



INFORME MECANISMOS DE COMUNICACIÓN – SOCKETS

INTEGRANTES: Marcos Pacheco – Nelson Ortega

Los mecanismos de comunicación sockets son una forma de establecer una conexión bidireccional entre dos procesos que se ejecutan en máquinas diferentes o en una misma máquina. Un socket es un identificador de la red de comunicaciones que permite enviar órdenes a un servidor encargado de escuchar y atender las diferentes peticiones. Los sockets están clasificados en:

- **Orientados a conexión (TCP):** También conocidos como socket de flujo, estos proporcionan un flujo de datos sin límites de registro, garantizando la entrega ordenada y no duplicada de los datos. El protocolo utilizado es el TCP, lo que le da esa fiabilidad en la llegada de los datos, ya que este protocolo se encarga de informar la llegada de los datos, así como el buen recibimiento de estos.
- **No orientados a conexión (UDP):** También llamados socket de datagrama, son orientados a registros, donde no se garantiza su entrega, orden o duplicidad. Este tipo de socket utiliza el protocolo UDP, por lo que son utilizados para transferencia de datos que sean no confiables y sin conexión, ya que los datos conservan los límites de registro de los datos, siempre que sean menor al límite del receptor. Un ejemplo de su uso es en transmisiones de tiempo real, ya que están aceptan cierto grado de pérdida.
- **De paquete secuencial:** Es una combinación de los sockets previamente dichos, ya que es orientada a la conexión, pero manteniendo un límite de paquetes delimitados.

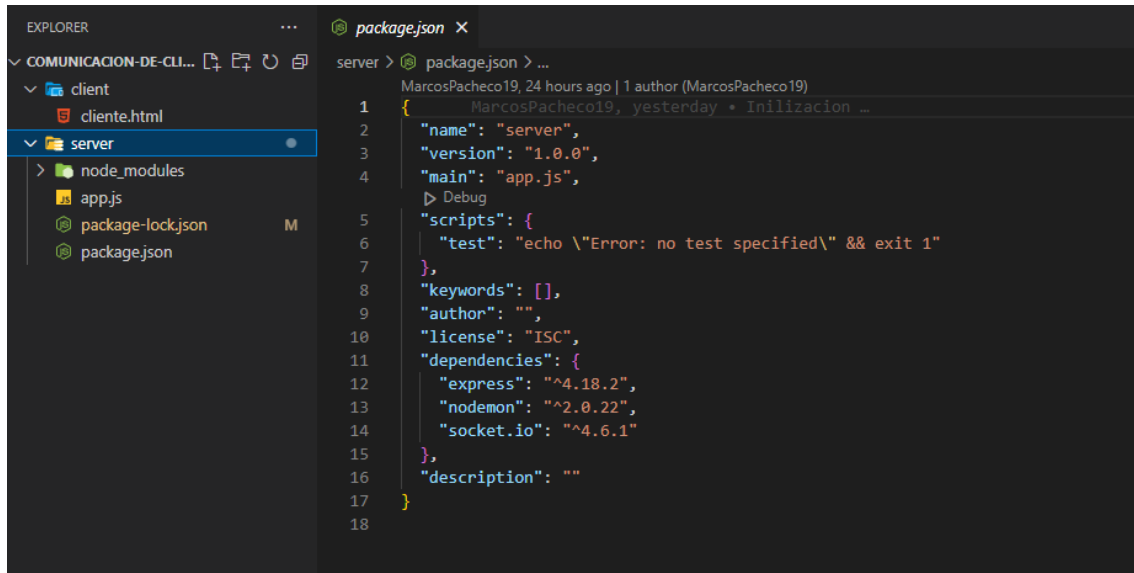
Se procede a desarrollar una aplicación de comunicación en tiempo real grupal tipo chat, con las siguientes indicaciones:

- a) Diseñar una arquitectura centralizada, de tipo cliente-servidor en la que varios clientes se conecten a un servidor central para enviar y recibir mensajes.
- b) Implementar sockets TCP en cada uno de los nodos del sistema para permitir la comunicación entre ellos.
- c) Implementar un protocolo de comunicación seguro que incluya elementos como la verificación de la integridad de los mensajes y/o encriptación de datos.
- d) Manejar las conexiones y desconexiones de los diferentes clientes de manera adecuada, permitiendo la reconexión automática en caso de que un cliente se desconecte de la red.

- e) Implementar una interfaz de usuario amigable que permita a los usuarios conectarse al servidor, enviar y recibir mensajes.

Por lo que creamos nuestro proyecto en donde tendrán dos carpetas: client y server.

En la carpeta de server instalamos los paquetes que vamos a utilizar los cuales son express, nodemon y socket.io.

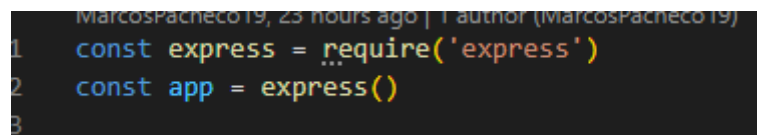


The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows a project structure with a 'server' folder containing 'app.js', 'package-lock.json', and 'package.json'. The 'package.json' file is open in the editor. The code in 'package.json' is as follows:

```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "app.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1"
7   },
8   "keywords": [],
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "express": "^4.18.2",
13    "nodemon": "^2.0.22",
14    "socket.io": "^4.6.1"
15  },
16  "description": ""
17 }
```

Luego la configuración del servidor se realiza en nuestro archivo app.js.

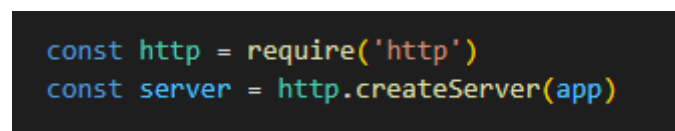
Primero procedemos a realizar la importación del módulo express para asignarla a una constante y así poder acceder a sus métodos.



The screenshot shows the first two lines of the 'app.js' file:

```
1 const express = require('express')
2 const app = express()
3
```

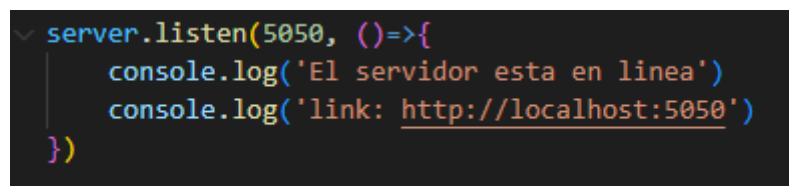
Para poder utilizar socket.io y express es necesario utilizar http, lo que nos permite crear un servidor.



The screenshot shows the code for importing the http module and creating a server:

```
const http = require('http')
const server = http.createServer(app)
```

Con el servidor creado, podemos empezar a configurarlo, como por ejemplo el asignar que puerto es el que va a escuchar, en nuestro caso el "5050".



The screenshot shows the 'server.listen' method call with a log message:

```
server.listen(5050, ()=>{
  console.log('El servidor esta en linea')
  console.log('link: http://localhost:5050')
})
```

Así como también unos pequeños mensajes en el log, al momento de que un cliente usuario se conecte o se desconecte, así como el mensaje que envían.

```

14  ✓ io.on('connection', (socket)=>{
15      console.log('Un usuario se ha conectado')
16
17  ✓  socket.on('disconnect', ()=>{
18      console.log('Un usuario se ha desconectado')
19  })
20
21  ✓  socket.on('chat', (msg)=>{
22      io.emit('chat', msg)
23  })
24  })
25

```

Y por último debemos indicar desde donde se nuestro servidor va a escuchar e interactuar con nuestros clientes.

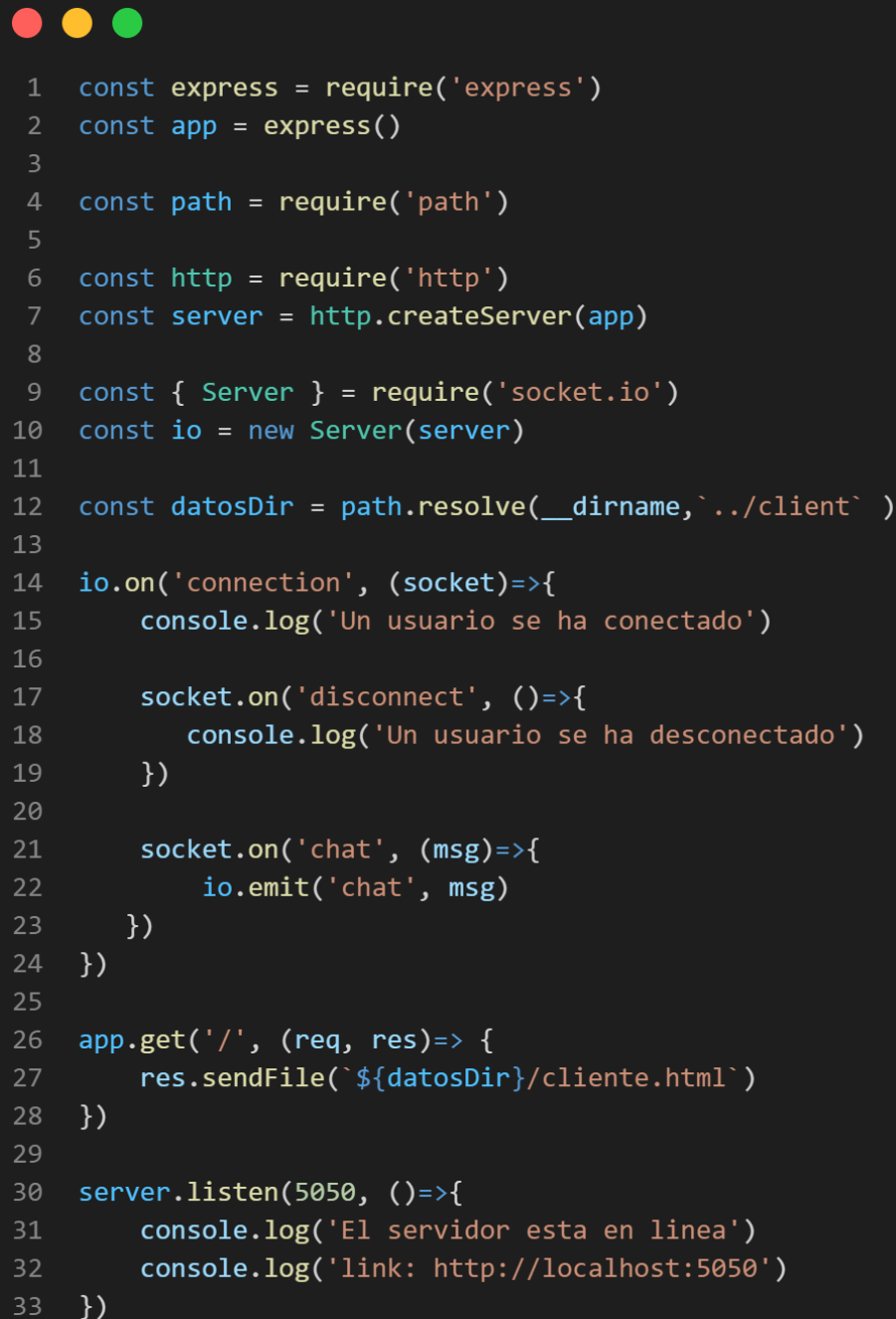
```
const datosDir = path.resolve(__dirname, `../client` )
```

```

app.get('/', (req, res)=> {
  res.sendFile(`${datosDir}/cliente.html`)
})

```

Por lo que desde el lado del servidor este sería nuestro código:



```
1  const express = require('express')
2  const app = express()
3
4  const path = require('path')
5
6  const http = require('http')
7  const server = http.createServer(app)
8
9  const { Server } = require('socket.io')
10 const io = new Server(server)
11
12 const datosDir = path.resolve(__dirname, `../client` )
13
14 io.on('connection', (socket)=>{
15     console.log('Un usuario se ha conectado')
16
17     socket.on('disconnect', ()=>{
18         console.log('Un usuario se ha desconectado')
19     })
20
21     socket.on('chat', (msg)=>{
22         io.emit('chat', msg)
23     })
24 })
25
26 app.get('/', (req, res)=> {
27     res.sendFile(`${datosDir}/cliente.html`)
28 })
29
30 server.listen(5050, ()=>{
31     console.log('El servidor esta en linea')
32     console.log('link: http://localhost:5050')
33 })
```

Ahora continuamos con la configuración desde el lado del cliente en nuestro archivo cliente.html.

En donde utilizamos de primeras utilizamos dos paquetes para el estilo de nuestro proyecto los cuales son bootstrap y font-awesome.

```

1 <link rel="stylesheet"
2   href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"
3   integrity="sha512-iecLmask17CVkqkXNQ/ZH/XLlVWZ0Jy7Yy7tceD1ypASozpmT/E0iPtmFIB46ZmdtAc9eNBvH0H/ZpiBw=="
4   crossorigin="anonymous"
5   referrerpolicy="no-referrer"/>
6 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
7   rel="stylesheet"
8   integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5Ily+dnN9+ndJOZ"
9   crossorigin="anonymous">

```

Y asignamos el estilo que va a tener nuestro chat, así como unos pequeños íconos que tendrá el chat.

```

1 <style>
2   body {
3     background: rgb(239, 234, 226);
4   }
5   form {
6     background: rgba(0, 0, 0, 0.15);
7     padding: 0.25rem;
8     position: fixed;
9     bottom: 0;
10    left: 0;
11    right: 0;
12    display: flex;
13    height: 3rem;
14    box-sizing: border-box;
15    backdrop-filter: blur(10px);
16  }
17
18  ul {
19    list-style-type: none;
20    margin: 0;
21    padding: 0;
22  }
23
24  ul > li {
25    padding: 0.5rem 1rem;
26  }
27
28  ul > li:nth-child(odd) {
29    background: #dcf8c6;
30    text-align: right;
31    font-style: italic;
32    font-weight: 600;
33    border-radius: 25px;
34  }
35
36  li i {
37    margin-right: 5px;
38    color: #333;
39  }
40 </style>

```

Lo siguiente que hacemos es crear un pequeño formulario que será el encargado de la recepción del mensaje por parte del usuario, así como un botón de envío.

```

<form action="">
  <input type="text" class="form-control" placeholder="Ingrese el mensaje">
  <button class="btn btn-primary">Enviar</button>
</form>

```

Por último, tenemos la configuración de nuestro código JS que se encarga de la creación del socket, su comunicación y el envío del evento hacia nuestro servidor.

```

1  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"
2    integrity="sha384-ENjd04Dr2bkBIFxQpoeTz1HICje39Wm4jDKdf19U8gI4ddQ3GYN57NTKfAdVQSZe"
3    crossorigin="anonymous"></script>
4  <script src="http://localhost:5050/socket.io/socket.io.js"></script>
5
6  <script>
7    let socket = io();
8    const form = document.querySelector('form');
9    const input = document.querySelector('input');
10   let mensajes = document.querySelector('ul');
11
12   form.addEventListener('submit', (e)=>{
13     e.preventDefault();
14     if (input.value) {
15       socket.emit('chat', input.value);
16       input.value = '';
17     }
18   });
19
20   socket.on('chat', (msg)=>{
21     let item = document.createElement('li');
22     item.innerHTML = '<i class="fa-sharp fa-solid fa-person fa-beat"></i>' + msg;
23     mensajes.appendChild(item);
24     window.scrollTo(0, document.body.scrollHeight);
25   });
26 </script>

```

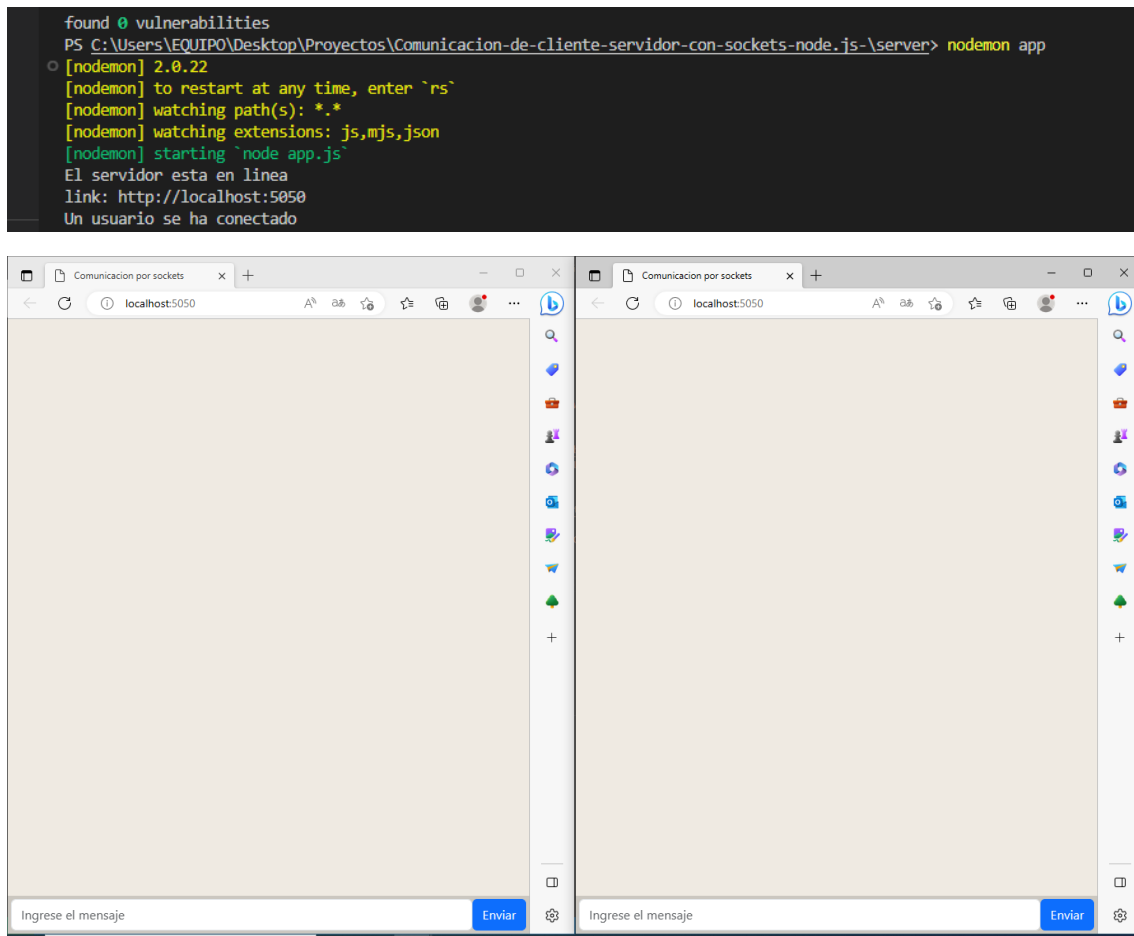
Por lo que nuestro código desde el lado del cliente quedaría así:

```

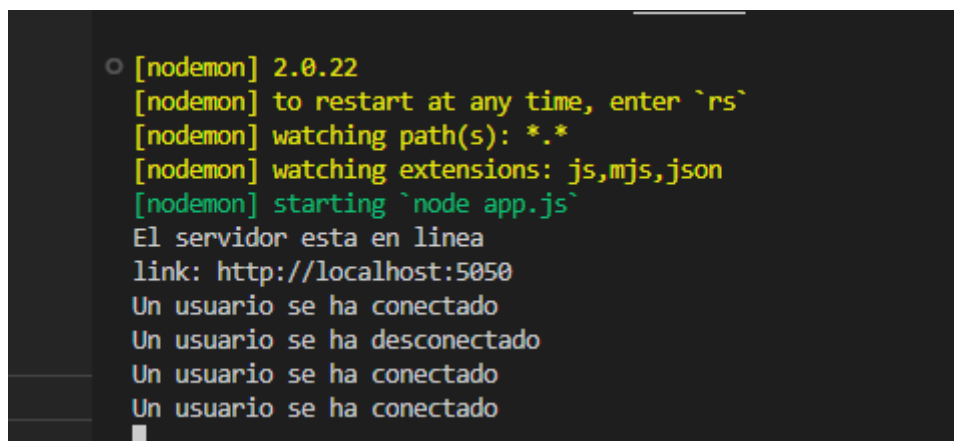
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1">
6    <title>Comunicacion por sockets</title>
7    <link rel="stylesheet"
8      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"
9      integrity="sha512-iecLmask17CVkqkXNQ/ZH/XLlvWZOJyj7Yy7tcenmpD1ypASozpmT/E0iPtmFIB46ZmdtAc9eNBvH0H/Zp1Bw=="
10     crossorigin="anonymous"
11     referrerpolicy="no-referrer"/>
12    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
13      rel="stylesheet"
14      integrity="sha384-KK94CHFLLe+nV2dmCWGMq91rCGa5gtU4mk92HdVYe+M/SXH301p5ILy+dN9+nJ0Z"
15      crossorigin="anonymous">
16  <style>
17    body {
18      background: rgb(239, 234, 226);
19    }
20    form {
21      background: rgba(0, 0, 0, 0.15);
22      padding: 0.25rem;
23      position: fixed;
24      bottom: 0;
25      left: 0;
26      right: 0;
27      display: flex;
28      height: 3rem;
29      box-sizing: border-box;
30      backdrop-filter: blur(10px);
31    }
32
33    ul {
34      list-style-type: none;
35      margin: 0;
36      padding: 0;
37    }
38
39    ul > li {
40      padding: 0.5rem 1rem;
41    }
42
43    ul > li:nth-child(odd) {
44      background: #dcf8c6;
45      text-align: right;
46      font-style: italic;
47      font-weight: 600;
48      border-radius: 25px;
49    }
50
51    li i {
52      margin-right: 5px;
53      color: #333;
54    }
55  </style>
56 </head>
57 <body>
58
59   <ul class="listado"></ul>
60
61   <form action="">
62     <input type="text" class="form-control" placeholder="Ingrese el mensaje">
63     <button class="btn btn-primary">Enviar</button>
64   </form>
65   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"
66     integrity="sha384-ENjd04Dr2bkBIFxQpoeTz1Hicje39Wm4jDKdf19U8gI4ddQ3GYNs7NTKfAdVQSZe"
67     crossorigin="anonymous"></script>
68   <script src="http://localhost:5050/socket.io/socket.io.js"></script>
69
70   <script>
71     let socket = io();
72     const form = document.querySelector('form');
73     const input = document.querySelector('input');
74     let mensajes = document.querySelector('ul');
75
76     form.addEventListener('submit', (e)=>{
77       e.preventDefault();
78       if (input.value) {
79         socket.emit('chat', input.value);
80         input.value = '';
81       }
82     });
83
84     socket.on('chat', (msg)=>{
85       let item = document.createElement('li');
86       item.innerHTML = '<i class="fa-sharp fa-solid fa-person fa-beat"></i>' + msg;
87       mensajes.appendChild(item);
88       window.scrollTo(0, document.body.scrollHeight);
89     });
90   </script>
91 </body>
92 </html>

```

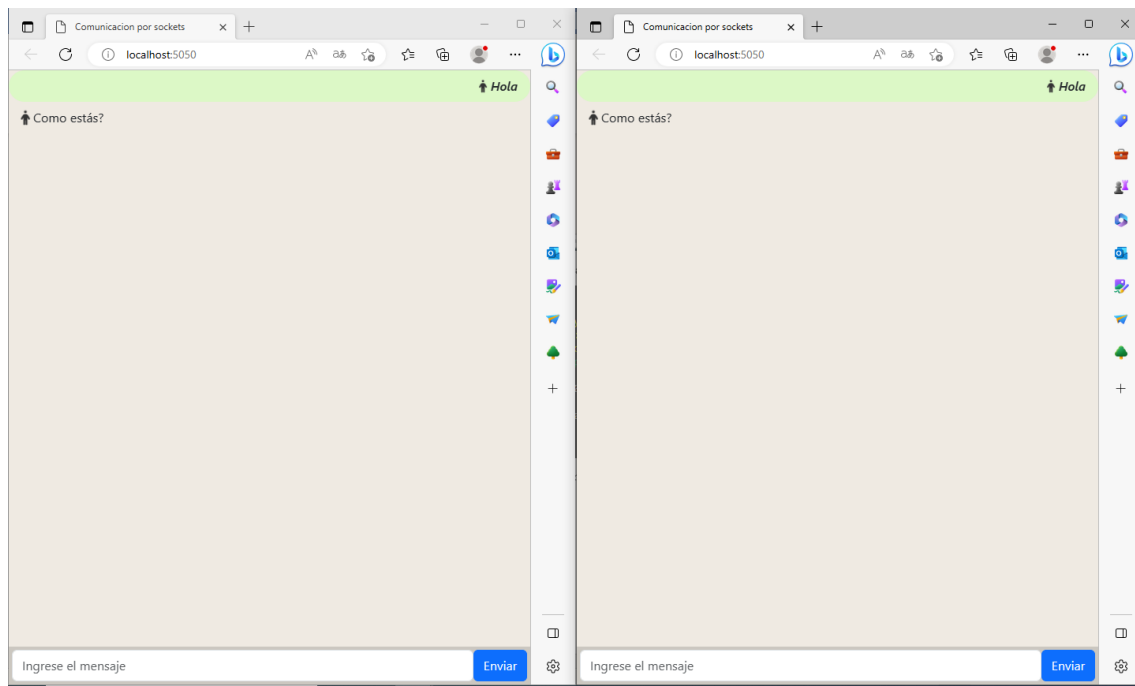
Ya con nuestro servidor y cliente configurados, podemos pasar a comprobar su funcionamiento al correr nuestro proyecto utilizando nodemon.app.



Podemos observar en nuestro terminal en el caso de que se conecte o desconecte.



Por último, así se vería nuestro chat al momento de enviar mensajes.



Conclusión:

Con esta práctica se ha podido realizar un acercamiento a lo que son los sockets, y a sus diferentes usos para la comunicación en sistemas distribuidos. Al reconocimiento de los diferentes tipos que existen mediante la investigación realizada, así como la configuración de hojas de estilo para dar una apariencia más amigable a nuestro chat.

Referencias:

- [1] IBM. (n.d.). Socket types. IBM Knowledge Center.
<https://www.ibm.com/docs/es/aix/7.3?topic=protocols-socket-types>
- [2] Socket.IO. (n.d.). Socket.IO: The fastest and most reliable real-time engine.
<https://socket.io/docs/v4/>