SoqueTIC

(Actualizado para SoqueTIC v1.2.6 *Hornero*)

Índice

- Índice
- ¿Qué es?
- Inspiración e Idea
- En Frontend
  - Instalación
  - Uso
    - getData
    - postData
    - receive
    - send
    - connect2Server
- En Backend
  - Instalación
  - Uso
    - onEvent
    - sendEvent
    - startServer
    - Buenas prácticas
- DEMOS
- Usos comunes con ejemplos
  - Comunicación iniciada por el frontend
  - Comunicación iniciada por el backend

¿Qué es?

¡Hola soquete! SoqueTIC es una herramienta desarrollada por el equipo de ORT TIC Belgrano para facilitar la comunicación frontend ⟺ backend en proyectos desarrollados con HTML, CSS, JS y Node JS.

La idea es simplificar los proyectos y la enseñanza de programación en el primer año de la orientación, ocultando conceptos de comunicación por internet que no son tan relevantes para estos primeros pasos en el mundo de software y corresponden a enseñanzas posteriores. Sin embargo, por las herramientas que se utilizan ese año, no se puede dejar afuera este tipo de comunicación si se quiere habilitar proyectos medianamente ambiciosos. Por eso "disfrazamos" la comunicación por internet con esta herramienta.

SoqueTIC entonces no es más que un envoltorio de la librería SocketIO, adaptándola a una sintaxis más funcional, similar a conceptos vistos en clase. Se pierde un poco del potencial de SocketIO, pero se dejo lo mínimo indispensable para desarrollar proyecto de 3ero.

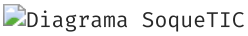
Inspiración e Idea

Para aprender a programar, se usa HTML, CSS y JS ejecutados en el browser, lo cual nos permite rápidamente crear buenas interfaces de usuario. Sin embargo, este entorno está pensado para páginas web, es decir, que estos archivos vendrían por internet para ser ejecutados por nuestro browser. Por eso, el browser los ejecuta *"sandboxeados"*, es decir, con un acceso limitadísimo a los archivos de nuestra computadora. ¡Imaginen los virus que habría si eso fuese posible!

Por eso, a pesar qué los proyectos en 3ero sean puramente locales, necesitamos obligatoriamente la división backend/frontend para saltar esta limitación. El backend sí tiene acceso total a los recursos de la computadora, entonces necesitamos tenerlo para hacer cosas tan básicas como leer y escribir archivos. Pero la interfaz de usuario se sigue haciendo en el browser.

¿Y entonces cómo comunicamos frontend y backend? Más si no sabemos comunicación a través de internet. *Enter SoqueTIC*. SoqueTIC es quien va a realizar la comunicación entre ambos programas, tal como muestra el siguiente

diagrama:



Algunos puntos importantes:

> El usuario interactúa con el frontend, que es el que tiene los elementos gráficos e interactivos. **El browser no puede ni leer ni escribir archivos. Tampoco leer el puerto serial para usar arduino, o cambiar características del sistema operativo.** Si la interacción del usuario requiere estas cosas, se debe enviar un mensaje al backend para que las haga usando SoqueTIC.
> El backend tiene acceso a todo salvo a la interfaz gráfica. Es decir, responde pedidos del frontend (que recibe input del usuario) e interactúa con archivos, periféricos y sistema operativo. Si el backend necesita actualizar algo en la interfaz gráfica para informar al usuario, necesariamente debe usar soqueTIC para realizarlo. **El DOM (document) solo existe en el frontend porque modela la página web**, no puede haber un llamado a `document` en el backend.

En Frontend

SoqueTIC comunica frontend y backend. Esta sección se dedica a mostrar esta herramienta desde el lado de un frontend hecho con HTML, CSS y JS para ser corrido por algún browser (Chrome, Firefox, etc.)

Instalación

Para usar a SoqueTIC en un archivo HTML, se debe hacer lo siguiente:

1) Importar el script de SocketIO y el de SoqueTIC Esto se puede hacer con los siguientes tags:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.8.1/socket.io.js"></script>
<script src="https://cdn.jsdelivr.net/gh/JZylber/SoqueTIC-Client@v1.2.6/soquetic-client.js"></script>
```

2) Linkear el archivo en el que van a usar SoqueTIC debajo de estos dos `<script>`.

IMPORTANTE: Para poder ejecutar SoqueTIC no alcanza con abrir el HTML en el browser: hay que armar un live server. La forma más común de hacer esto es usando la extensión de VS Code.

Uso

Para comunicarse con el backend, se pueden usar las funciones descritas a continuación. Obviamente, si el backend no está encendido y ejecutándose, nada va a andar.

getData

Esta función está pensada para hacer pedidos al backend pero no es necesario mandarle nada para que pueda responder al pedido. Esta recibe 2 parámeteros:

> `type` con el que se pueden distinguir distintos eventos. Debe coincidir con alguna función del backend.
> `callback` **función** a ser llamada cuando el servidor responda con la información deseada. Esta debe tomar un único parámetro, `data`, que sería la información a recibir. Lo que recibe es lo que sea que la función a la que respondió a este evento en el backend haya retornado.

postData

Esta función esta pensada para mandarle información al backend o para hacerle pedidos que impliquen mandarle información. Esta recibe 3 parámeteros (1 opcional):

> `type` con el que se pueden distinguir distintos eventos. Debe coincidir con alguna función del backend.
> `data` es la información a ser enviada al servidor. Es un único parámetro, si se quiere mandar un conjunto de datos usar una estructura de datos que modele conjuntos, como listas u objetos.
> `callback` (opcional, puede quedar vacío) **función** a ser llamada cuando el servidor responda con la información deseada. Esta debe tomar un único parámetro, `data`, que sería la información que recibe del servidor. Lo que recibe es lo que sea que la función a la que respondió a este evento en el backend haya retornado.

receive

Reservada para eventos en tiempo real, es decir, para cuando el proyecto cuenta con actualizaciones periódicas del backend. Esta función está para procesar eventos iniciados desde el backend, y toma dos parámetros:

> `type` con el que se pueden distinguir distintos eventos. Debe coincidir con alguna función del backend.
> `callback` **función** a ser llamada cuando el servidor envía un evento al frontend. Esta debe tomar un único parámetro, `data`, que sería la información a recibir. Lo que recibe es lo que sea que la función que emitió el

evento en el backend haya enviado.

send

DEPRECADA al ser redundante con postData. Queda *legacy* para algunos proyectos en versiones previas de SoqueTIC.

connect2Server

Función necesaria para iniciar la conexión al servidor. Solo puede conectarse a servidores locales. Toma un parámetro **opcional** `PORT`, que en caso de no especificarse toma el valor 3000. Pasarle el puerto si es que el servidor está corriendo en un puerto distinto a 3000.

IMPORTANTE: Se debe llamar a `connect2Server` en cada archivo que utilice SoqueTIC para comunicarse con el backend.

En Backend

SoqueTIC comunica frontend y backend. Esta sección se dedica a mostrar esta herramienta desde el lado de un backend hecho con Node JS.

Instalación

SoqueTIC está publicado en npm, por lo tanto, para instalarlo alcanza con correr el siguiente comando dentro del proyecto:

```
npm i soquetic
```

Luego se deben importar las funciones correspondientes en cada archivo js que las use. Por ejemplo, para importar todas las funciones, se puede hacer

```
import { onEvent, sendEvent, startServer } from "soquetic";
```

Uso

Para comunicarse y recibir eventos del frontend, se pueden usar las funciones descritas a continuación.

onEvent

Función para escuchar eventos emitidos desde el frontend. Toma 2 parámetros.

`type` es un string que se utiliza para identificar el evento a responder. Debe coincidir con el llamado del frontend.
`callback` es la **función** a ser llamada cuando llegue dicho evento. Tiene que tomar un único parámetro, `data`, en donde llega la información necesaria para responder al evento. El retorno de esta función es lo que será enviado al frontend.

sendEvent

Esta función se usa para eventos de tiempo real. Esto es ya que envía eventos al frontend sin que necesariamente los pida. Toma 2 parámetros:

`type` con el que se pueden distinguir distintos eventos. El frontend debe tener un `recieve` con el mismo tipo.
`data` es la información a ser enviada al frontend. Es un único parámetro, si se quiere mandar un conjunto de datos usar una estructura de datos que modele conjuntos, como listas u objetos.

startServer

Sirve para inicializar el backend y colgarse escuchando eventos. Esta función se debe usar en el archivo principal a correr para levantar el servidor. Es decir, `startServer` necesariamente se debe ejecutar en el archivo siendo corrido por Node JS para que soqueTIC funcione.

Esta función toma 2 parámetros **opcionales**:

`PORT`: número del puerto en donde el servidor va a escuchar requests. En caso de no especificar, su valor default es `3000`.
`DEBUG`: determina si iniciar SoqueTIC en modo DEBUG o no. En caso de no especificar, su valor default es `true`. En el modo DEBUG, el servidor escribe por la consola todo lo recibido y enviado, entre otros mensajes sobre el funcionamiento de SoqueTIC.

Buenas prácticas

Para usar SoqueTIC hay que hacer tantos `onEvent` como eventos quiero saber responder, y en el archivo principal a correr con `node JS` llamar a la función `startServer`. Si hay periféricos que deban actualizar su estado al frontend, usar la función `sendEvent`.

DEMOS

Muchas veces no hay nada mejor ver un ejemplo para entender mejor. A continuación una serie de proyectos de prueba hechos por el equipo docente usando SoqueTIC.

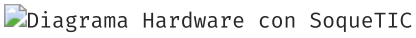
- Demo Básica: Envío de mensajes de frontend a backend
- Demo Arduino: Envío de mensajes entre frontend, backend y arduino
- Fixture: Ejemplo más complejo de frontend y backend con lectura y escritura de json.

Usos comunes con ejemplos

A continuación exponemos y explicamos los casos de uso de SoqueTIC más comunes. Para que se entienda mejor, los exponemos con ejemplo

Comunicación iniciada por el frontend

Como el usuario es quien realiza la mayoría de las acciones, y este interactúa con el frontend, es generalmente el frontend quien inicia la comunicación con el backend. Mostremos el caso en el que el frontend le envía cierta información al backend y hace algo con la respuesta. El siguiente diagrama modela esa situación, en donde el frontend usa la función `postData` y el backend la función `onEvent`:



La función `postData` envía el evento de nombre *type* con el parámetro *data* al backend. En el diagrama, type es "envio" y la data es `dataS`. `dataS` puede ser de cualquier tipo. Para enviar más de una sola cosa, usar un tipo de datos que modele conjuntos, como listas u objetos.

En el backend, hay alguien esperando el evento de este tipo: la función `onEvent`. En *type* tiene el mismo string que `postData`, en este caso `envio`. Y en callback, tiene la función que se va a encargar de procesar la data enviada por el frontend, en este caso, `f(data)`. Cuando el frontend ejecuta `postData` enviando datos, el backend los recibe y llama al callback con la información recibida. En este caso, llama a f, pasandole como primer y único parámetro `dataS`.

Al final la ejecución del callback de `onEvent`, lo que sea que retorne dicha función volverá al frontend como respuesta. En este caso, la ejecución de f con `dataS` retorna `dataR`. Del lado del frontend, el tercer parámetro de `postData` era el callback, en este caso, `g(data)`. Este es el encargado de recibir la respuesta del backend. Cuando el backend termina la ejecución de su callback (f), lo que sea que retorne (dataR), es enviado como primer y único parámetro a un llamado de la función de callback del frontend (g). En este caso, cuando el backend responde con `dataR`, el frontend ejecuta la función g en donde su parámetro data es `dataR`.

Para dar un ejemplo, vamos a usar el código de la Demo Básica. En este caso, el usuario escribe una palabra en el input, cuando apreta un botón lo envía al backend, este lo recibe y lo devuelve. El frontend toma lo devuelto y lo muestra en pantalla. Veamos el fragmento de código relevante del frontend:

```
form.addEventListener("submit", (e) => {
  e.preventDefault();
  if (input.value) {
    postData("message", { msg: input.value }, (data) => {
      a.innerHTML = data.msg;
    });
    input.value = "";
  }
});
```

Se puede ver como cuando se hace "submit" del formulario (apretar el botón que hace submit), se llama a una función que ejecuta postData. Esta entonces envía el evento "message" con un objeto de un único atributo, *msg*, que tiene lo que el usuario escribió en el input (`input.value`).

Del lado del backend, hay un onEvent apropiado esperándolo. A continuación el código:

```
onEvent("message", (data) => {
  console.log(`Mensaje recibido: ${data.msg}`);
  return { msg: `Mensaje recibido: ${data.msg}` };
});
```

En este caso, al ejecutarse postData en el frontend, el backend llama a su callback recibiendo el objeto en el parámetro *data*. Lo muestra por consola y retorna **otro** objeto con un único atributo, también de nombre *msg*, que contiene el string "Mensaje recibido: " + el string recibido en el atributo *msg* del objeto enviado por el frontend.

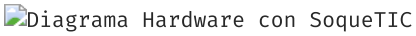
Al retornar este objeto, el frontend lo recibe llamando a su callback y pasándoselo por parámetro. En este caso, toma el parámetro en *data*, y con ese modifica el *innerText* de un `<p>` previamente creado con el atributo *msg* del objeto con el que respondió el backend.

El caso de uso de `getData` es muy similar, la única diferencia es que no se envía ningún parámetro al callback de `onEvent`, por lo tanto, este pasa a ser una función que no toma parámetros.

Comunicación iniciada por el backend

Un ejemplo de comunicación iniciada por el backend es por ejemplo cuando hay un componente de hardware.

En este caso, la comunicación iniciada por input del usuario se da igual, ya que el usuario interactúa con el frontend. La diferencia aparece para el caso en el que el backend es el que desea enviar un mensaje, sin que necesariamente lo pida el frontend. Como es el backend el que tiene la posibilidad de comunicarse a recursos externos como hardware, es el backend el que recibe información de ellos. Entonces, para informar al usuario, el backend debe iniciar la comunicación con el frontend en vez de esperar eventos. Para eso, utiliza la función `sendEvent`. A su vez, el frontend recibe ese mensaje con la función `recieve`. El siguiente diagrama ilustra esa situación:



Para dar un ejemplo, vamos a usar el código de la Demo Arduino. En esta, el usuario elige un color desde el frontend y el led toma ese color. A su vez hay un botón que prende/apaga el LED, y se ve por pantalla si el LED está prendido o apagado.

La parte de enviar el color de frontend a hardware es similar a lo visto anteriormente, la única diferencia es que la función `onEvent` inicia comunicación serial para informar al arduino.

El caso interesante es cómo el estado del botón (el prendido y apagado del LED) llega al frontend. Lo primero que ocurre es que el estado del botón es informado al backend. Esto se hace usando comunicación serial, y el backend usa la librería serialport para realizar este tipo de comunicación. A continuación, el fragmento de código relevante del **backend**:

```
port.on("data", function (data) {
  let status = data.toString().trim();
  let ledOn = status === "on";
  sendEvent("boton", { on: ledOn });
});
```

No nos interesa deternos en la sintaxis de la librería serialport, basta saber que en el parámetro `data` viene la información recibida por el puerto serial. El backend procesa la información, pero a diferencia de lo visto en la sección anterior, no alcanza con retornar, ya que **no está respondiendo un pedido del frontend**. Debe iniciar activamente el intercambio con el frontend, y para eso utiliza la función sendevent. El frontend, por su parte, debe estar preparado para recibir los eventos del backend. A continuación el fragmento de código relevante del **frontend**:

```
function botonApretado(status){
  if(status.on){
    estado.innerText = "prendido";
  } else {
    estado.innerText = "apagado";
  }
}
receive("boton",botonApretado)
```

Como vemos, el `recieve` es del mismo tipo que el `sendevent`, y toma en el parámetro del *callback* lo enviado por el frontend y en base a eso refleja información en pantalla.