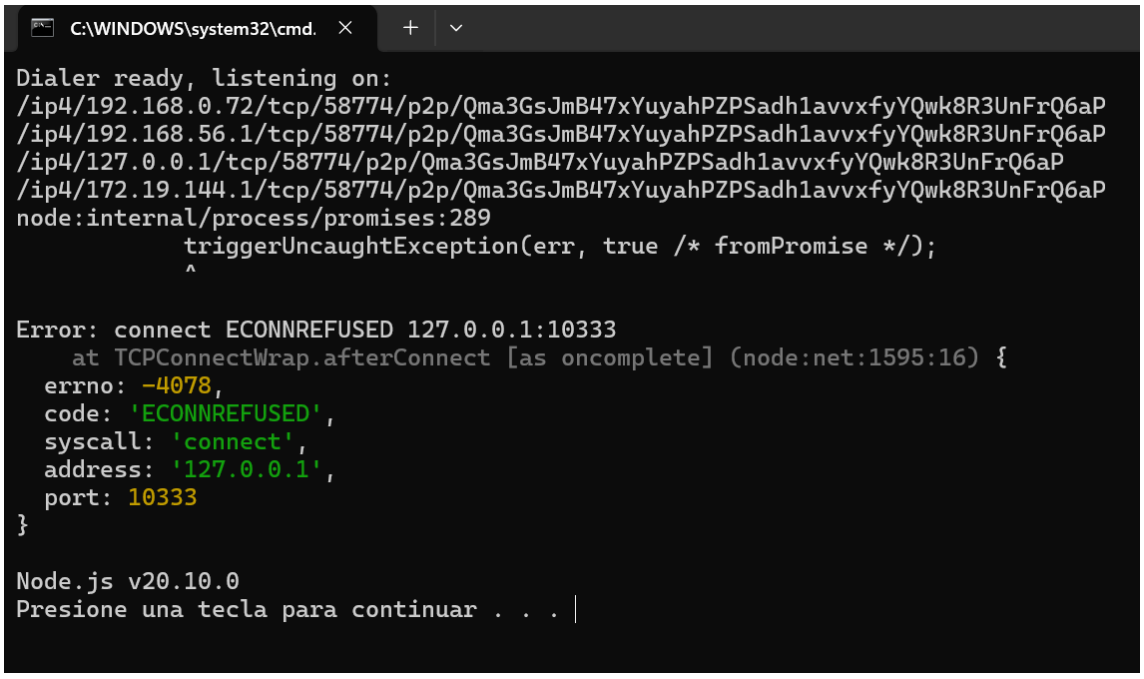

Ilustración 21: Error de conexión del Dialer.

Una vez solucionado el posible error, arrancamos el listener primero y, una vez cargado, ejecutamos el *dialer*. Para ello, utilizaremos los comandos de ***node listener.js*** y

node dialer.js. Una vez inicializada la conexión podremos utilizar el chat entre nosotros sin ningún problema.

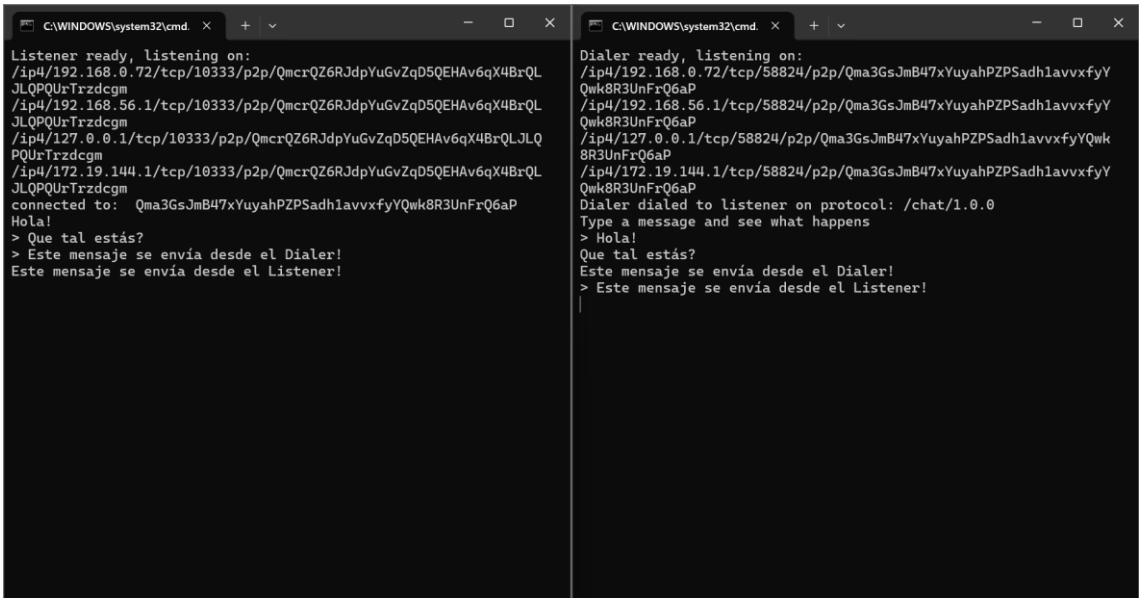

Ilustración 22. Prueba de chat funcionando

Otra forma de ejecutar los chats de una forma directa es con la creación de un *script*. Para crearlo solo necesitamos colocar dentro de un fichero de texto la siguiente línea de comandos:

@echo off

cd C:\ruta\del\scripts

node listener.js

pause

Una vez realizado dentro del fichero *.txt*, debemos guardarlo este fichero en formato *.bat* y el nombre que deseemos. En mi caso particular se guardó como *listener.bat* y *dialer.bat*:

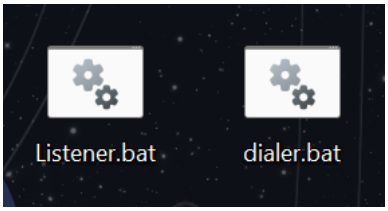

Ilustración 23. Ficheros .bat de inicio rápido

7. CONCLUSIÓN Y AMPLIACIONES

La conclusión general obtenida tras aplicar una ingeniería inversa a este proyecto (un chat *libp2p* entre consolas de Windows) es que *libp2p,* como librería, tiene un gran potencial para establecer comunicaciones seguras e independientes de grandes empresas monopolistas.

Este proyecto direcciona desde lo particular, lo mínimo, a lo general o máximo, es decir, para saber realizar un proyecto de una forma mucho más grande primero tendremos que centrarnos en el caso base. Así, en este contexto, partimos desde un caso base para intuir un proyecto a mayor escala, donde podríamos añadir toda serie de ampliaciones.

Por ello, gracias a este estudio he comprendido mucho mejor cómo funciona el protocolo *libp2p* y su labor de permitir a los nodos descubrirse mutuamente, estableciendo una conexión directa entre ellos para así no depender de un servidor central ni de empresas externas. Además, gracias a la ingeniería inversa pude analizar el código objeto de este TFG y mediante la aplicación de los conocimientos aprendidos durante el Grado he sido capaz de poder realizar una investigación profunda mediante la lectura de artículos sobre el funcionamiento de *libp2p* y *JavaScript*.

En lo relativo a las posibles ampliaciones que se pueden realizar, después de conocer todo el proyecto aplicando ingeniería inversa, destacan:

- Integración de interfaz gráfica: Desarrollar una interfaz gráfica que simplifique el uso del sistema chat.
- Escalabilidad y redes distribuidas: Ampliación del proyecto para que no solo funcione de forma local, sino también a través de internet, permitiendo la comunicación de nodos dispersos.
- Soporte para múltiples sistemas operativos: Añadir soporte para sistemas operativos como pueden ser *Linux* o *MacOS* para llegar a un público más amplio.
- Más funcionalidades: Añadir más funcionalidades al chat como podría ser el envío de mensajes de voz, imágenes o videos.
- Servidor distribuido: Establecer un servidor que permita establecer la comunicación chat y que sea controlado por un grupo de personas de confianza.

8. BIBLIOGRAFÍA Y WEBGRAFÍA EMPLEADAS.

Boehm, B. (1984). Software Engineering Economics. *IEEE Transactions on Software Engineering*, SE-10(1), 4-21. https://doi.org/10.1109/tse.1984.5010193

Boehm, B., Abts, C., y Chulani, S. (2000). Software Development Cost Estimation Approaches—A survey. *Annals Of Software Engineering*, 10 (1/4), 177-205. https://doi.org/10.1023/a:1018991717352

Durán, A., y Bernárdez, B., (2002). Metodología para la Elicitación de Requisitos de Sistemas Software, Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla, Versión 2.3, p. 42. https://www.juntadeandalucia.es/servicios/madeja/node/931/download/metodologia_elicitacion_2_3.pdf.zip

Garita-González, G. (2014). Métodos analíticos y métricas de calidad del software. San José. Costa Rica: Editorial EUNED.

Garita-González, G., y Lizano-Madriz, F. (2018). Estimación de costo de software: Una propuesta de aplicación pedagógica de COCOMO. *Uniciencia*, 32(1), 118. https://doi.org/10.15359/ru.32-1.8

Servei de Salut de les Illes Balears. (2024, enero). Boletín núm. 111: Riesgos de las aplicaciones de mensajería instantánea. *Boletines de la Oficina de Seguridad*. https://www.ibsalut.es/es/profesionales/e-salud-tecnologias-de-la-informacion-y-la-comunicacion/seguridad-de-la-informacion/boletines-de-la-oficina-de-seguridad/4273-boletin-num-111-riesgos-de-las-aplicaciones-de-mensajeria-instantanea

WEBGRAFÍA.

Libp2p Documentation Portal (s. f.). *What is libp2p*. Libp2p. https://docs.libp2p.io/concepts/introduction/overview/

Martins, J. (2024). *Scrum: conceptos clave y cómo se aplica en la gestión de proyectos* [2024], Asana. https://asana.com/es/resources/what-is-scrum