



**¡Les damos la  
bienvenida!**

¿Comenzamos?

Esta clase va a ser

- grabada

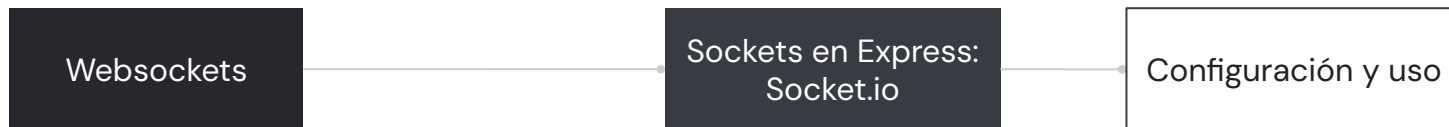
**Clase 10.** DESARROLLO AVANZADO DE BACKEND

# Websockets

# Objetivos de la clase

- Comprender la diferencia entre HTTP y Websockets
- Configurar un servidor websocket dentro de nuestro servidor express.
- Integrar websocket en nuestras plantillas

## MAPA DE CONCEPTOS



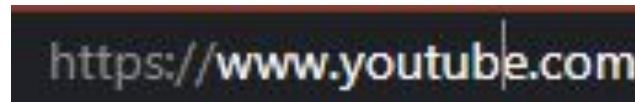
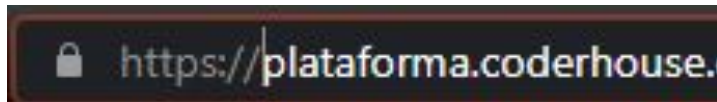
# Websockets

# WebSocket

WebSocket es un protocolo de comunicación basado en TCP para poder establecer esa conexión entre el cliente y el servidor, justo como sabemos, es el mismo objetivo que cubre HTTP.

Cuando llegamos a este punto, seguramente nos planteamos:

- ✓ ¿Por qué es necesario aprender otro protocolo?
- ✓ ¿Cuándo debo considerar un protocolo y cuándo otro?
- ✓ ¿Sirve realmente el protocolo websocket? En todas las páginas web veo http o https, nunca websocket



# ¿Qué hace que websocket se destaque?

A pesar de que websocket y HTTP son protocolos como lo mencionamos anteriormente, websocket tiene una característica muy importante: Su protocolo TCP establece dos endpoints de comunicación, **a cada endpoint se le conoce como socket.**

El contar con estos dos sockets permitirá establecer una **comunicación bidireccional** entre el cliente y el servidor. La comunicación bidireccional implica:

- ✓ Que el cliente puede obtener recursos del servidor cuando lo solicite (como en HTTP)
- ✓ Que el servidor pueda entregar información al cliente sin necesidad de que el cliente haga una petición.



# Caso práctico: Sistema de subastas en línea



# El gran problema de las subastas en línea

Los negocios por internet son algo de lo más común, sin embargo, no toda la compra-venta de productos se da de la manera habitual del carrito de compra; algunos productos suelen ser sometidos a un proceso de subasta, donde los compradores deben realizar “pujas” constantes con el fin de ver quién ofrece la mayor cantidad de dinero.

Suena como un proceso común, pero computacionalmente hablando representa un asunto de cuidado: Actualizar información en tiempo real.





# Planteamiento del problema

Sabemos ya que, según el protocolo bajo el cual hemos trabajado, el cliente debe hacer una petición de información al servidor, para que éste le responda con algo.

Si el comprador 1 hace una puja. ¿Cómo podrá ver el estado de su puja? Tendría que actualizar la página, para que se haga nuevamente la petición al servidor con el estado actualizado.

Ahora, si hay 100 compradores haciendo pujas constantemente. ¿Qué tan consistente será la información? ¿Qué tan eficiente es que tenga que estar refrescando la página cada vez que quiera ver el nuevo estado de la subasta?

# Primer intento: HTTP Long Polling

El HTTP Long polling consiste en que el cliente vuelva a hacer una petición tan pronto como reciba una respuesta del servidor, es decir, bombardea al servidor constantemente de peticiones para emular respuestas “en tiempo real”

Sin embargo, se concluyó que esta operación es costosa en recursos y, al final, un tanto lenta para realmente considerarse “tiempo real”.

# ¿Cómo funciona Websockets?





## Para pensar

Ya analizamos un caso práctico donde el uso de websockets es necesario.

**¿Qué otros casos requerirían de websocket?**

Contesta la encuesta de Zoom

# Websocket: comparación con HTTP

 HTTP	Websocket 
Son peticiones al servidor que esperan una respuesta. Como un walkie talkie.	Es un canal abierto entre servidor y cliente. Como una llamada telefónica.
Se solicita información y se espera una respuesta. Ej: un formulario de login	Se usa para comunicación en tiempo real. Ej: un chat
Se usa para consumir APIs y recursos web	Se usa para escuchar información en tiempo real
Protocolo HTTP	Es un protocolo de comunicación
Conexión de una sola vía	Conexión de doble vía
No sustituye a WebSockets	No sustituye a HTTP

# ¡Importante!

Como podrás notar, se menciona que HTTP no es reemplazo de Websocket, ni websocket es reemplazo de HTTP. Ambos son complementos que se pueden utilizar en conjunto, con el fin de hacer sistemas completos y complejos.





# Break

¡10 minutos y volvemos!

# Sockets en Express con Socket.io

# Socket.io

- ✓ Es una biblioteca de Javascript para poder implementar los sockets anteriormente mencionados.
- ✓ Debido al funcionamiento que hemos visto en clase. **socket.io debe instanciarse tanto de lado del cliente, como del servidor.**
- ✓ Permite utilizar todo el potencial mencionado de los websockets, y cuenta con una API casi idéntica para cliente y para servidor.



# Socket.io:

## características

- ✓ Socket.IO **utiliza principalmente** el protocolo **Websocket** proporcionando la misma interfaz.
- ✓ Se puede **usar como** un **contenedor** para Websocket aunque proporciona muchas más funciones, incluida la transmisión a múltiples sockets, el almacenamiento de datos asociados con cada cliente y E/S asíncronas.
- ✓ Se puede instalar con **npm**.

# Socket.io:

## características

- ✓ **Fiabilidad:** Las conexiones se establecen incluso en presencia de:
  - ✓ proxies y balanceadores de carga.
  - ✓ firewall personal y software antivirus.
- ✓ **Soporte de reconexión automática:** A menos que se le indique lo contrario, un cliente desconectado intentará siempre volver a conectarse, hasta que el servidor vuelva a estar disponible.

# Socket.io:

## características

- ✓ **Detección de desconexión:** Se implementa un mecanismo de heartbeat, lo que permite que tanto el servidor como el cliente sepan cuando el otro ya no responde.
- ✓ **Soporte binario:** Se puede emitir cualquier estructura de datos serializable, que incluye:
  - ✓ ArrayBuffer y Blob en el navegador
  - ✓ ArrayBuffer y Buffer en Node.js

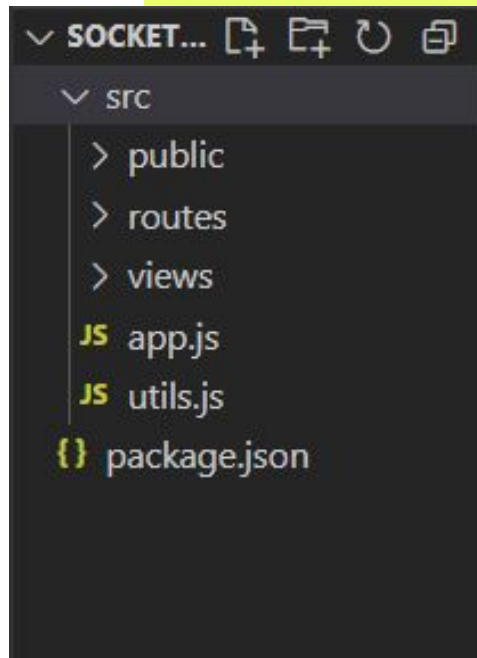
# Instalación y configuración de Socket.io

# 1. Tener listo un servidor Express

Para poder trabajar con websockets en Express, necesitamos un servidor para que trabajen en conjunto, de manera que levantaremos un servidor express como ya lo conocemos.

Utilizaremos la misma estructura de plantillas trabajadas con Handlebars, de manera que debemos contar con la arquitectura

La estructura inicial deberá ser como lo indica la imagen







APROXIMACIÓN AL PROCESO

## 2. Realizar las instalaciones

Una vez que tenemos la estructura de carpetas inicial, realizamos la instalación de nuestros elementos cruciales para trabajar con websockets.

- ✓ **express:** Nuestro servidor principal.
- ✓ **express-handlebars:** Para las plantillas donde colocaremos el socket de lado de cliente.
- ✓ **socket.io:** Para trabajar con websockets, tanto para cliente como para servidor.

```
\Socketioexample> npm install express express-handlebars socket.io
```

# Configuración y uso del socket del lado del servidor

# El cliente y el servidor deben estar ligados

Hasta este momento, tenemos a socket.io de lado del cliente listo para conectarse a nuestro servidor, sin embargo, aún no hemos enseñado a nuestro servidor a escuchar el handshake por parte del cliente, para ello tenemos nuestro socketServer (normalmente se llama "io", pero lo utilizaremos como socketServer para que sea más claro).

Tomaremos nuestro socketServer y lo pondremos "a escuchar una conexión". Una vez que un socket se conecte, podemos reconocerlo y tomar alguna acción a partir de ello.



## Para pensar

El cliente puede comunicarse de una sola forma con el servidor. Sin embargo, como vemos, hay diferentes formas de comunicarse desde el servidor hacia el cliente.

**¿Cuándo deberíamos utilizar cada emit?**

Contesta mediante el chat de Zoom



ACTIVIDAD EN CLASE

# Servidor con Websockets

**Sobre el proyecto de websocket que venimos desarrollando:**

- ✓ Sobre la estructura anteriormente creada, agregar en la vista de HOME un elemento de entrada de texto donde al introducir texto, el mensaje se vea reflejado en todos los clientes conectados en un párrafo por debajo del input.  
El texto debe ser enviado caracter a caracter y debe reemplazar el mensaje previo.



## ACTIVIDAD EN CLASE

# Servidor con Websockets

### Sobre el proyecto de websocket que venimos desarrollando:

- ✓ Basado en el ejercicio que venimos realizando, ahora los mensajes enviados por los clientes deberán ser almacenados en el servidor y reflejados por debajo del elemento de entrada de texto cada vez que el usuario haga un envío. La estructura de almacenamiento será un array de objetos, donde cada objeto tendrá la siguiente estructura:  
`{ socketid: (el socket.id del que envió el mensaje), message: (texto enviado)}`
- Cada cliente que se conecte recibirá la lista de mensajes completa.
- Modificar el elemento de entrada en el cliente para que disponga de un botón de envío de mensaje.
- Cada mensaje de cliente se representará en un renglón aparte, anteponiendo el socket id.

¿Preguntas?

**Opina y valora**  
esta clase



**Muchas gracias.**