



Universidad Nacional de Asunción

Facultad de Politécnica

WebSockets, investigación e implementación

Asignatura: Sistemas Distribuidos.

Tutores:

- Prof. Ing. Fernando Mancía.
- Prof. Ing. Juan M. Ferreira

Alumnos:

- Jara Eichenbrenner, Arturo Gabriel.
- Notario Candia, Alejandro Alexander.
- Ortellado Fernández, Cristian Daniel.
- Paiva Galeano, Guillermo José.
- Weiss Van Der Pol, Ivan.

San Lorenzo, 2021

Contenido

WebSockets.....	3
Preliminares.....	3
¿Qué es WebSocket?	4
¿Qué es el handshake?	4
Ventajas	5
Cuando Utilizar WebSockets	5
¿Qué es Socket.IO?	6
¿Cómo funciona?	7
¿Cómo funciona el sitio web?.....	10
1. Pantalla de Inicio:	10
2. Listado de Hospitales (en <i>Ver Estado</i>):.....	11
3. Editar datos en cada hospital (en <i>Ver Hospital</i>):	12
4. Agregar una Cama:	13
5. Modificar el estado de una cama:	14
6. Eliminar una cama:	14
Bibliografía.....	16

WebSockets

Preliminares

Anteriormente, crear una aplicación web que necesite una comunicación bidireccional entre el cliente y el servidor (por ejemplo, mensajería instantánea y aplicaciones de juegos) ha requerido de un abuso de HTTP para sondear el server por actualizaciones mientras se envían notificaciones en flujo hacia el servidor como diferentes llamadas HTTP.

Esto resultó en una variedad de problemas:

- Cada servidor es forzado a usar un número de diferentes conexiones TCP para cada cliente; uno para enviar información al cliente y uno nuevo para cada mensaje entrante.
- El protocolo de enlace es altamente sobrecargado, con cada mensaje cliente-servidor con propio encabezado HTTP.
- El proceso del lado cliente es forzado a mantener un mapeo de las conexiones salientes a las conexiones entrantes para dar seguimiento a las respuestas.

Una solución simple sería utilizar una única conexión TCP para el tráfico en ambas direcciones. Esto es lo que el protocolo WebSocket provee.

Esta técnica puede ser usada por una variedad de aplicaciones; juegos, ventanas de ventas, aplicaciones multiusuario con modificaciones en simultáneo, interfaces exponiendo servicios del lado servidor en tiempo real, etc.

¿Qué es WebSocket?

WebSocket es un protocolo de red basado en TCP que establece cómo deben intercambiarse datos entre redes. Puesto que es un protocolo fiable y eficiente, es utilizado por prácticamente todos los clientes. El protocolo TCP establece conexiones entre dos puntos finales de comunicación, llamados sockets. De esta manera, el intercambio de datos puede producirse en las dos direcciones.

En las conexiones bidireccionales, como las que crea WebSocket, se intercambian datos en ambas direcciones al mismo tiempo. La ventaja de este intercambio es que se accede de forma más rápida a los datos. En concreto, WebSocket permite así una comunicación directa entre una aplicación web y un servidor WebSocket. En otras palabras: la web que se solicita se muestra en tiempo real.

El protocolo WebSocket es diseñado para reemplazar la comunicación de tecnologías bidireccionales que usan HTTP como una capa de transporte beneficiándose de la infraestructura existente (proxys, autenticación, etc.). Tales tecnologías fueron implementadas como concesiones entre eficiencia y confiabilidad porque HTTP no fue originalmente pensado para su uso bidireccional.

WebSocket, siendo un protocolo full-dúplex usado en el mismo escenario de comunicación cliente-servidor como HTTP, a diferencia de este inicia con ws:// o wss://. Es un protocolo de estado, lo cual significa que la conexión entre cliente y servidor se mantiene operativa hasta que es finalizada por una de las partes involucradas (cliente o servidor). Luego de cerrada la conexión, la conexión es finalizada para ambos extremos de la comunicación.

¿Qué es el handshake?

El handshake se denomina al proceso de establecer la conexión usando el protocolo WebSocket, entre un cliente y un servidor. Se establece mediante TCP, es compatible con intermediarios y software del lado cliente basados en HTTP, de tal forma que un único puerto puede ser usado tanto por clientes HTTP comunicándose con el server y clientes WebSocket en conexión con el server. Para este fin, el handshake del cliente WebSocket es una solicitud HTTP de la siguiente estructura:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
```

```
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==  
Origin: http://example.com  
Sec-WebSocket-Protocol: chat, superchat  
Sec-WebSocket-Version: 13
```

Esta solicitud de upgrade a una conexión WebSocket es respondida por el servidor que maneja dicho protocolo como:

```
HTTP/1.1 101 Switching Protocols  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=  
Sec-WebSocket-Protocol: superchat
```

Este proceso de establecer una conexión anteriormente descrita se conoce como handshake.

Ventajas

El uso tradicional de las conexiones HTTP tiene el inconveniente de que el cliente siempre carga la página HTML entera. Para resolver el problema, se desarrolló la tecnología AJAX. Esta tenía, por su parte, la desventaja de establecer conexiones unidireccionales, es decir, que solo permiten la comunicación en una dirección, lo cual daría lugar a largos tiempos de espera en las intensivas aplicaciones de hoy en día, especialmente en los chats.

WebSocket, en cambio, crea conexiones bidireccionales que permiten el intercambio de datos en ambos sentidos, lo cual hace posible el contacto directo con el navegador y, con ello, permite **cortos periodos de carga**: en cuanto se envía un mensaje, como podría ser uno en un chat de soporte técnico, este llega y se muestra directamente al otro lado.

Cuando Utilizar WebSockets

- Aplicaciones web de tiempo real: Las aplicaciones web de tiempo real usan WebSockets para mostrar el dato en el extremo cliente, el cual está continuamente siendo enviada al servidor del backend. En WebSockets, los datos son continuamente empujadas/transmitidas en la misma conexión ya abierta, por este motivo es rápido y mejora el rendimiento y desempeño de la aplicación.

Por ejemplo, en un sitio web de trading o de bitcoin trading, los cuales se encuentran cambiando constantemente, para mostrar la fluctuación de precio y para la transferencia de datos continuos del servidor de datos continuos al extremo cliente mediante el canal WebSocket.

- Aplicaciones de juegos: En una aplicación de juegos, el enfoque se centra en los datos continuamente siendo recibidos por el server sin recargar la UI, tendrá efecto en la visualización. La UI consigue automáticamente recargar sin necesitar una nueva conexión, lo cual es muy importante una aplicación de videojuegos.

- Aplicaciones de Chat: Una aplicación de chat que usa WebSockets establecerá la conexión una única vez para el intercambio, publicación o la difusión de los mensajes entre los participantes. Se rehusa la misma conexión WebSocket, tanto para enviar como para recibir mensajes y para intercambios entre pares.

¿Qué es Socket.IO?

Socket.IO es una librería que permite la comunicación en tiempo real, bidireccional y basado en eventos entre el navegador y el servidor. Consiste de:

- Un servidor Node.js.
- Una librería cliente JavaScript para el navegador (el cuál puede también ejecutarse con Node.js).

El canal bidireccional entre el servidor Node.js y el cliente Socket.io (navegador, Node.js, o algún otro lenguaje de programación) es establecido con una conexión WebSocket, y usará el sondeo HTTP como alternativa.

Socket.IO enmascara y nos abstrae de los procedimientos básicos del protocolo WebSocket, facilitando el desarrollo de la aplicación web responsiva. Permite al desarrollador enfocarse en los niveles superiores de la estructura y diseño de una web. No necesitamos iniciar el *handshake* ni cerrar la conexión entre cliente servidor, porque se efectúan con esta herramienta sin necesitar que el desarrollador los realice de manera explícita.

¿Cómo funciona?

Tenemos en el extremo servidor las operaciones emitidas por el websocket, en un switch para cada operación válida:

```
// Operaciones solicitadas, de acuerdo a especificaciones del trabajo.
/*Lista de operaciones en server:
- Op 1 (ver_estado): Se ve el estado de las todas las camas, disponible u ocupado.
- Op 2 (crear_cama): Crea una cama en el hospital seleccionado.
- Op 3 (eliminar_cama): Elimina una cama del hospital seleccionado.
- Op 4 (ocupar_cama): Ocupa la cama seleccionada.
- Op 5 (desocupar_cama): Desocupa la cama seleccionada.
- Op 6 (lista_hospitales): Muestra la lista de hospitales.
- Op 7 (lista_camas): Muestra la lista de camas por hospital.
- Op 8 (datos_uti): Datos del hospital para mostrar en tiempo real.
- Otros: Respuesta -1 para transaccion indeterminada.*/

socket.on('operacion', async ( data ) => {

  const { tipo_operacion, datos } = data

  if ( datos ) {
    var { uidHospital, uidCamaUTI } = datos
    uidHospital = String(uidHospital)
  }

  try {
    let response
    socket.emit('bienvenido', {
      bienvenido: `Bienvenido ${data.id}`
    })

    switch ( tipo_operacion ) {
      case 1:
        response = await ver_estado()
        console.log('response', response)

        if (response.estado != 0) {
          socket.emit('responseServer_problemSystem', response)
          return
        }

        socket.emit('responseServer_verEstado', response)
        break;
    }
  }
})
```

El *emit* de las otras operaciones se realiza de manera análoga y se puede observar en el código fuente del proyecto Server (TP_WebSockets/Server/src/Socket.io).

Para las funciones de WebSockets que son llamadas en el archivo anterior, generando una respuesta por parte del servidor, tenemos:

```

// Funciones solicitadas

functionsWebSocket.ver_estado = async () => {
  // Se obtiene todos los datos de cada hospital
  const estadoHospital = await Hospital.verEstadoDeTodosLosHospitales()

  // Registro de actividad
  const registroActividad = new RegistroActividad( null, null, tipoOperacion.verEstado )
  await registroActividad.guardarRegistroActividad()

  // Respuesta del servidor
  const response = new ResponseServer({
    estado: 0,
    mensaje: `ok`,
    tipo_operacion: 1,
    respuesta: estadoHospital
  })

  return response.getResponseServer()
}

```

Las respuestas que se obtiene luego de verificar la base de datos se envían a Socket.io para luego ser actualizada en el extremo cliente, en tiempo real.

Esta función mostrada es de la primera función requerida, *ver estado de una cama*. El proceso de generar la respuesta del server y enviarla, es análogo para las demás funciones WebSocket.

Desde el lado cliente se utiliza el modulo socket.io-client, donde en su archivo main.js importamos y utilizamos como función. Como primer parámetro del socket.io-client al que llamamos SocketIOClient, tenemos la url del servidores y como segundo parámetro *opciones*, en las que se configura el socket cliente una vez llamada la función SocketIOClient.


```

    },
    sockets: {
      connect: function () {
        console.log('socket connected')
      },
      responseServer_verEstado: function (data) {
        console.log('responseServer_verEstado: ', data)
        const { respuesta } = data

        this.constructorDeEstado_listaIdHospitales(respuesta)
        this.constructorDeEstado_listaDatosHospitalCamas(respuesta)
      },
    },
    async created() {
      this.$socket.emit('operacion', {
        tipo_operacion: 1,
      })
    },
  },
}
</script>

```

Con la imagen anterior, se puede ejemplificar parte del proceso. Cuando se llama a la vista "estado", se ejecuta la llamada *created()*, y este llama a la operación del tipo 1.

Entonces, el servidor ejecuta la operación del mismo tipo (en este caso, del tipo 1), que se encuentra en el Switch-case principal del lado servidor (primera imagen), el cual llama a la función *ver_estado()*, cuyo objetivo es brindar los datos de estado de los hospitales.

Ya manejando el servidor, si no hay errores en la comunicación, llama a *responseServer_verEstado* (segunda imagen) que extrae los datos de hospitales de la base de datos.

Por último, el cliente ejecuta el evento iniciado y los datos se visualizan.

¿Cómo funciona el sitio web?

Accediendo a la dirección <https://websocketscamasuti.web.app>, (o <https://camasuti.herokuapp.com>), podemos ver la lista de hospitales y comenzar a interactuar registrando, ocupando, desocupando, listando y viendo el estado de las camas UTI disponibles.

La página presenta las siguientes pantallas:

1. Pantalla de Inicio:



Se presenta la Lista de Hospitales, mostrando los datos de cada hospital y la opción de obtener los datos específicos de cada hospital, además de permitir modificar o actualizar los datos (en *Ver Hospital*). También se puede acceder al listado general del estado de todas las camas, diferenciando entre hospitales (en *Ver Estado*).


2. Listado de Hospitales (en *Ver Estado*):

Podemos acceder a la lista de camas en cada hospital. El punto parpadeante indica el estado de cada cama. Rojo es desocupado y verde como ocupado.

En este ejemplo, tenemos el hospital 1,

Hospital 1

Hospital 2



1 - IPS

El Instituto de Previsión Social o IPS es el instituto encargado de administrar el seguro social en el Paraguay. Fue creado por Decreto Ley N° 18071 del 18 de febrero de 1943 en el gobierno de Higinio Morínigo.

Cantidad de camas 3. - Cantidad de pacientes 2.

Cama: 6n1uWlqqEdpfh1D5DNZ

Estado: Desocupado

Cama: VNErVWkdHnLCo57BrBvZ

Estado: Ocupado


Cama: VnBOrDfKsU6LwgSYdOLM

Estado: Ocupado

com/estado/hospital-1

y el hospital número 2:

Hospital 1
Hospital 2



2 - Sanatorio Metropolitano

Desde 1987, se ha destacado por su calidad y calidez en la atención, que lo coloca en un lugar de prestigio entre la comunidad médica, pacientes, familiares y público en general, apoyándose en dos pilares fundamentales para la recuperación de la salud: los avances médicos, tanto en tecnología como en conocimiento, y una atención con alto sentido humano y de respeto por el paciente.

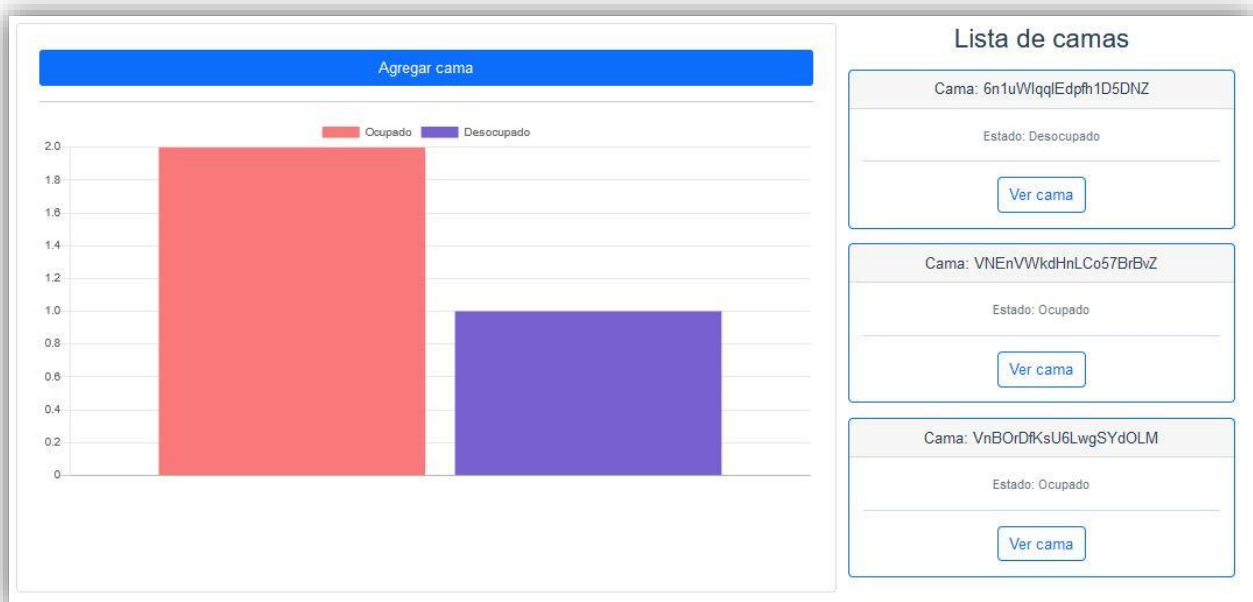
Cantidad de camas 2. - Cantidad de pacientes 1.

Cama: 0Svl2Pfj6wpMHgr51K0B
Estado: Ocupado

Cama: txbeg8hvNmgT3tcFaRht
Estado: Desocupado

3. Editar datos en cada hospital (en Ver Hospital):

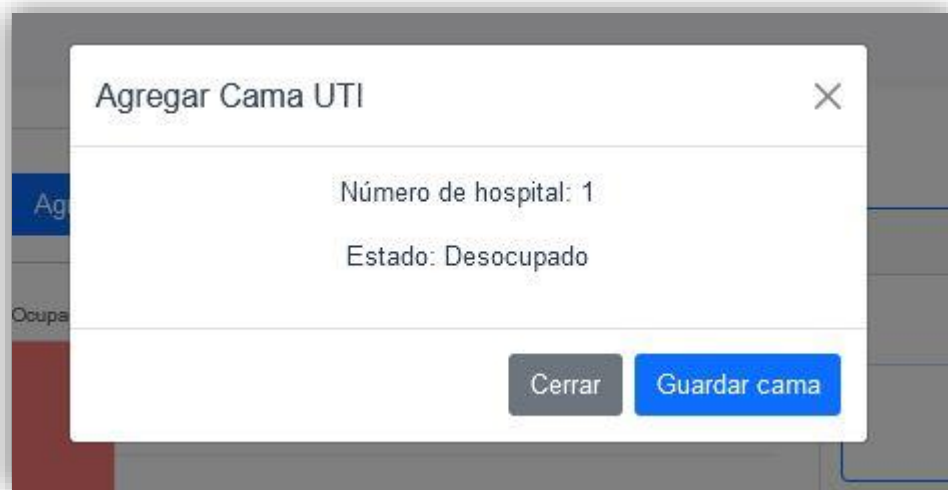
Si accedemos a la opción Ver Hospital, en la pantalla de inicio (ítem 1), se despliega una lista de las camas en el respectivo hospital, además de un chart representando las estadísticas en tiempo real de el nivel de ocupación de las camas.



Podemos agregar una nueva cama en el hospital (en Agregar Cama), o ver una cama específica para modificar su estado (en Ver Cama).

4. Agregar una Cama:

Continuando de la pantalla 4, podemos agregar una nueva cama.



A screenshot of a web application showing a modal dialog box titled "Agregar Cama UTI". The dialog has a close button (X) in the top right corner. Inside the dialog, there are two text labels: "Número de hospital: 1" and "Estado: Desocupado". At the bottom right of the dialog, there are two buttons: "Cerrar" (Close) and "Guardar cama" (Save bed). The "Guardar cama" button is highlighted in blue. The background of the application is partially visible, showing a sidebar with a blue header and a red footer.

Se guardan los cambios y se muestra el estado actualizado del hospital en la pantalla del ítem 3.

5. Modificar el estado de una cama:

En Ver Cama, de la pantalla del ítem 3, se puede modificar el estado de una cama.

Cama: 6UFXxPRYghoO3T0wpjpv

Estado: Desocupado

Ocupar Desocupar Eliminar

Región de los mensajes de error

Si se encuentra desocupado e intentamos ocupar, el estado de la cama se actualiza.

Si intentamos desocupar una cama ya desocupada, nos muestra un mensaje de error:

Operación: 5 Estado: 777 X

No se puede desocupar una cama desocupada

6. Eliminar una cama:

Con la opción de Eliminar Cama, en la pantalla del ítem 5, podemos eliminar una cama del sistema.



Eliminación de una cama UTI

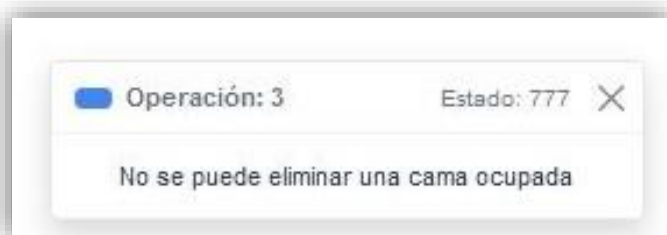
Escriba la uid de la cama: 6UFXxPRYghoO3T0wpjpv

UID Cama

Cerrar

Debemos ingresar el UID de la cama para confirmar la eliminación. Confirmamos y se regresa a la pantalla anterior.

Si la cama se encuentra actualmente ocupada, entonces nos muestra el siguiente mensaje de error:



Operación: 3 Estado: 777

No se puede eliminar una cama ocupada

Y no se elimina la cama.

Bibliografía

- Ian Fette, Alexey Melnikov. *The WebSocket Protocol*. IETF. Recuperado el 24 de agosto de 2021. <https://datatracker.ietf.org/doc/html/rfc6455>
- Digital Guide, IONOS (07 de agosto de 2020). *¿Qué es WebSocket?* IONOS. Recuperado el 25 de agosto de 2021. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-websocket/>
- Arpit Asati (actualizado el 27 de junio de 2021). *What is web socket and how it is different from the HTTP*. Recuperado el 24 de agosto de 2021. <https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>
- Nikhil. *How to Implement WebSocket Server with Node.js*. Mind Browser. <https://www.mindbrowser.com/websockets-with-node-js/>