

BROKER DE MENSAJES STOMP CON WEBSOCKETS + HTML5 CANVAS.

AUTORES:

YORKS GOMEZ – CESAR VÁSQUEZ

PROFESOR:

JAVIER IVAN TOQUICA

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

ARQUITECTURAS DE SOTFWARE - GRUPO # 1

INGENIERÍA DE SISTEMAS

BOGOTÁ D.C.

2023

INTRODUCCIÓN

En este laboratorio trabajaremos con la combinación de websockets y STOMP bidireccionalmente lo cual es especialmente útil en aplicaciones web en las que se requiere una comunicación en tiempo real, como en juegos en línea, sistemas de colaboración, etc.

Además, usaremos HTML5 Canvas para que los clientes puedan visualizar de forma más dinámica, suele ser útil en aplicaciones que necesitan mostrar datos en tiempo real de una manera más visual, como en monitoreo de datos en tiempo real, gráficos y tableros de control.

Intentaremos usar un broker de mensajes STOMP con websockets la cual es una solución escalable para aplicaciones que necesitan manejar grandes cantidades de mensajes en tiempo real. Al utilizar un broker de mensajes, se puede delegar la tarea de distribuir los mensajes a través de múltiples servidores, lo que puede mejorar la capacidad de procesamiento y reducir la carga en un solo servidor.

DESARROLLO

PARTE I

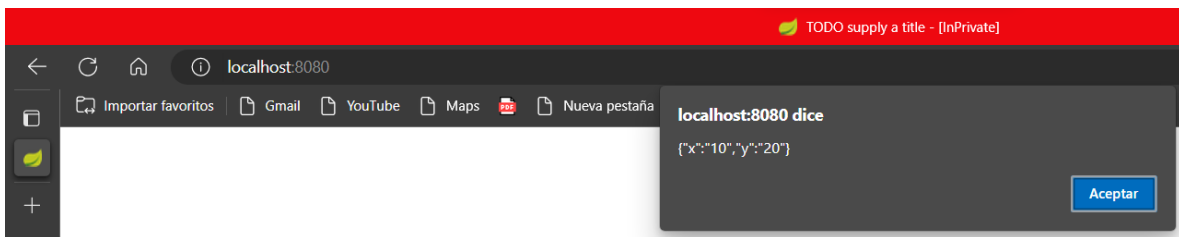
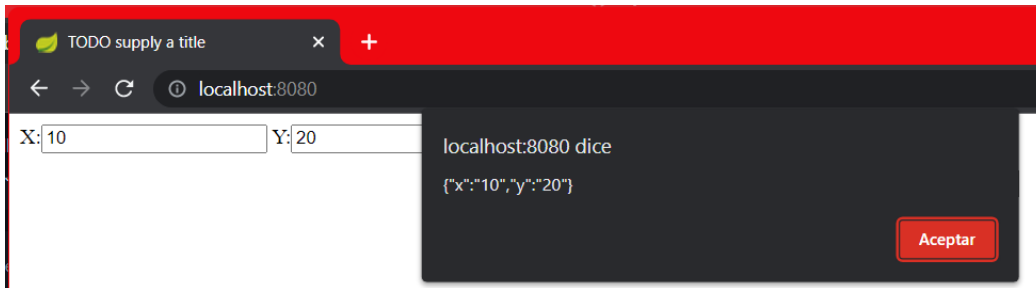
Hacemos que la aplicación HTML5/JS al ingresarle en los campos de X y Y, además de graficarlos, los publicamos en el tópico: /topic/newpoint.

Dentro del módulo JavaScript modificamos la función de conexión/suscripción al WebSocket, para que la aplicación se suscriba al tópico "/topic/newpoint". Asociamos como 'callback' de este suscriptor una función que muestre en un mensaje de alerta el evento recibido. Como se sabe que en el tópico indicado se publicarán sólo puntos, extraemos el contenido enviado con el evento (objeto JavaScript en versión de texto), convirtiéndolo en objeto JSON, y extraemos de éste sus propiedades (coordenadas X y Y).

```
var connectAndSubscribe = function () {  
  const (local var) socket: any = ...;  
  var socket = new SockJS('/stompendpoint');  
  stompClient = Stomp.over(socket);  
  
  //subscribe to /topic/newpoint when connections succeed  
  stompClient.connect({}, function (frame) {  
    console.log('Connected: ' + frame);  
    stompClient.subscribe('/topic/newpoint', function (eventbody) {  
      alert(eventbody.body);  
      let points = JSON.parse(eventbody.body);  
    });  
  });  
};
```

```
publishPoint: function(){  
  let px = document.getElementById("x").value;  
  let py = document.getElementById("y").value;  
  var pt = new Point(px,py);  
  console.info("publishing point at (" + pt.x + ", " + pt.y + ")");  
  
  //publicar el evento  
  stompClient.send("/topic/newpoint", {}, JSON.stringify(pt));  
},
```

Compilamos y ejecutamos nuestra aplicación. Abrimos la aplicación en varias pestañas diferentes. Ingresamos los datos, damos click a la acción del botón, y verificamos que en todas la pestañas se haya lanzado la alerta con los datos ingresados.



PARTE II

Hacemos que el 'callback' asociado al tópico /topic/newpoint en lugar de mostrar una alerta, dibuje un punto en el canvas en las coordenadas enviadas con los eventos recibidos.

Dibujamos un círculo de radio 1.

```
var connectAndSubscribe = function () {
  console.info('Connecting to WS...');
  var socket = new SockJS('/stompendpoint');
  stompClient = Stomp.over(socket);

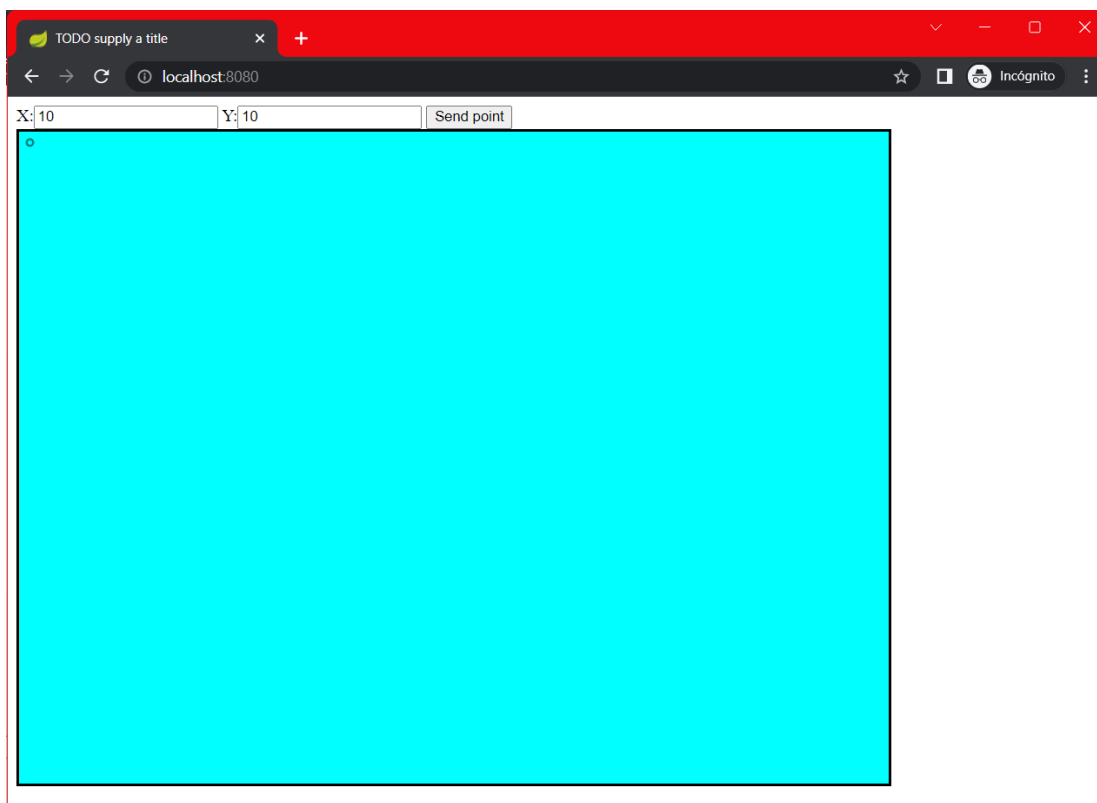
  //subscribe to /topic/newpoint when connections succeed
  stompClient.connect({}, function (frame) {
    console.log('Connected: ' + frame);
    stompClient.subscribe('/topic/newpoint', function (eventbody) {
      //alert(eventbody.body);
      let point = JSON.parse(eventbody.body);
      var pt=new Point(point.x, point.y);
      addPointToCanvas(pt);
    });
  });
};
```

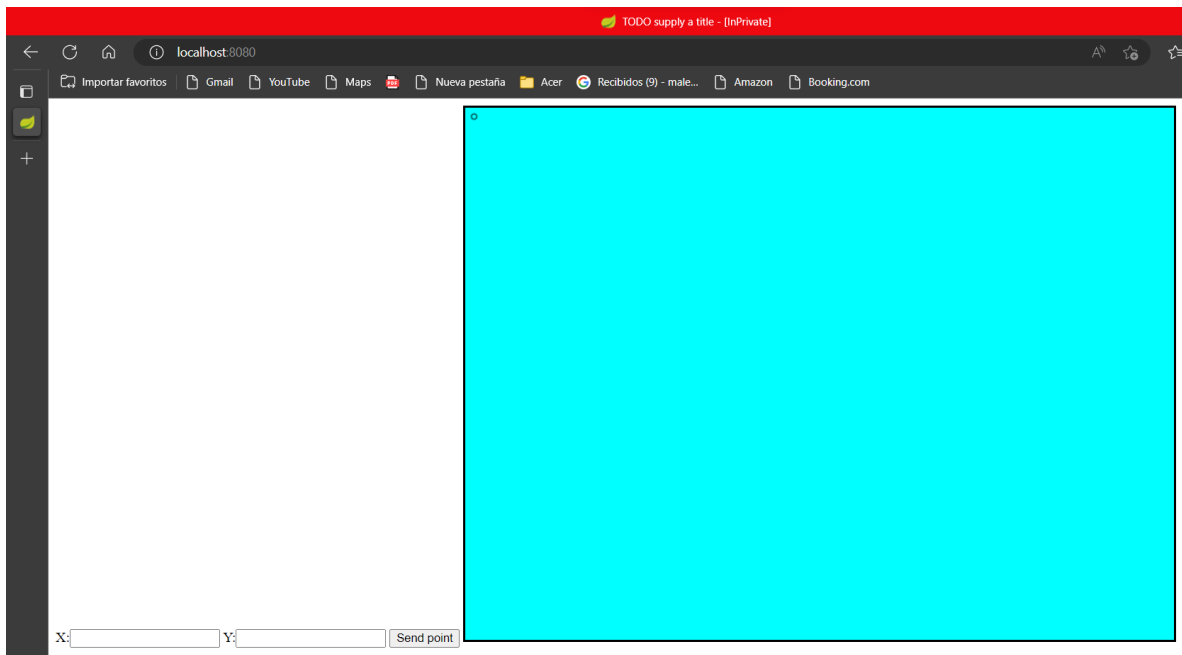
```
publishPoint: function(x,y){
    var pt = new Point(x, y);
    console.info("publishing point at (" + pt.x + ", " + pt.y + ")");

    //publicar el evento
    stompClient.send("/topic/newpoint", {}, JSON.stringify(pt));
},
```

```
<body onload="app.init()">
  <div>
    <label> X:</label><input id="x" type="number"/>
    <label> Y:</label><input id="y" type="number"/>
    <button onclick="app.publishPoint(document.getElementById('x').value,document.getElementById('y').value)">Send point</but
    <canvas id="canvas" width="800" height="600"></canvas>
  </div>
```

Ejecutamos la aplicación en varios navegadores (y si puede en varios computadores, accediendo a la aplicación 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 aplicación.

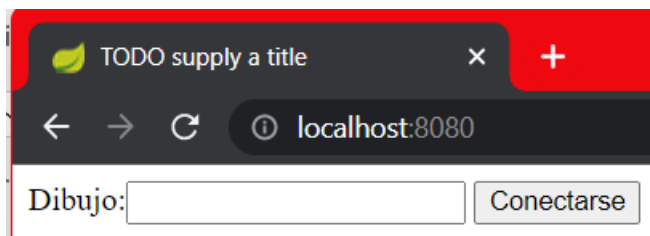




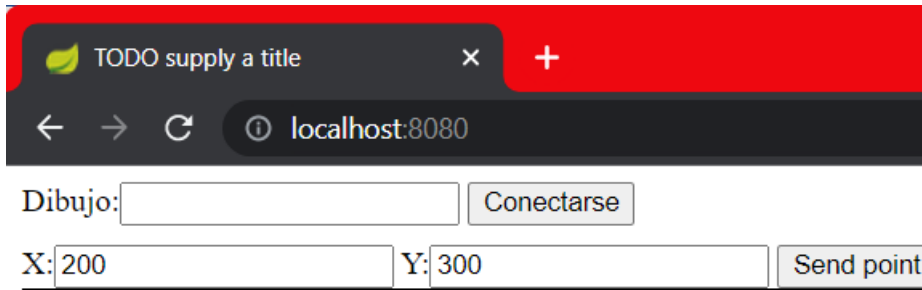
Efectivamente se dibuja de manera correcta en los diferentes navegadores.

PARTE III

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



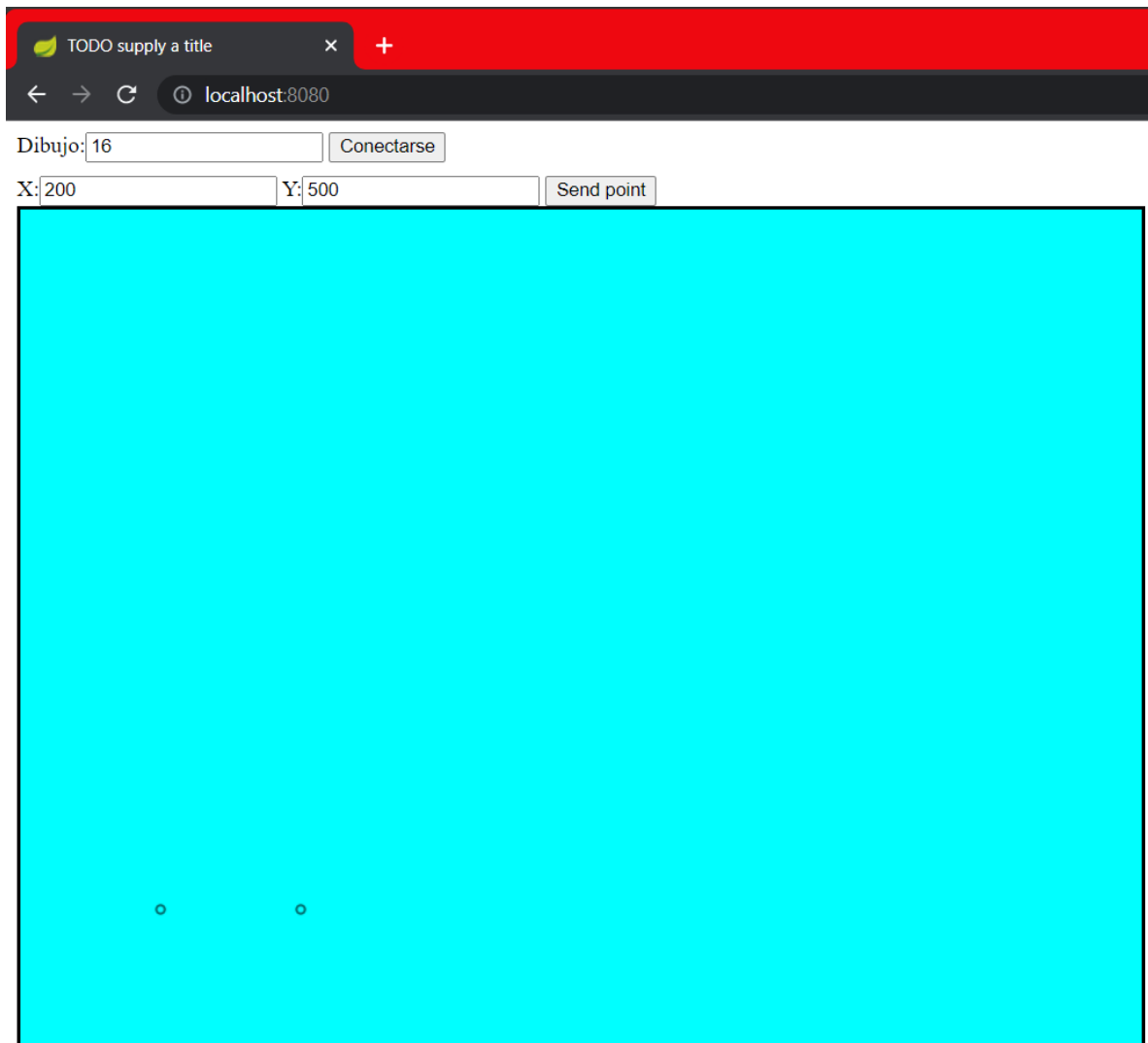
Modificamos la aplicación para que, en lugar de conectarse y suscribirse automáticamente, lo haga a través de botón 'conectarse'. Éste, al oprimirse debe realizar la conexión y suscribir al cliente a un tópico que tenga un nombre dinámico, asociado el identificador ingresado, por ejemplo: /topic/newpoint.25,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



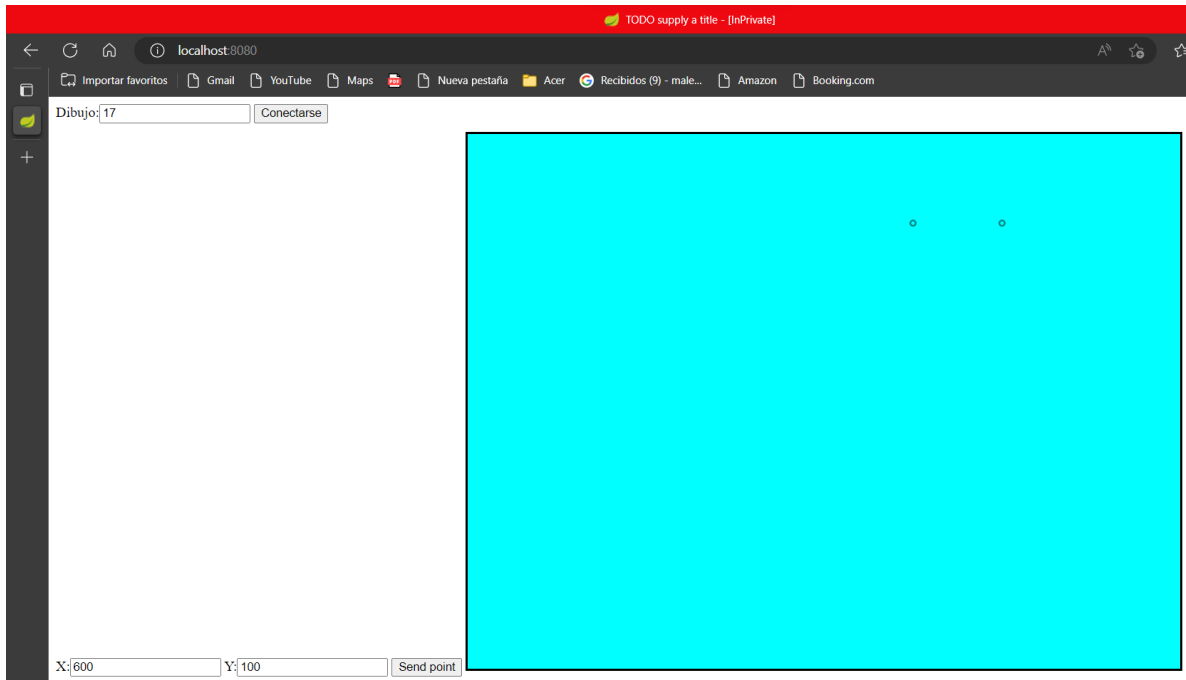
A screenshot of a web browser window. The address bar shows 'localhost:8080'. Below the address bar, there is a form with the following elements: a label 'Dibujo:' followed by an empty text input field, a 'Conectarse' button, a label 'X:' followed by a text input field containing '200', a label 'Y:' followed by a text input field containing '300', and a 'Send point' button.

De la misma manera, hacemos que las publicaciones se realicen al tópico asociado al identificador ingresado por el usuario.

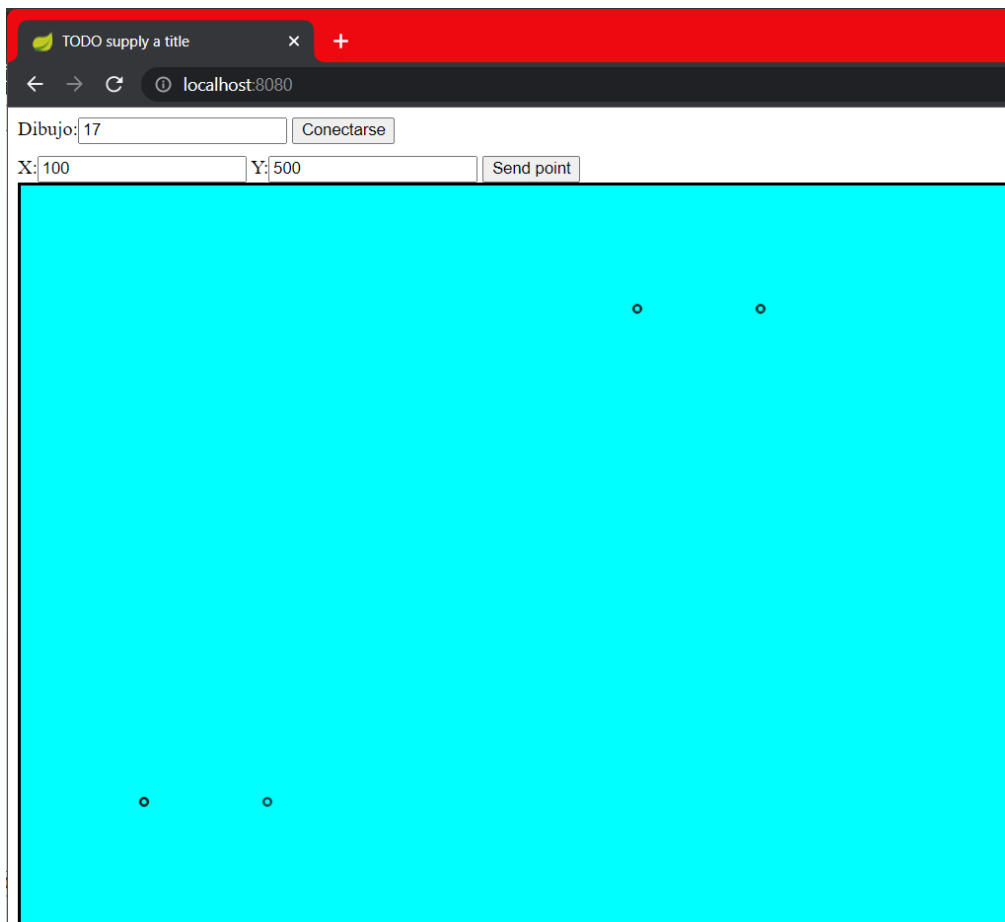
Rectificamos que se puedan realizar dos dibujos de forma independiente, cada uno de éstos entre dos o más clientes.



A screenshot of a web browser window. The address bar shows 'localhost:8080'. Below the address bar, there is a form with the following elements: a label 'Dibujo:' followed by a text input field containing '16', a 'Conectarse' button, a label 'X:' followed by a text input field containing '200', a label 'Y:' followed by a text input field containing '500', and a 'Send point' button. Below the form, there is a large cyan rectangular area representing a drawing canvas. At the bottom left of this canvas, there are two small black dots.

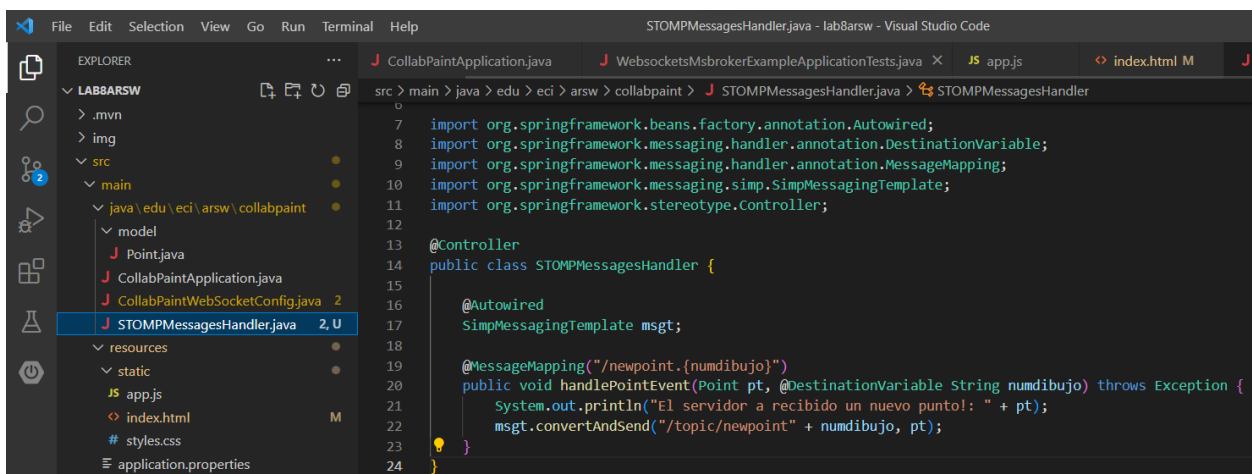


Ahora realizamos la conexión entre los dos y verificamos que se haga de manera correcta.



PARTE IV

Creamos la nueva clase controladora **STOMPMessagesHandler**

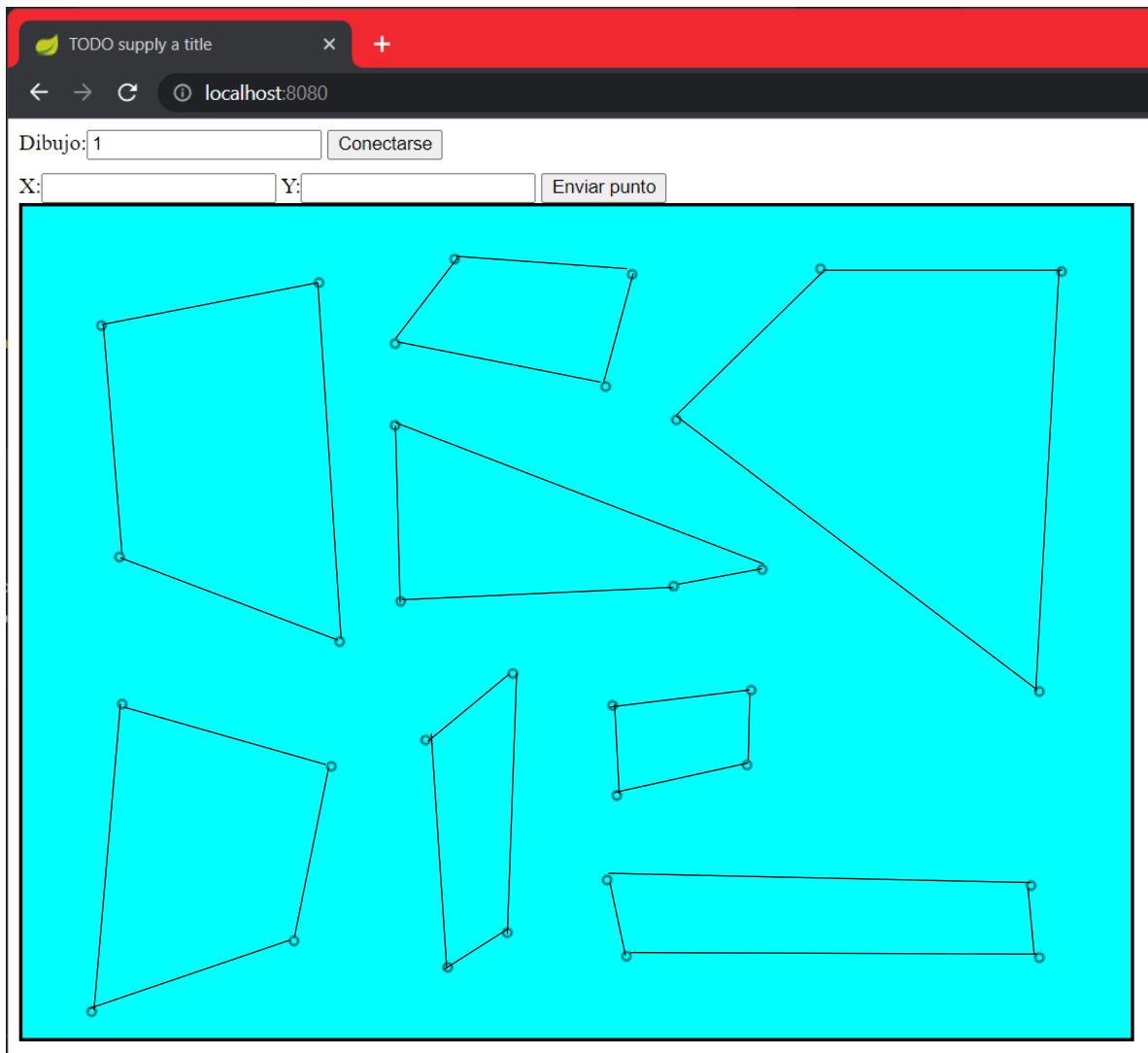


Ajustamos el cliente para que, en lugar de publicar los puntos en el t pico

/topic/newpoint.{numdibujo}, lo haga en /app/newpoint.{numdibujo}.



El cliente, ahora tambi n se suscribir  al t pico '/topic/newpolygon'. El 'callback' asociado a la recepci n de eventos en el mismo debe, con los datos recibidos, dibujar un pol gono.



Efectivamente se pintan los polígonos de forma correcta.

DIAGRAMA DE ACTIVIDADES

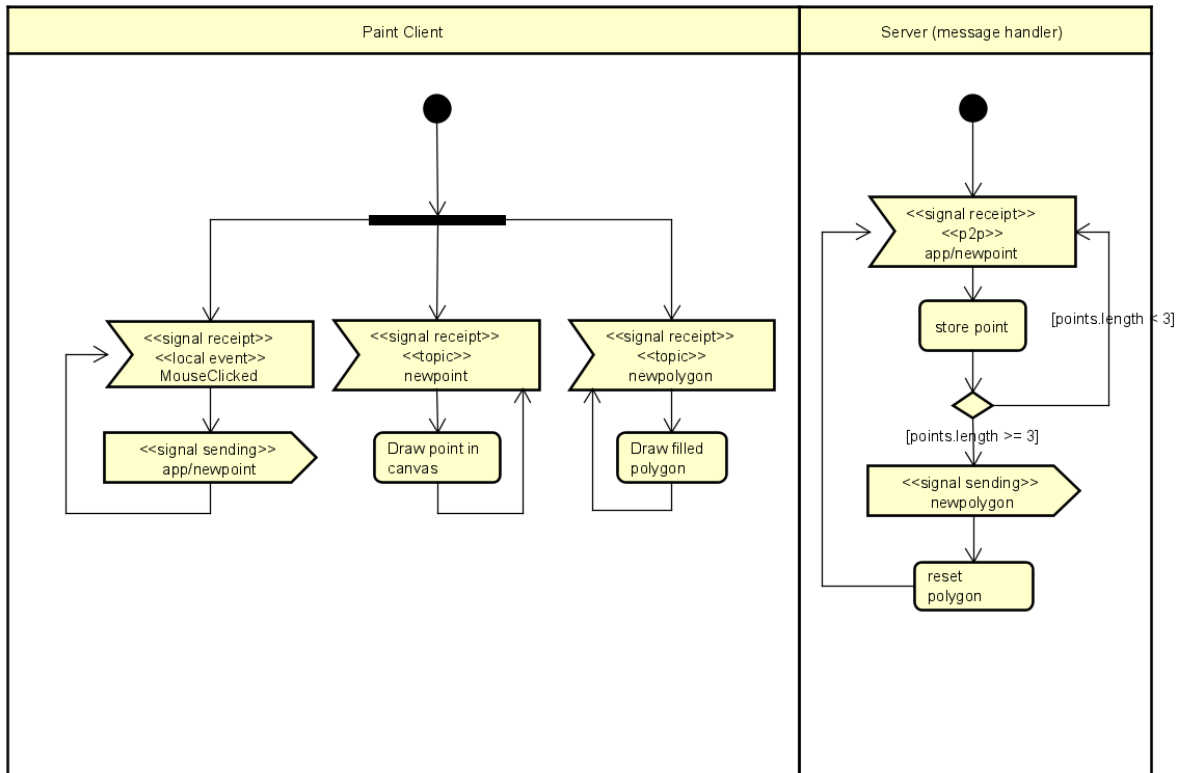
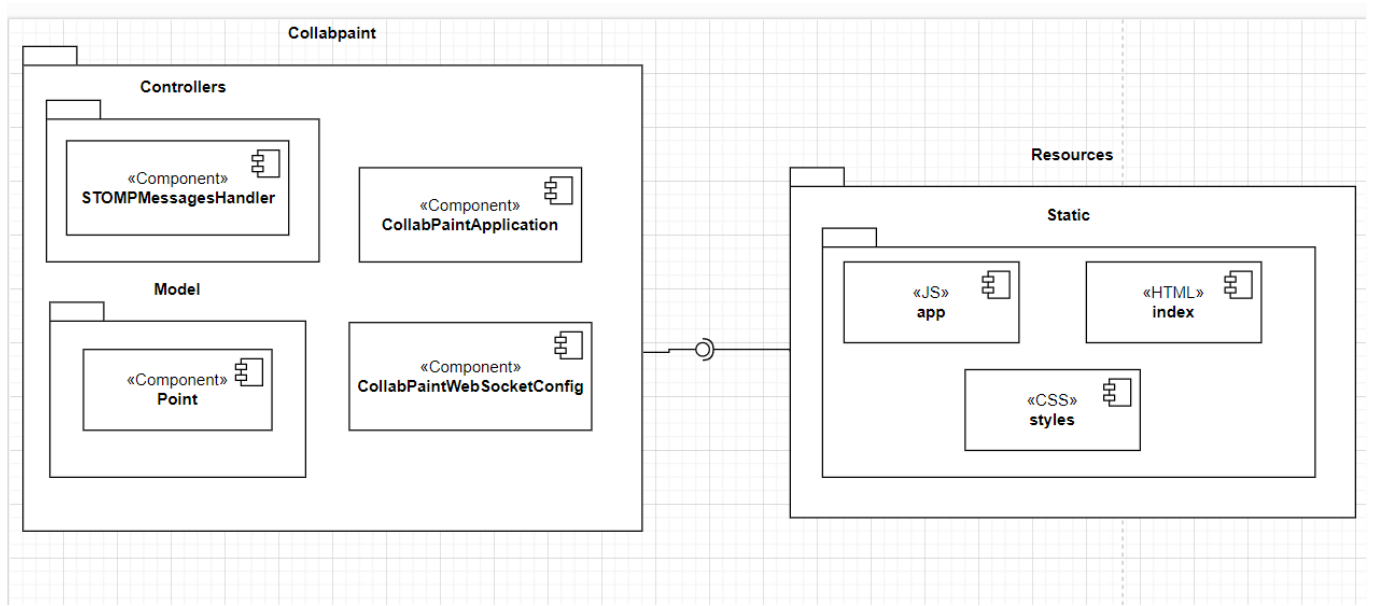


DIAGRAMA DE COMPONENTES



CONCLUSIONES

- Aprendimos de una mejor forma el uso de comunicación en tiempo real, la combinación de websockets y STOMP nos permitió una comunicación bidireccional en tiempo real entre el cliente y el servidor.
- Entendimos el proceso de visualización de datos para observar la capacidad de enviar datos en tiempo real desde el servidor al cliente y visualizarlos en un lienzo de HTML5 Canvas.
- Aprendimos el uso del protocolo STOMP el cual trata a cerca de mensajería interoperable que puede ser utilizado por diferentes sistemas. Por lo tanto, el uso de un broker de mensajes STOMP con websockets puede permitir la integración de diferentes sistemas que necesitan comunicarse entre sí en tiempo real.
- Fue interesante para nosotros aplicar las herramientas de forma correcta, ya que debemos tener en cuenta este laboratorio para la elaboración del proyecto de la materia.