

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO
ARQUITECTURA DE SOFTWARE – ARSW

LABORATORIO 8
STOMP

PROFESOR
JAVIER IVÁN TOQUICA BARRERA

ESTUDIANTES
NICOLÁS ARIZA BARBOSA
MATEO OLAYA GARZÓN
SANTIAGO ANDRÉS ROCHA CRISTANCHO
DAVID EDUARDO VALENCIA CARDONA

BOGOTÁ D.C, NOVIEMBRE 2023

Tabla de contenido

Introducción:3

Desarrollo:.....3

 Parte I:3

 Parte II:5

 Parte III:6

 Parte IV:7

Conclusiones:9

Introducción:

El presente laboratorio de Arquitectura de Software se enfoca en la creación de una aplicación de dibujo colaborativo basada en tecnologías web como STOMP, WebSockets, HTML5 Canvas y JavaScript. El objetivo es permitir a múltiples usuarios colaborar en la creación de dibujos en tiempo real. El laboratorio se divide en cuatro partes, cada una de las cuales representa un paso incremental en el desarrollo de la aplicación.

Desarrollo:

Parte I:

1. Haga que la aplicación HTML5/JS al ingresarle en los campos de X y Y, además de graficarlos, los publique en el tópico: /topic/newpoint . Para esto tenga en cuenta (1) usar el cliente STOMP creado en el módulo de JavaScript y (2) enviar la representación textual del objeto JSON (usar JSON.stringify). Por ejemplo:
 - a. //creando un objeto literal
stompClient.send("/topic/newpoint", {}, JSON.stringify({x:10,y:10}));
 - b. //enviando un objeto creado a partir de una clase
stompClient.send("/topic/newpoint", {}, JSON.stringify(pt));

```
if (pointsBuffer.length%4 == 0) {  
    stompClient.send("/topic/newpolygon." + currentNumber, {}, JSON.stringify(pointsBuffer));  
    pointsBuffer = []  
} else {  
    stompClient.send("/topic/newpoint." + currentNumber, {}, JSON.stringify(pt));  
}
```

2. Dentro del módulo JavaScript modifique la función de conexión/suscripción al WebSocket, para que la aplicación se suscriba al tópico "/topic/newpoint" (en lugar del tópico /TOPIC0XX). Asocie como 'callback' de este suscriptor una función que muestre en un mensaje de alerta (alert()) el evento recibido. Como se sabe que en el tópico indicado se publicarán sólo puntos, extraiga el contenido enviado con el evento (objeto JavaScript en versión de texto), conviértalo en objeto JSON, y extraiga de éste sus propiedades (coordenadas X y Y). Para extraer el contenido del evento use la propiedad 'body' del mismo, y para convertirlo en objeto, use JSON.parse. Por ejemplo:
 - a. var theObject=JSON.parse(message.body);

```
// Suscribirse a /topic/newpoint
stompClient.subscribe("/topic/newpoint." + number, function (eventbody) {
    theMessage = JSON.parse(eventbody.body);
    addPointToCanvas(theMessage);
});

// Suscribirse a /topic/newpolygon
stompClient.subscribe("/topic/newpolygon." + number, function (eventbody) {
    theMessage = JSON.parse(eventbody.body);
    console.log("theMessage");
    console.log(theMessage);
    drawPolygon(theMessage);
});
```

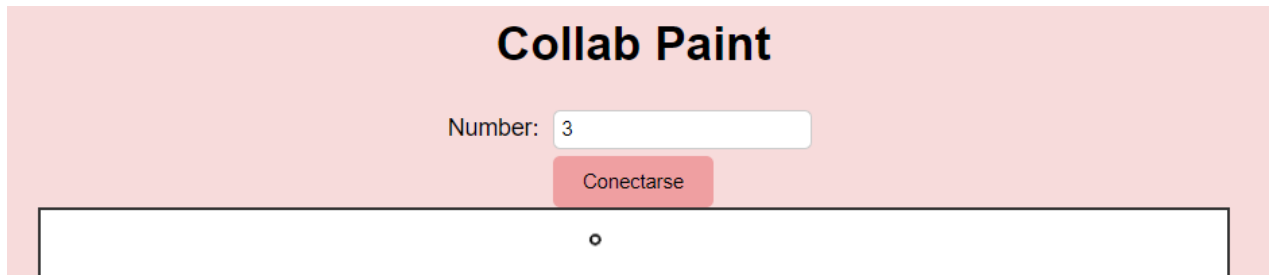
3. Compile y ejecute su aplicación. Abra la aplicación en varias pestañas diferentes (para evitar problemas con el caché del navegador, use el modo 'incógnito' en cada prueba).
4. Ingrese los datos, ejecute la acción del botón, y verifique que en todas las pestañas se haya lanzado la alerta con los datos ingresados.

The screenshot shows a web application titled "Collab Paint" with a text input field containing the number "3" and a "Conectarse" button. The application is displayed in two browser tabs. In the center, a developer console shows a series of WebSocket messages. The messages include subscription confirmations and data points being sent to the application. The data points are represented as objects with 'x' and 'y' coordinates. For example, one message shows a point at (295, 8579559326172). The console also shows the application's internal logic, such as "Adding point..." and "Actual Polygon".

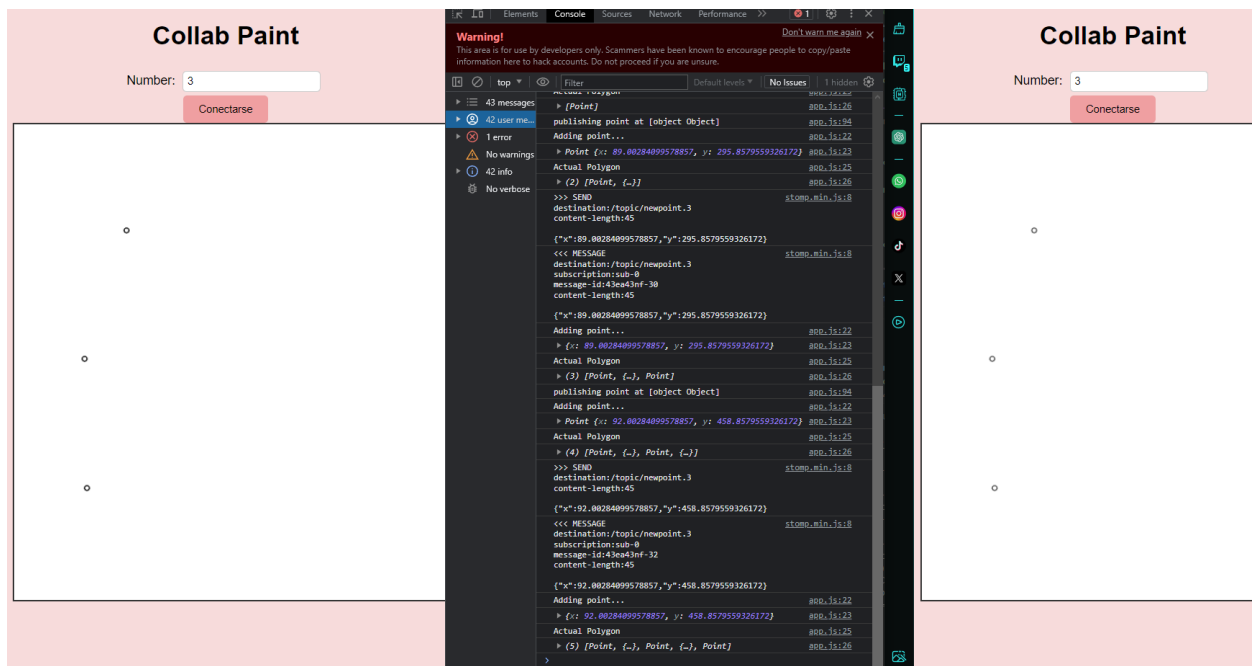
5. Haga commit de lo realizado, para demarcar el avance de la parte 2.
 - a. git commit -m "PARTE 1".

Parte II:

1. Haga que el 'callback' asociado al t pico /topic/newpoint en lugar de mostrar una alerta, dibuje un punto en el canvas en las coordenadas enviadas con los eventos recibidos. Para esto puede [dibujar un c rculo de radio 1](#).



2. Ejecute su aplicaci n en varios navegadores (y si puede en varios computadores, accediendo a la aplicaci n mediante la IP donde corre el servidor). Compruebe que a medida que se dibuja un punto, el mismo es replicado en todas las instancias abiertas de la aplicaci n.

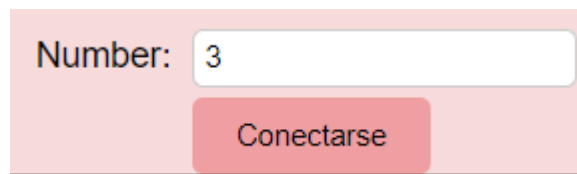


3. Haga commit de lo realizado, para marcar el avance de la parte 2.

1. git commit -m "PARTE 2".

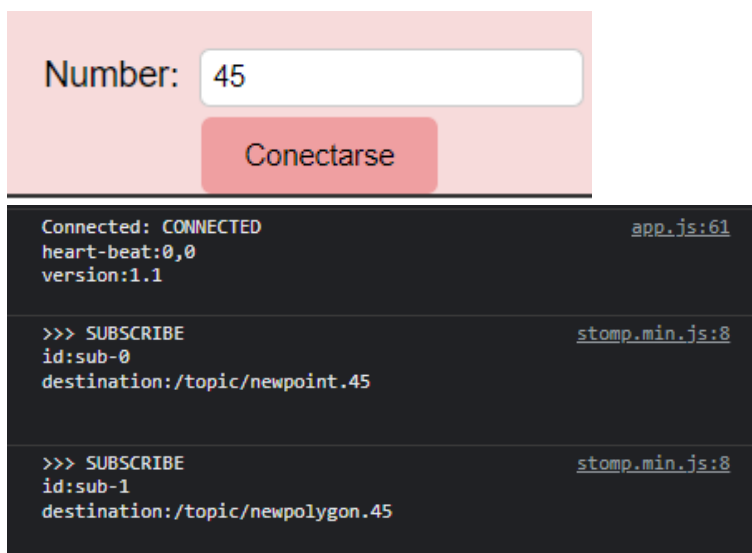
Parte III:

1. Agregue un campo en la vista, en el cual el usuario pueda ingresar un número. El número corresponderá al identificador del dibujo que se creará.



A form with a light pink background. It contains a label 'Number:' followed by a text input field containing the number '3'. Below the input field is a red button with the text 'Conectarse'.

2. Modifique la aplicación para que, en lugar de conectarse y suscribirse automáticamente (en la función `init()`), lo haga a través de botón 'conectarse'. Éste, al oprimirse debe realizar la conexión y suscribir al cliente a un tópico que tenga un nombre dinámico, asociado el identificador ingresado, por ejemplo: `/topic/newpoint.25`, `topic/newpoint.80`, para los dibujos 25 y 80 respectivamente.



The top part shows a form with a light pink background. It contains a label 'Number:' followed by a text input field containing the number '45'. Below the input field is a red button with the text 'Conectarse'.

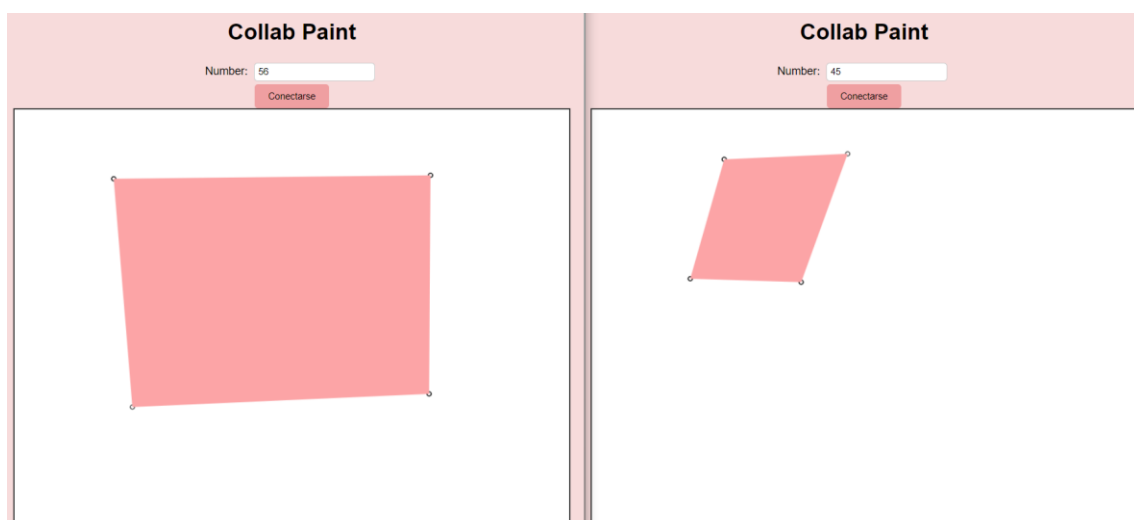
The bottom part shows a terminal window with a dark background. It displays the following output:

```
Connected: CONNECTED                                app.js:61
heart-beat:0,0
version:1.1

>>> SUBSCRIBE                                       stomp.min.js:8
id:sub-0
destination:/topic/newpoint.45

>>> SUBSCRIBE                                       stomp.min.js:8
id:sub-1
destination:/topic/newpolygon.45
```

3. De la misma manera, haga que las publicaciones se realicen al tópico asociado al identificador ingresado por el usuario.
4. Rectifique que se puedan realizar dos dibujos de forma independiente, cada uno de éstos entre dos o más clientes.



Two side-by-side windows, each titled 'Collab Paint'. Each window has a light pink header with a 'Number:' label and a text input field, and a red 'Conectarse' button below it.

The left window shows a large red rectangle drawn on a white canvas. The right window shows a smaller red parallelogram drawn on a white canvas.

Parte IV:

1. Cree una nueva clase que haga el papel de 'Controlador' para ciertos mensajes STOMP (en este caso, aquellos enviados a través de "/app/newpoint.{numdibujo}"). A este controlador se le inyectará un bean de tipo `SimpMessagingTemplate`, un Bean de Spring que permitirá publicar eventos en un determinado tópico. Por ahora, se definirá que cuando se intercepten los eventos enviados a "/app/newpoint.{numdibujo}" (que se supone deben incluir un punto), se mostrará por pantalla el punto recibido, y luego se procederá a reenviar el evento al tópico al cual están suscritos los clientes "/topic/newpoint".

```
@Controller
public class STOMPMessagesHandler {

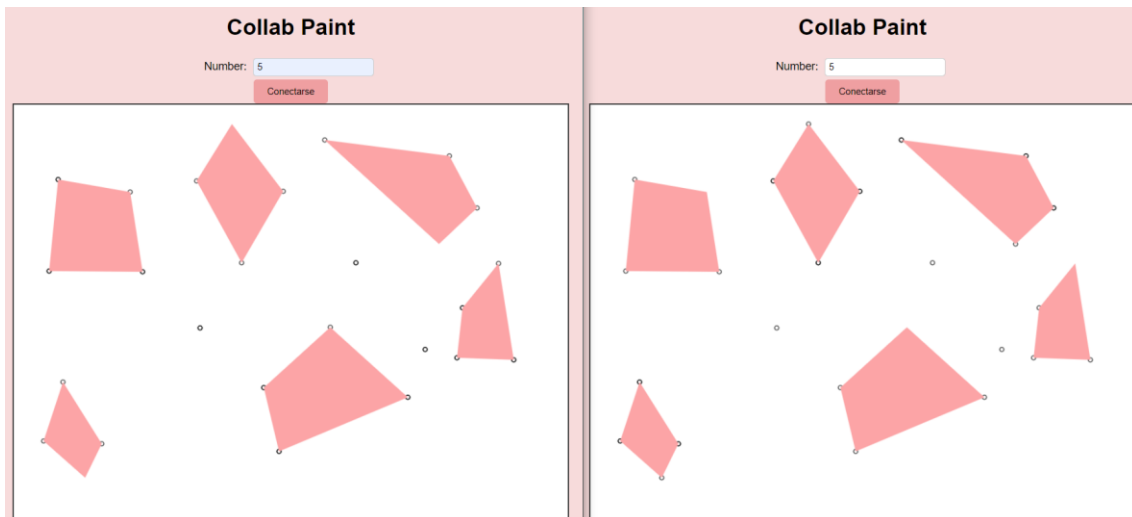
    @Autowired
    SimpMessagingTemplate msgt;

    @MessageMapping("/newpoint.{numdibujo}")
    public void handlePointEvent(Point pt,@DestinationVariable String numdibujo) throws Exception {
        System.out.println("Nuevo punto recibido en el servidor!:"+pt);
        msgt.convertAndSend("/topic/newpoint"+numdibujo, pt);
    }
}
```

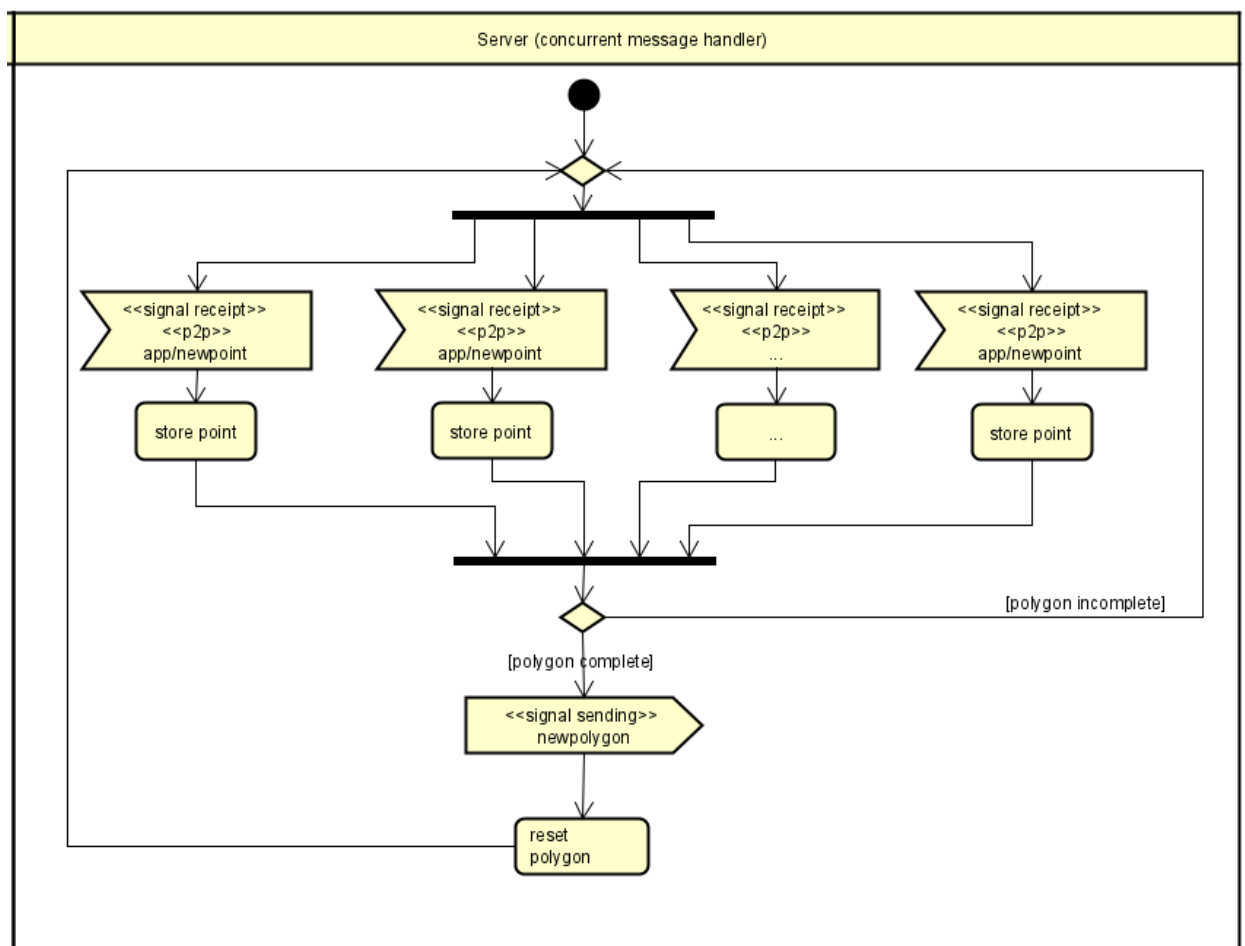
2. Ajuste su cliente para que, en lugar de publicar los puntos en el tópico /topic/newpoint.{numdibujo}, lo haga en /app/newpoint.{numdibujo}. Ejecute de nuevo la aplicación y rectifique que funcione igual, pero ahora mostrando en el servidor los detalles de los puntos recibidos.

```
if (pointsBuffer.length%4 == 0) {
    stompClient.send("/topic/newpolygon." + currentNumber, {}, JSON.stringify(pointsBuffer));
    pointsBuffer = []
} else {
    stompClient.send("/topic/newpoint." + currentNumber, {}, JSON.stringify(pt));
}
```

3. Una vez rectificado el funcionamiento, se quiere aprovechar este 'interceptor' de eventos para cambiar ligeramente la funcionalidad:
 1. Se va a manejar un nuevo tópico llamado '/topic/newpolygon.{numdibujo}', en donde el lugar de puntos, se recibirán objetos javascript que tengan como propiedad un conjunto de puntos.
 2. El manejador de eventos de /app/newpoint.{numdibujo}, además de propagar los puntos a través del tópico '/topic/newpoints', llevará el control de los puntos recibidos(que podrán haber sido dibujados por diferentes clientes). Cuando se completen tres o más puntos, publicará el polígono en el tópico '/topic/newpolygon'. Recuerde que esto se realizará concurrentemente, de manera que REVISE LAS POSIBLES CONDICIONES DE CARRERA!. También tenga en cuenta que desde el manejador de eventos del servidor se tendrán N dibujos independientes!.
 3. El cliente, ahora también se suscribirá al tópico '/topic/newpolygon'. El 'callback' asociado a la recepción de eventos en el mismo debe, con los datos recibidos, dibujar un polígono, tal como se muestran en ese ejemplo.
 4. Verifique la funcionalidad: igual a la anterior, pero ahora dibujando polígonos cada vez que se agreguen cuatro puntos.



4. A partir de los diagramas dados en el archivo ASTAH incluido, haga un nuevo diagrama de actividades correspondiente a lo realizado hasta este punto, teniendo en cuenta el detalle de que ahora se tendrán tópicos dinámicos para manejar diferentes dibujos simultáneamente.



5. Haga commit de lo realizado.
1. git commit -m "PARTE FINAL".

Conclusiones:

1. La implementación de una aplicación de dibujo colaborativo en tiempo real utilizando tecnologías web como STOMP y WebSockets permite la colaboración efectiva entre usuarios, lo que puede ser útil en diversos escenarios, como la educación o la colaboración en diseño.
2. La capacidad de gestionar múltiples dibujos de forma simultánea a través de tópicos dinámicos en STOMP agrega versatilidad a la aplicación y la hace más escalable.
3. La coordinación entre el cliente y el servidor para el manejo de puntos y la generación de polígonos requiere un manejo cuidadoso de posibles condiciones de carrera y sincronización, lo que demuestra la complejidad de las aplicaciones en tiempo real.
4. La implementación de funcionalidades en tiempo real en aplicaciones web es esencial en un mundo cada vez más orientado a la colaboración y la interacción en línea.
5. La documentación y la comprensión de las tecnologías utilizadas, como STOMP, WebSockets, y HTML5 Canvas, son fundamentales para el éxito en el desarrollo de aplicaciones de este tipo.