

# HTML

**Favicon:** <link rel="icon" type="image/x-icon" href="/images/favicon.ico">

## Listas:

*ordered list* <ol>

```
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
</ol>
```

*unordered list*

<ul>

```
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
```

</ul>

*description list*

<dl>

```
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
```

</dl>

**Tablas** <table </table>

<table>

```
<tr> -> table row (especificamos la primera fila)
      <th>Month</th> -> table header (cabecera)
      <th>Savings</th>
```

</tr>

```
<tr> -> table row (especificamos la segunda fila)
      <td>January</td> -> table data (datos)
      <td>$100</td>
```

</tr>

</table>

**Month Savings**  
January \$100

**Enlaces** <a href="URL">Texto del enlace</a> -> **USAR RUTAS RELATIVAS**

**href:** para enlazar al element con id id\_name

<a href="URL" target="target">Texto del enlace</a>

**target:** indica dónde aparece la URL enlazada

<a href="/images/imagen.jpg" download="nombre\_imagen.jpg">

**download:** el navegador trata la URL como una descarga

**Span** <span> </span> -> contenedor de línea. Remarcar parte de un texto o documento (como el nombre de nuestra aplicación, que está una parte en azul y otra en verde)

**Input** (para darle al submit en los formularios): <input> **VOID element**

[<input>: The HTML Input element - HTML: HyperText Markup Language | MDN](#)

- Text: `<input type="text" placeholder="Introduce el texto">`
- Radio: `<input type="radio">`
- Checkbox: `<input type="checkbox">`
  - [Se puede hacer hasta arte](#)
- Button: `<input type="button">` **NO**
- Password: `<input type="password">`
- Hidden: `<input type="hidden">` **OCULTAR**
- Números: `<input type="number">`

`<input required ...>` este campo es obligatorio

- `<input minlength="5" ...>` no puede ser de menos de “x” caracteres
- `<input maxlength="50" ...>` no puede ser de más de “x” caracteres

### Formulario para seleccionar distintas opciones:

`<label for="cars">Choose a car:</label>`

`<select name="cars" id="cars">`

`<option value="volvo">Volvo</option>` // **value** es el dato que se envía al servidor

`<option value="saab">Saab</option>`

`</select>`

### Formulario:

`<form action="/action_page.php" method="get">`

`<label>First name: <input type="text" name="fname"></label><br>`

`<label for="lastname">Last name: </label>`

`<input type="text" name="lname" id="lastname"><br>`

`<input type="submit" value="Submit">`

`</form>`

**Action:** especificar la ruta, debe ser IGUAL en app.js/server.js. OJO!! En routes, mientras sea distinto método (distinto get a post), poner en action: ‘/’ -> así solamente.

**Method:** get, con el que se envían datos en la cabecera. Post, con el que se envían datos en el cuerpo.

### Estructura del contenido:

`<article>` Composición en un documento (post, comentario...)

`<footer>` Pie de página con información del autor, copyright y otros

`<nav>` Enlaces de navegación como menús o tablas de contenido

`<header>` Cabecera de la página

`<section>` Sección de un documento

`<aside>` Menú lateral



# CSS

Selectores:

1. **Simple**: basados en el **nombre** de un elemento. Sintaxis: `nombre_elemento {}`  
basados en el atributo **id** del elemento (**ÚNICO!**). Sintaxis: `#nombre_id {}`  
basados en el atributo **class** de un elemento (**PUEDE repetirse**).  
Sintaxis: `.nombre_class {}` -> **¡OJO!** Va por orden entonces coge el último.  
(en caso de que todo sea igual, se aplica el **ÚLTIMO** -> tiene prioridad el último archivo del HTML y el último estilo con el mismo nombre en CSS.  
**Specificity: id > class > element** ).  
Se puede **especificar** a un **tipo determinado** de **elementos**:  
`nombre_elemento.nombre_class {}`  
selector **universal**. Selecciona **TODOS** elementos del HTML. Sintaxis: `* {}`
2. **Pseudoclases**: definen un estado especial de un elemento.

`:hover` → Selecciona al elemento que tiene el ratón encima  
`:focus` → Selecciona al elemento que tiene el foco  
`:first-child` → Selecciona al primer hijo  
`:last-child` → Selecciona al último hijo

: identificar una parte especial de un elemento (**selector : : pseudo-element {}**)

`::after` → Para insertar algo antes. Usar con la propiedad `content`  
`::before` → Para insertar algo después. Usar con `content`  
`::first-letter` → Selecciona la primera letra  
`::first-line` → Selecciona la primera línea  
`::selection` → La selección del usuario

3. **Otros**:
  - a. `[attribute~="value"]` -> contiene una **palabra específica**.
  - b. `[attribute|=“value”]` -> empieza por una **palabra concreta**.
  - c. `[attribute^="value"]` -> empieza de una **forma concreta**.
  - d. `[attribute$="value"]` -> termina de una **forma concreta**.
  - e. `[attribute*="value"]` -> contiene un **valor específico**.

Declaraciones: **Modelo de caja**.

- **Contenido**: El contenido de la caja como texto o imágenes
- **Padding**: Área vacía alrededor del contenido. Es transparente
- **Borde**: Borde alrededor del padding y del contenido
- **Margen**: Área vacía fuera del borde. Es transparente



- **Margin**: Espacio alrededor del elemento por fuera. *margin-top, margin-right, margin-bottom, margin-left*. Pueden ser valores negativos.

- **Padding:** Espacio alrededor del contenido del elemento y dentro de los bordes. *padding-top, padding-right, padding-bottom, padding-left* . NO valores negativos.
- **Ancho (width) y Altura (height):** *max-width, max-height, min-width, min-height*. Máximo / mínimo ancho y alto del contenido. *none*: no hay.

Declaraciones: **Text**.

- `color`
- `background-color`
- `text-align: left/right/center/justify;`
- `direction`
- `vertical-align`
- `text-decoration`
- `text-transform: uppercase/lowercase/capitalize;`
- `text-indent: Sangrado de la primera línea`
- `letter-spacing`
- `line-height`
- `word-spacing`
- `white-space`
- `text-shadow`

Declaraciones: **Font**.

- `font-family: "Times New Roman", Times, serif;`
- `font-style: normal/italic/oblique;`
- `font-weight: normal/bold;`
- `font-variant: normal/small-caps;`
- `font-size`
- `font: font-style font-variant font-weight font-size/line-height font-family;`

Declaraciones: **Position**. -> **tipo de posicionamiento** usado para el elemento.

- `static`
  - Colocado según el **flujo normal de la página**. Por defecto.
- `relative`
  - Relativo a su posición por defecto.
- `fixed`
  - Relativo al **viewport**, no se mueve al hacer scroll.
- `absolute`
  - Relativo al **ancestro más cercano**
- **Valores (cont.):**
  - `sticky`
    - Se queda "pegado" al hacer scroll
- **Para posicionar usar además:**
  - `top`
  - `right`
  - `bottom`
  - `left`
  - `z-index`

[CSS Layout - The position Property](#)

Valores: **Colores**. [HTML Color Names](#) -> sino poner el nombre y ya se elige sólo.

Recomendaciones de diseño: **Usar 12 columnas** -> Se puede dividir en columnas de 1, 2, 3, 4 y 6. *Full width: 12 columnas || Half width: 6 columnas || Third width: 4 columnas || Quarter width: 3 columnas.*

**CSS Cheat Sheet:** [CSS Cheat Sheet - Interactive, not a PDF | HTMLCheatSheet.com](#)

**Grid:** **2D – Divide la página en regiones**. Permite mucha versatilidad. Declarar un contenedor (*container*) como **display: grid;**

**Flexbox:** **1D – Filas o columnas**. Facilitar el layout, alineamiento y distribución de elementos. Alterar el tamaño de los elementos para rellenar el espacio. Declarar un contenedor (*container*) como **display: flex;**

# ¿CÓMO IMPLEMENTAR SOCKET.IO?

## app.js

```
const http = require('http');
const { Server } = require("socket.io");
//rutas -> todas las que sean van aquí
const app = express();
//para que funcione socket.io
const server = http.createServer(app);
const io = new Server(server);

//con esta parte, añado puerto!
const port = process.env.PORT || 3050; //aquí cambiamos por el puerto que queremos
server.listen(port, () => {
  console.log(`Servidor ejecutándose en http://localhost:${port}`);
});

// Inicialización de socket.io en el servidor
io.on('connection', (socket) => {
  console.log('User connected');
  //escucho mensajes enviados por cliente (recibidos)
  socket.on('chat', (data) => {
    console.log("Mensaje recibido del cliente " + data.user);
    io.emit('chat', { user: data.user, message: data.message }); //voy a enviarle a todos los
    usuarios el msj que me ha llegado
  });
  socket.on('disconnect', () => {
    console.log('Usuario desconectado');
  });
}); //socket.emit -> envio msjs ; socket.on -> recibir / enviar msjs (defino el callback
para ambas)

//view engine setup
//app.use y todo lo de la sesión + rutas
//used to: manejar mensajes y errores temporales para mostrarlos en las vistas
renderizadas
//app.use -> de errores y esas cosas
modules.exports = app;
```

## /views/chat.ejs

```
<%- include("header", {}) %>
<!-- Incluir etiquetas link específicas de esta view -->
<link rel='stylesheet' href='/stylesheets/chat.css'>
<script src="/socket.io/socket.io.js"></script>
<script src="/javascripts/chat.js" defer></script>
<%- include("nav", {}) %>
<div class="container">
  <h1><%= title %></h1>
  <p>Bienvenido a <%= title %></p>
```

```

    <p>Aquí podrá escribir mensajes con otros usuarios de nuestra plataforma.</p>
</div>
<div id="main">
    <ul id="messages" class="messages"></ul>
    <form id="form">
        <input id="input">
        <button>Send</button>
    </form>
    <!-- Pasar el nombre del usuario a través de un atributo data-* -->
    <div id="user" data-username="<%= user.username %>"></div>
</div>
<%- include("footer", {}) %>

```

**/routes/chat.js** -> aquí con copiar la plantilla de siempre, basta

```

const express = require('express');
const router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
    res.render('chat', {user:req.session.user, cookies: req.session.user ?
    req.session.user.cookiesAccepted : false, title:"Embutidos León"});
});

module.exports = router;

```

**/public/javascripts/chat.js**

```

//AQUÍ SE INTRODUCE LA PARTE DEL "CLIENTE"
const username = document.getElementById('user').getAttribute('data-username');
const socket = io();

const form = document.getElementById('form');
const input = document.getElementById('input');
const messages = document.getElementById('messages');

form.addEventListener('submit', (e)=> {
    e.preventDefault();
    if(input.value){
        socket.emit('chat', { message: input.value, user: username }); //así añadido tb el nombre
        del usuario que lo manda
        input.value = "";
    }
});

socket.on('chat', (msg) => {
    console.log("Mensaje recibido"); //hago un callback
    const item = document.createElement("li"); //me creo la lista para introducir los
    mensajes
    item.textContent = msg.user + ": " + msg.message; //muestro el nombre del usuario
    junto con el mensaje

```

```
messages.appendChild(item); //y los añado
});
```

```
//socket.emit -> envio msjs ; socket.on -> recibir / enviar msjs (defino el callback para ambas)
```

## **~AÑADIR CARPETA bin – ARCHIVO www**

### **bin/www**

```
#!/usr/bin/env node
/**
 * Module dependencies.
 */
var app = require('..../app');
var debug = require('debug')('expresslogin:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */
var port = normalizePort(process.env.PORT || '3000'); //aquí me dice, o lo inicializas en la
var PORT (definida fuera), o en el puerto 3000
app.set('port', port);

/**
 * Create HTTP server.
 */
var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */
function normalizePort(val) {
  var port = parseInt(val, 10);
  if (isNaN(port)) {
    // named pipe
    return val;
  }
  if (port >= 0) {
    // port number
    return port;
  }
  return false;
}
```

```

/**
 * Event listener for HTTP server "error" event.
 */
function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }
  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;
  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */
function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}

```



## ¿CÓMO DIFERENCIAR ROLES?

1º - **En base de datos** es donde le vamos a decir a cada uno que role tiene. Añadimos un parámetro a la hora de logearse, en el ejemplo que yo hice se añade por defecto “user” y luego registro a un admin a mano. En cambio, se puede hacer un selector que coja el valor en el register y se lo añada en base de datos (como en el *ejemplo de simulacro\_2023*).

2º - **En el login**, debo obtener el role de la “sesión”. Como lo tengo en base de datos, pues hago una función que sea `getParams` (copiarla de *ejemplo de simulacro\_2024*). Para obtenerlo de una manera correcta, hacerlo en dos pasos:

```
const user = req.body.user;
if(await database.user.isLoginRight(user, req.body.pass)){
  const userParams = database.user.getParams(user); el role
  req.session.role = userParams.role; }
```

3º - Como ya tengo el role en `req.session.role`, ahora puedo **añadirlo en nav/navBar/header** (como se llame mi barra de navegación), y poner condiciones según quien sea, pues que le aparezcan unas u otras opciones. Esto lo hago así:

```
<% if (user) { %> <!-- Si el usuario está autenticado -->
  <% if (role === 'admin') { %> <!-- Opciones para admin -->
    <li><a href="/adminPannel">Administrator</a></li>
  <% } %>
  <% if (role === 'user') { %> <!-- Opciones para usuario normal -->
    <li><a href="/restricted">Restringido</a></li>
    <li><a href="/chat">Chat</a></li>
  <% } %>
  <li class="right"><a href="/logout">Logout</a></li>
} else { %> <!-- Si no está autenticado -->
  <li class="right"><a href="/login">Login</a></li>
} %>
```

4º - Por último, en el **middleware de la sesión** (en `app.js – app.use(req,res,next => {})`); -> suele ser la que tiene lo de los mensajes y tal), la creo sino existe, sino añado lo siguiente: (de esta manera se lo añado en local para que pueda usarlo con sesión)

```
if (req.session.user) {
  res.locals.user = req.session.user; // Asigna el usuario completo
  res.locals.role = req.session.role; // Asigna el rol (debe obtenerse durante el login)
} else {
  res.locals.user = null;
  res.locals.role = null;
}
```