Reflejo del cambio en tiempo real

### **Pooling**

La más simple de todas las formas es el pooling. Es muy simple, para saber si algo paso, tenemos que preguntar constantemente. La estrategia del pooling consiste en consultar al servidor en un periodo constante de tiempo, usualmente muy pequeño, digamos cada 3 segundos.

**Ventajas**

* Es extremadamente simple, realizar peticiones constantes cada cierto periodo de tiempo.
* No requiere de tecnologías especiales más que AJAX, para poder hacer las consultas.
* Muy fácil de implementar, sólo colocas tus consultas en un intervalo y listo.

**Desventajas**

* Sobrecargas al servidor, es muy probable que muchas de las peticiones reciban como respuesta nada, en caso de que no haya información nueva que comunicar, sin embargo, las peticiones siguen ejecutándose y el servidor tiene que responderlas.
* Pequeña latencia. Si tu aplicación es de respuesta crítica, considera que con el pooling siempre habrá un ligero retraso entre el momento en el que la información se produce y el momento en el cliente se entera, si por ejemplo, mandas una petición cada 10 segundos, este será el tiempo máximo en que la información podría llegar retrasada.

Variante de pooling **Long pooling**

El flujo de una implementación con long pooling es la siguiente:

* El cliente envía un mensaje Http al servidor consultando información nueva.

El servidor tiene de dos:

1. Si existe información nueva que reportar, la envía inmediatamente.
2. Si no existe información nueva que reportar, mantiene esta conexión Http en espera y abierta, hasta que exista algo que reportar, entonces envía la información al cliente y cierra dicha conexión.

* El cliente recibe respuesta de su mensaje con datos nuevos, ya sea tan pronto como la envió o luego de haber esperado por bastante tiempo a que hubiera algo nuevo que reportar.
* El cliente manda una nueva petición hasta que la anterior fue contestada, así, esta nueva recibirá respuesta hasta que haya nuevos datos.

**Ventajas**

* Todas las del pooling, a final de cuentas es casi la misma metodología.
* Menos peticiones que el pooling, por lo tanto, un servidor con menos carga y más eficiente.
* Mejora significativa en qué tan rápido recibimos los datos nuevos, ya que para cuando estos se crean, ya hay una conexión esperando para que se envíen al cliente.

**Desventajas**

* Seguimos realizando y abriendo peticiones Http, aún cuando estas quizás nunca reciban respuesta.
* Algunos servidores no permiten que las conexiones Http permanezcan abiertas por mucho tiempo, por lo que cada que se cierran, debemos crear peticiones nuevas, aumentando la carga del servidor.

### **WebSockets**

En 2010, justo en la cúspide de la popularidad del término HTML5, se introdujo al navegador la habilidad de establecer conexión a dos vías, directamente con el servidor, las WebSockets.

**Ventajas**

* Tiempo de respuesta más rápido que otros mecanismos, ideal para juegos multijugador, chats, transacciones, etc.
* Aplicaciones más rápidas, ya que la comunicación queda permanentemente abierta, mientras que los mecanismos con Http involucran recorridos completos por la red para volver a enviar un mensaje al servidor.
* Modernas, con excelentes herramientas para el Backend y el Frontend.

**Desventajas**

* No son fáciles de testear.
* Pueden implicar recursos más caros cuando tienes muchos usuarios.
* Debes hacerlas tolerante a fallos, ¿qué pasa si hay una desconexión?
* En ocasiones puedes toparte con mensajes repetidos y carreras de tiempo. Resolver estos conflictos es tu responsabilidad.
* Implican trabajo y configuración en el backend, a diferencia de las estrategias con Http.

### **Eventos del servidor**

Los SSE (server-sent events) son eventos enviados del servidor para comunicar mensajes hacia el cliente. A diferencia de las WebSockets, esta conexión no es a dos vías, es sólo el servidor el que le envía datos al cliente. Lo interesante de este enfoque es que precisamente es el servidor quien sabe cuándo debe comunicar nuevos datos, cuándo se generó información nueva para enviar, etc. Por lo que tiene sentido que sea el servidor quien se encargue de iniciar la comunicación con el cliente.

**Ventajas**

* Más fáciles de configurar que las WebSockets, sin embargo, sí requieren modificaciones en el backend a diferencia del Pooling y el Long Pooling.
* Más simples en ambos lados el cliente y el servidor.
* Comunicación directa del servidor hacia el cliente.
* Usan el protocolo Http.

Desventajas

* No es soportado ni en Edge, ni en ninguna versión de Internet Explorer. Las WebSockets, por otro lado, tiene soporte en Edge y en IE11.
* Cada mensaje nuevo, es una petición nueva.
* Son más fáciles de probar que las WebSockets, pero no que los mecanismos de pooling.

[Referencia codigofacilito](https://codigofacilito.com/articulos/como-programar-aplicaciones-en-tiempo-real)

[Referencia telerik](https://www.telerik.com/blogs/websockets-vs-server-sent-events)

**Conclusiones**:

Estrategia seleccionada WebSockets

Debido al lo que nos ofrecen las diferentes estrategias hemos optado elegir la estrategia de websockets, si bien tuvimos dudas al decidirnos sobre websockets y eventos del servidor ya que ambas tecnologías nos permiten hacer lo que necesitamos sin consumir demasiados recursos como lo hace el pooling, hemos seleccionado websockets ya que es más de nuestro interés aprender una estrategia que es bidireccional y además que satisface las necesidades del proyecto sin problema.