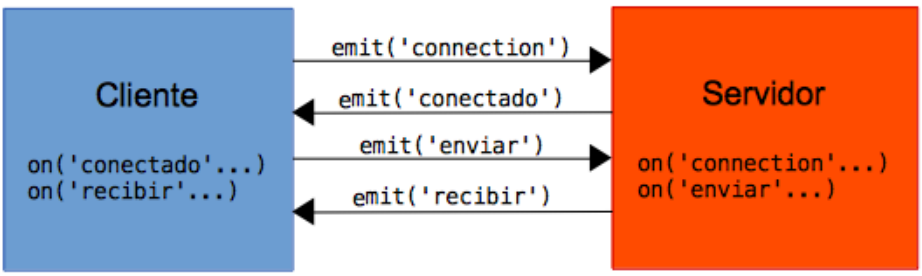
Comunicació en temps real amb *socket.io*

Socket.io és una llibreria emprada per a comunicacions i gestió d'esdeveniments en temps real entre client i servidor. Esta llibreria s'empra en tots dos costats de la comunicació (client i servidor) per a crear un canal de comunicació entre tots dos denominat *socket* (o més concretament, *web socket*).

La principal utilitat d'esta llibreria consistix a poder emetre i detectar esdeveniments des dels dos costats de la comunicació, i poder enviar dades d'un costat a un altre en qualsevol moment. Així, un esdeveniment emés pel client, per exemple, pot rebre's i tractar-se en el servidor, i viceversa.



D'esta manera, es trenca el tradicional esquema de comunicació HTTP, on és el client qui ha de sol·licitar un recurs al servidor perquè este li responga. Amb *socket.io*, una vegada establida la connexió client-servidor, el servidor pot tindre la iniciativa d'enviar al client dades en qualsevol moment, sense que este els haja sol·licitats.

Al llarg d'esta secció, desenvoluparem un xicotet xat utilitzant *socket.io*. Hem de començar per descarregar els recursos corresponents. És un arxiu ZIP que, en descomprimir-lo, crea una carpeta anomenada "*ChatSocketIO*", que hauràs de copiar en el teu espai de treball "*ProjectesNode/Exercicis*". Sobre esta base de projecte afegirem els canvis necessaris per a construir el xat.

1. Descàrrega i instal·lació

Socket.io és un mòdul que pot utilitzar-se per si mateix en una aplicació Node, o integrat en Express, depenent del que vulguem fer. En qualsevol dels dos casos, és necessari descarregar amb `npm install` el corresponent mòdul (*socket.io*), i fer el corresponent `npm init` previ per a preparar l'arxiu `package.json`, si no s'ha fet ja prèviament.

```
npm install socket.io
```

En el nostre cas, per al projecte "*ChatSocketIO*" que hauries d'haver descarregat seguint els passos previs, ja tenim configurat l'arxiu `package.json` amb tot el necessari, per la qual cosa només caldrà executar un

comando `npm install` per a instal·lar les dependències de mòduls (*Express* i *socket.io*, a més del motor de plantilles Nunjucks, en el nostre cas).

1.1. Ús en client i servidor

Com hem comentat, *socket.io* és una llibreria que s'empra tant en client com en servidor per a comunicar totes dues parts. Del costat del servidor no hi ha problema, ja que s'emprarà la llibreria des de la carpeta corresponent en `node_modules`, però també serà necessari instal·lar-ho dins del contingut estàtic de la part del client. Per a això, podem descarregar la llibreria solta d'Internet (arxiu `socket.io.js`), o podem localitzar este mateix arxiu dins de *node_modules/socket.io/client-dist* i copiar-lo i pegar-lo en la nostra carpeta de contingut estàtic. Este pas ja el tens dau en el projecte que t'has descarregat (veuràs l'arxiu `socket.io.js` en la carpeta "*public/js*").

1.2. Estructura del projecte de prova

En el projecte descarregat per a este apartat veureu els següents continguts:

- Carpeta "*public*", amb tot el contingut estàtic que necessitarem:
  - Arxiu `css/estilos.css` amb uns estils bàsics perquè la web quede una mica més "vistosa"
  - Arxiu `js/socket.io.js` amb la llibreria client per a *socket.io*, ja preparada
  - Arxiu `js/chat.js` on col·locarem el codi per a connectar amb el servidor des del client i enviar i rebre informació mitjançant *socket.io*. Inicialment, este arxiu està buit.
- Carpeta "*views*" amb una única vista, `index.njk`, ja completa per a mostrar el formulari de xat. No serà necessari tocar res en esta vista per a este projecte.
- Arxiu principal `app.js`, també buit, on afegirem el codi per a posar en marxa el servidor, connectar amb el client i intercanviar-se informació.

2. Enllaçar client i servidor

Ara que ja tenim el projecte preparat, escriurem les primeres línies de codi per a enllaçar client i servidor

2.1. Connectar des del costat del client

En primer lloc (encara que no té per què ser així), editarem l'arxiu del client (situat en `public/js/chat.js`). Les següents línies permeten crear una instància de *socket.io* i connectar amb el servidor en la URL especificada, que serà *localhost* en el nostre exemple, però podria ser una direcció remota:

```
let socket = io('http://localhost:8080');
alert("Connectat");
```

Òbviament, la segona línia de codi no és necessària, però ens servirà per a mostrar un missatge en pantalla quan s'establisca la connexió.

2.2. Connectar des del costat del servidor

Anem ara al costat del servidor (arxive `app.js` de la carpeta principal del projecte), i afegim estes línies per a establir una connexió amb el client. Vegem les línies pas a pas:

1. Les primeres línies inclouen les llibreries necessàries (*http*, *express* i *socket.io*, a més del motor de plantilles *nunjucks*):

```
const express = require('express');
const http = require('http');
const socketio = require('socket.io');
const nunjucks = require('nunjucks');
```

2. Les següents línies inicialitzen l'aplicació Express i connecten *socket.io* amb el servidor Express.

```
let app = express();
let server = http.createServer(app);
let io = socketio(server);
```

> Notar que emprem el mòdul *http* com a intermediari entre *socket.io* i Express, per a poder comunicar una cosa amb una altra. Este mòdul és propi del nucli de Node, per la qual cosa no és necessari instal·lar-lo amb `npm install`.

3. Després, inicialitzem el middleware necessari: motor de plantilles per a carregar el formulari de xat, i processament de contingut estàtic:

```
nunjucks.configure('views', {
    autoescape: true,
    express: app
});

app.set('view engine', 'njk');
app.use(express.static('./public'));
```

4. Després, definim una ruta bàsica per a mostrar la pàgina principal (el formulari de xat):

```
app.get('/', (req, res) => {
    res.render('index');
});
```

5. Posem a escoltar el servidor...

```
server.listen(8080);
```

6. ... i, finalment, definim un esdeveniment anomenat "*connection*" (predefinit en *socket.io*) perquè, quan li arribe una connexió d'un client, faça "alguna cosa" (en este cas, de moment, ens limitarem a mostrar per consola que s'ha connectat un client).

```
io.on('connection', (socket) => {
    console.log("Client connectat");
});
```

Dins d'este últim codi (de l'esdeveniment que hem definit) serà on processarem tot el que ens arribe del client en qüestió, a través del socket de comunicació que s'ha creat entre totes dues parts.

En este punt, podem posar en marxa el servidor Express i accedir a la URL:

```
http://localhost:8080
```

Veurem el missatge `alert` confirmant la connexió en el client, i després el formulari de xat (encara que de moment no hi ha botó per a enviar textos).

3. Definició, emissió i captura d'esdeveniments

Ara que ja hem connectat client i servidor, vegem el mecanisme per a emetre i capturar esdeveniments entre client i servidor:

- Per a **capturar** esdeveniments, utilitzarem el mètode `on`, passant-li com a paràmetres el nom de l'esdeveniment en qüestió, i una funció *callback* de resposta. És el que hem fet ja en l'aplicació servidor, per a detectar la connexió del client a través de l'esdeveniment "*connection*":
- Per a **emetre** esdeveniments i que es puguen capturar a continuació, s'utilitza el mètode `emit`, passant-li com a paràmetre l'esdeveniment a emetre.

3.1. Primera prova: confirmació de la connexió

Per a veure millor com funciona esta seqüència d'emissió i captura d'esdeveniments, aprofitarem l'esdeveniment `on("connection")` que ja tenim definit en el servidor `app.js`. Dins del codi del callback, en lloc de traure per consola un missatge, emetrem un esdeveniment (que anomenarem, per exemple, "connectat"), perquè el puga capturar el client:

```
io.on('connection', (socket) => {
    socket.emit('connectat');
});
```

Després, en el codi del client (`public/js/chat.js`), definim un mètode `on` per a capturar l'esdeveniment "connectat", i traslladem la instrucció `alert` dins del callback d'este esdeveniment, perquè es mostre en confirmar la connexió. De pas, aprofitem este esdeveniment per fer visible la part del formulari que estava oculta, per a enviar missatges:

```
let socket = io('http://localhost:8080');

socket.on('connectat', function() {
    alert("Connectat");
    document.getElementById('formulario').style.display = "";
});
```

Si executem ara el servidor i accedim a la URL principal, veurem el missatge de confirmació i el formulari de xat, ja complet:

3.2. Esdeveniments per al xat

Definirem ara els esdeveniments i respostes restants perquè el xat puga funcionar:

Començarem per emetre un esdeveniment en el client que s'active cada vegada que premem el botó d'enviar del formulari de xat. Si et fixes en el codi de la plantilla `index.njk` en la carpeta de vistes, veuràs que en prémer el botó d'Enviar es diu a una funció JavaScript anomenada `enviar` , que encara no hem implementat. La implementem en el codi del client (`chat.js`):

```
function enviar() {
    let nick = document.getElementById('nick').value;
    let text = document.getElementById('texto').value;
    if (text != "" && nick != "")
        socket.emit('enviar', {nick: nick, text: text});
}
```

Observeu que, al costat de l'esdeveniment "enviar", s'envia un objecte JavaScript amb el nick de l'usuari que envia el missatge, i el missatge en si. Este objecte es recollirà en el servidor amb eixe format.

En el servidor, recollim este esdeveniment "enviar" amb el corresponent `on` (dins del codi de l'esdeveniment "connected"), i el que farem serà emetre un nou esdeveniment, que hem anomenat "difondre", perquè el reben els clients:

```
io.on('connection', (socket) => {
    socket.emit('connectat');
    socket.on('enviar', (dades) => {
        io.emit('difondre', dades);
    });
});
```

Sobre este pas, hi ha algunes consideracions a tindre en compte: La dada que es recull i envia és l'objecte JavaScript que es va emetre en enviar el missatge, amb el nick i el missatge a enviar. Observa que, si des del servidor volem emetre un esdeveniment només al client que el seu socket estem tractant, usem `socket.emit`, mentre que si volem difondre l'esdeveniment a TOTS els clients connectats, emprarem `io.emit`.

Finalment, en el client "*chat.js*" recollim l'esdeveniment "difondre" emés pel servidor, i vam mostrar en l'àrea de xat el missatge que ens arriba:

```
socket.on('difondre', function (dades) {
    let xat = document.getElementById('chat');
    xat.innerHTML += '<p><em>' + dades.nick + '</em>: ' +
        dades.text + '</p>';
});
```

Ara ja tenim el xat operatiu. Podem posar en marxa el servidor, i obrir un parell de navegadors (o de pestanyes en el mateix navegador), cadascuna amb un nick diferent, i provar d'enviar missatges entre elles:

4. Altres opcions de socket.io

El vist fins ara ens servix com a exemple bàsic de les possibilitats que oferix la llibreria. Vegem a continuació breument algunes de les opcions addicionals que existixen.

4.1. L'esdeveniment de desconnexió

Existix un esdeveniment, també predefinit en *socket.io*, anomenat "disconnect", que podem emprar tant en el client com en el servidor per a donar resposta al fet que una de les dues parts es desconnecte (per exemple, que caiga el servidor).

```
socket.on('disconnect', () => { ... });
```

4.2. Reexpedir missatges a tots menys a l'emissor original

Hem vist dos mecanismes d'enviament d'esdeveniments en socket.io:

- `socket.emit` per a emetre un esdeveniment únicament al receptor a l'altre costat del socket
- `io.emit`, utilitzat des del servidor, per a difondre un esdeveniment a tots els clients connectats.

Existix una tercera alternativa, que consistix a utilitzar `socket.broadcast.emit`, per a emetre l'esdeveniment a tots els clients connectats, excepte al qual està a l'altre costat del socket actual.

4.3. Agrupar comunicacions per grups o sales

En el cas que vulguem separar les converses dins d'una mateixa URL o aplicació per grups o sales, de manera que un client només pot comunicar-se i veure les comunicacions del grup al qual pertany, hem d'associar al client que es connecta a un grup, mitjançant el mètode join:

```
io.on('connection', (socket) => {
    ...
    socket.join('nom_grup');
    ...
```

Pot ser el propi client el que trie el grup al qual connectar-se mitjançant un formulari, o el servidor qui li assigne un grup, aleatòria o intencionadament. També podem eixir d'un grup amb el mètode leave:

```
socket.leave('nom_grup');
```

Finalment, a l'hora d'enviar missatges únicament als membres d'eixe grup, tenim disponible el mètode `to`, per a intercalar abans de l'anomenada a emit:

```
io.to('nom_grup').emit(...);
```

4.4. Altres usos de *socket.io*

Se'ns queden diverses coses en el tinter en esta introducció a *socket.io*, com podreu imaginar. En general, podem emprar esta llibreria per a qualsevol tipus d'aplicació en temps real, com per exemple:

- Gestió d'e-mails, on el servidor notifica als clients tan prompte com un e-mail nou arriba, i els clients poden enviar e-mails al servidor perquè notifique als destinataris.
- Videojocs on els clients envien situació actualitzada al servidor, i este la notifique o difonga a la resta de jugadors.
- ... etc.