

**Ingeniería en Sistemas**

**Computacionales**

**Raúl Tostado Blázquez I.D. 145901**

**Materia: Sistemas Distribuidos**

**Nombre del Trabajo: Práctica Websocket-Play**

**Profesora: Ofelia D. Cervantes Villagómez**

**San Andrés Cholula, Puebla a**

**29 de abril de 2015**

Reporte práctica Websocket-play

**Introducción a la práctica**

Esta es una simple aplicación que implementa un chat anónimo, utilizando webSockets, además se ilustra el uso de actores con Akka y Play Framework. La especificación WebSocket define un API que establece conexiones "socket" entre un navegador web y un servidor. Dicho con otras palabras: existe una conexión persistente entre el cliente y el servidor, y ambas partes pueden empezar a enviar datos en cualquier momento.

**Antes de comenzar:**

El primer paso fue dar una lectura detallada a la práctica para saber cuál es el contenido de la misma, posteriormente se descargaron como Zip los programas con los que se trabajó y otros complementos necesarios para trabajar.

**Actividad 1 - Análisis**

**WebSocket** es un protocolo que provee canales de comunicación bidireccional sobre una conexión TCP única. Desde el 2011 el protocolo es un estándar y la API Web está en proceso de estandarizarse por la W3C. Los **WebSockets** utilizan un único canal de comunicación bidireaccinal en donde se asegura una alta velocidad de intercambio de datos en demanda.

**Estructura de la Aplicación**

La **aplicación** se encuentra desarrollada en Scala y JavaScript. La naturaleza de **Play Framework** nos permite mantener un orden en la estructura de la aplicación. El uso del patrón **Modelo Vista Controlador** es muy claro, puesto que Play separa las características de la **vista** (HTML, CSS), de la implementación de los **WebSockets**. Cabe mencionar que debido a la naturaleza de Play para desarrollar fácilmente métodos REST a partir del controlador es muy fácil desarrollar servicios como WebSocket. A continuación se muestra la arquitectura de la aplicación. Ahora que has observado las características de la aplicación, explica el resultado de tu análisis ampliamente en tu reporte respondiendo a las siguientes preguntas.

* **Analizar** la estructura del proyecto en Play Framework.
  + ¿Cuál es el objetivo de la carpeta **App**?
  + ¿Cuál es el objetivo del patrón **Modelo Vista Controlador**?
* **Analizar** la arquitectura Cliente-Servidor del proyecto.
  + **Identificar** el Cliente y explicar su objetivo.
  + **Identificar** el Servidor y explicar su objetivo.
* ¿Cómo se relacionan conf/routes y app/controllers/Application.scala?
* **Analizar** la vista (carpeta Views en Play)
  + ¿Cuál es la utilidad del atributo **data-ws-url**?

R= Es donde se va a crear el nuevo websocket, en cuanto a usar al patrón MVC nos ayuda a separar los componentes de la aplicación y así sean más fáciles de utilizar.

A continuación se listan algunas tecnologías utilizadas en este proyecto, que no son explotadas en su totalidad pero son interesantes de analizar y considerar para futuros desarrollos:

* [**Bootstrap**](http://getbootstrap.com/) es un framework para el **front-end**, es una colección de herramientas para desarrollar sitios Web responsivos (tanto para dispositivos móviles como para escritorio). **Bootstrap** incluye templates en HTML y CSS para agregar componentes, como botones, formularios, tipografías y componentes de navegación que pueden utilizarse instantáneamente.
* [**Akka**](http://akka.io/) Es un toolkit open-source para simplificar la construcción de aplicaciones concurrentes y distribuídas bajo la JVM. **Akka** soporta diferentes modelos de programación para implementar concurrencia, especialmente el modelo basado en actores.
* [**Scala**](http://www.scala-lang.org/) es un lenguaje de programación multi-paradigma diseñado para expresar patrones comunes de programación en forma concisa, elegante y con tipos seguros. Integra sutilmente características de lenguajes funcionales y orientados a objetos. La implementación actual corre en la máquina virtual de Java y es compatible con las aplicaciones Java existentes.

**Actividad 2 - Cliente-WebSocket**

Ahora que hemos analizado y comprendido la estructura de la aplicación es momento de implementar los **WebSockets**, por ahora no debes preocuparte por los actores. Analizaremos el modelo de actores en una actividad posterior.

El objetivo de la Actividad 2 es comprender la implementación de **WebSockets** desde el lado del Cliente, es decir, cómo se envían mensajes a un Servidor y su implementación en JavaScript. Desarrollando un pequeño **Chat Anónimo**.

Como podrás observar en la carpeta **assets/javascripts** se encuentra un archivo **index.js** el cuál se encuentra vacío. Como habrás notado en tu análisis, ya se encuentra (mayormente) desarrollado el código del WebSocket en Scala, de hecho, si observas el archivo conf/routes podrás observar como se ha definido un servicio REST de la siguiente manera:

GET /ws controllers.Application.ws

Este es el Servicio del **WebSocket**, listo para ser implementado en el cliente, por lo que podemos comenzar a utilizarlo directamente en nuestra aplicación. Para comenzar estableceremos el canal de comunicación entre el cliente y el servidor. Para esto implementa el código que se muestra a continuación dentro de **assets/javascripts/index.js**

**Recibiendo Datos**

var ws;

ws = new WebSocket($("body").data("ws-url")); //Se toma La URL del Tag en Body

ws.onmessage = function(event) {

var message;

message = JSON.parse(event.data);

switch (message.type) {

case "message":

return $("#board tbody").append("<tr><td>"+ message.msg + "</td></tr>");

default:

return console.log(message);

}

};

El fragmento de código anterior realiza una conexión al **WebSocket** definido en el servidor, cada vez que el servidor envía información, se muestra en el cliente como una fila dentro de una tabla.

Ahora enviemos datos a través del formulario, de igual forma copia el siguiente fragmento de código.

**Enviando Datos**

$("#msgform").submit(function(event) {

event.preventDefault();

console.log($("#msgtext").val());

ws.send(JSON.stringify({ //Enviar el mensaje en el formulario como JSON.

msg: $("#msgtext").val()

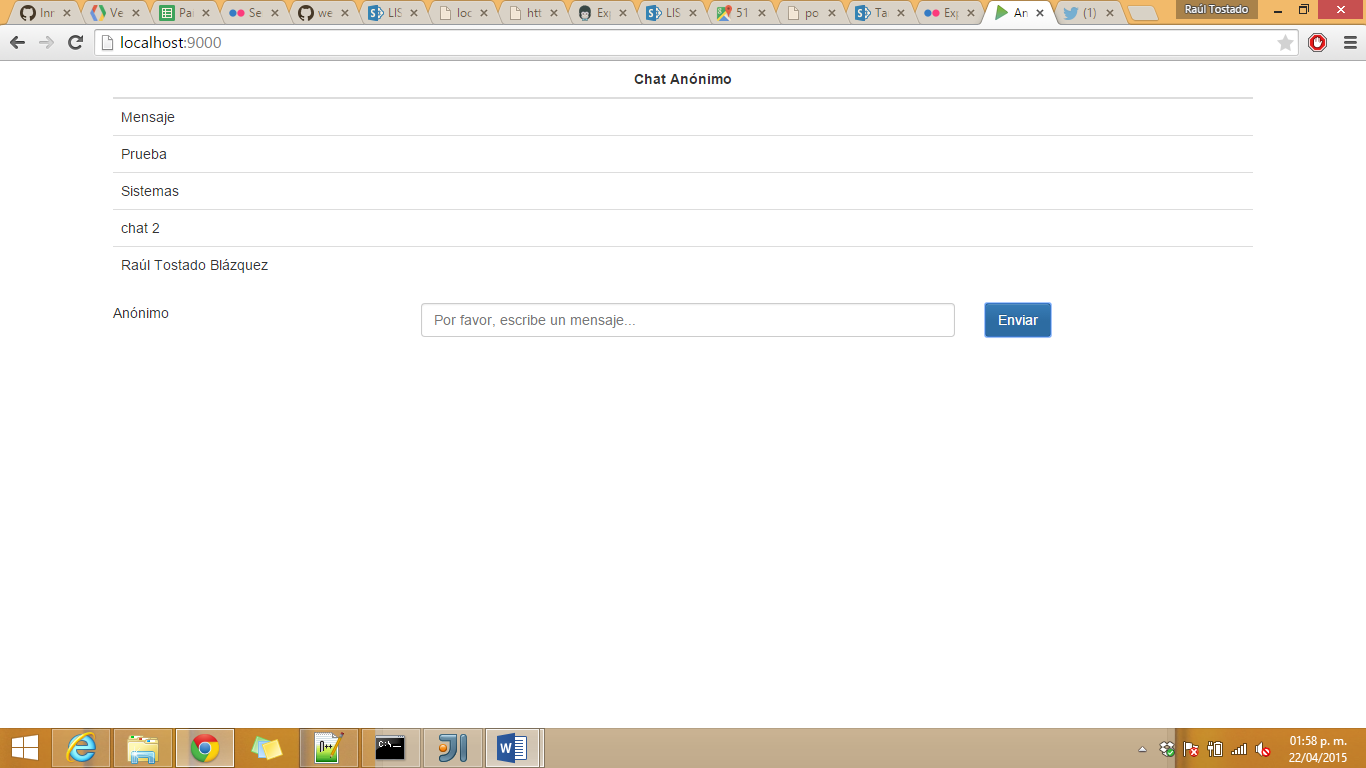
}));

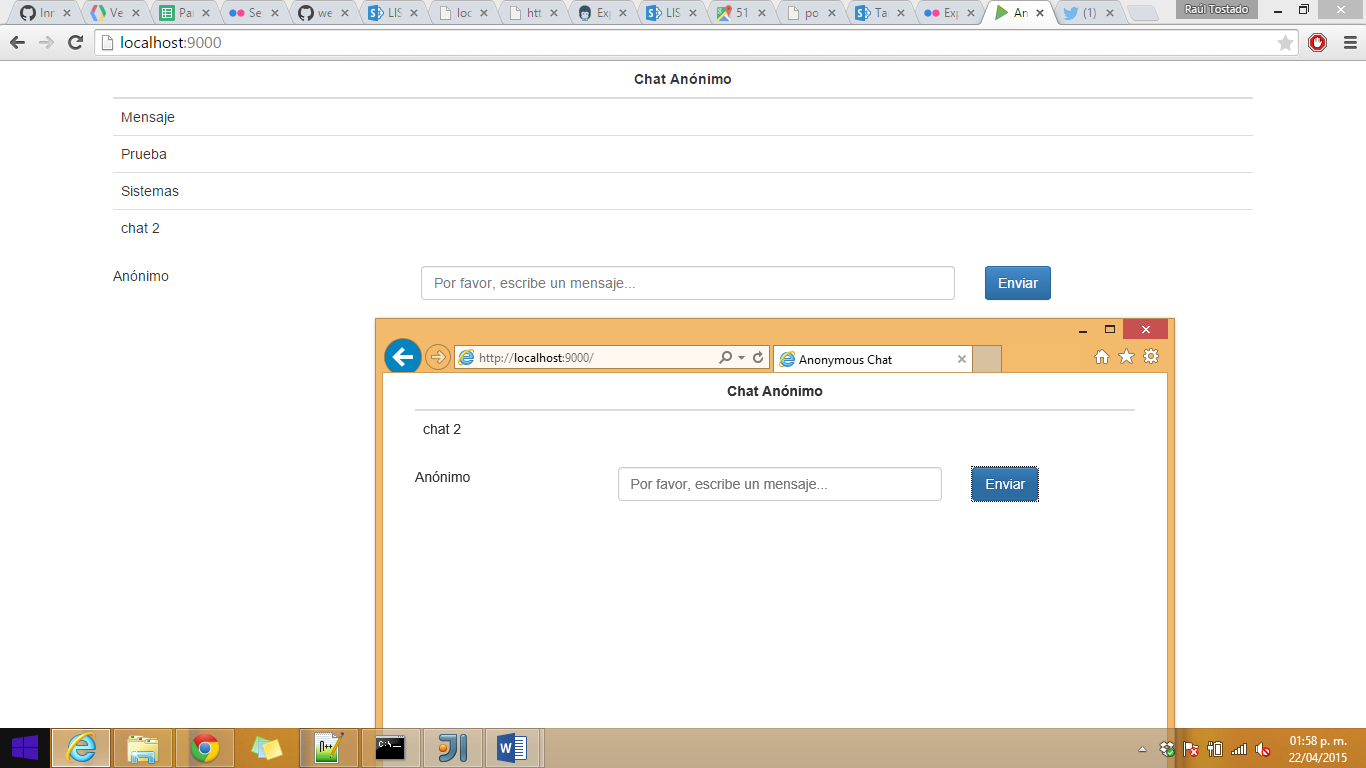
return $("#msgtext").val("");

});

Listo, ahora deberías poder comunicarte con el **WebSocket**, pruébalo ahora envíando un mensaje desde el formulario de la interfaz y haciendo click en el botón **Enviar** ¿Qué es lo que pasa aquí?, la siguiente imagen ilustra las acciones del **Websocket:**

Por el momento el **Chat Anónimo** lo puedes usar al mismo tiempo desde diferentes navegadores o pestañas para experimentar como el WebSocket atiende las peticiones de todos los clientes.





R= Como se puede ver en las imágenes anteriores, ponemos a funcionar el chat, si se abre en navegadores diferentes, actúa como si fueran dos clientes totalmente diferentes, por lo cual podemos ver la frase chat 2 en ambas ventanas.

**Envíar datos**

Ahora te toca a tí, dado lo que has aprendido con WebSockets, deberás simular a otra persona enviándo mensajes:

* **Desarrollar** una función JavaScript, que permita enviar mensajes a través del WebSocket al servidor cada 5 segundos, puedes extraer los mensajes de este arreglo:

var speech = ["Hello there!","Welcome to the world of Pokémon!",

"My name is Oak!","People call me the Pokémon Prof!",

"This world is inhabited by creatures called Pokémon!",

"For some people, Pokémon are pets",

"Other use them for fights",

"Myself… I study Pokémon as a profession",

"First, what is your name?"];

* La función **window.setInterval** podría ser de utilidad:

window.setInterval(function(){

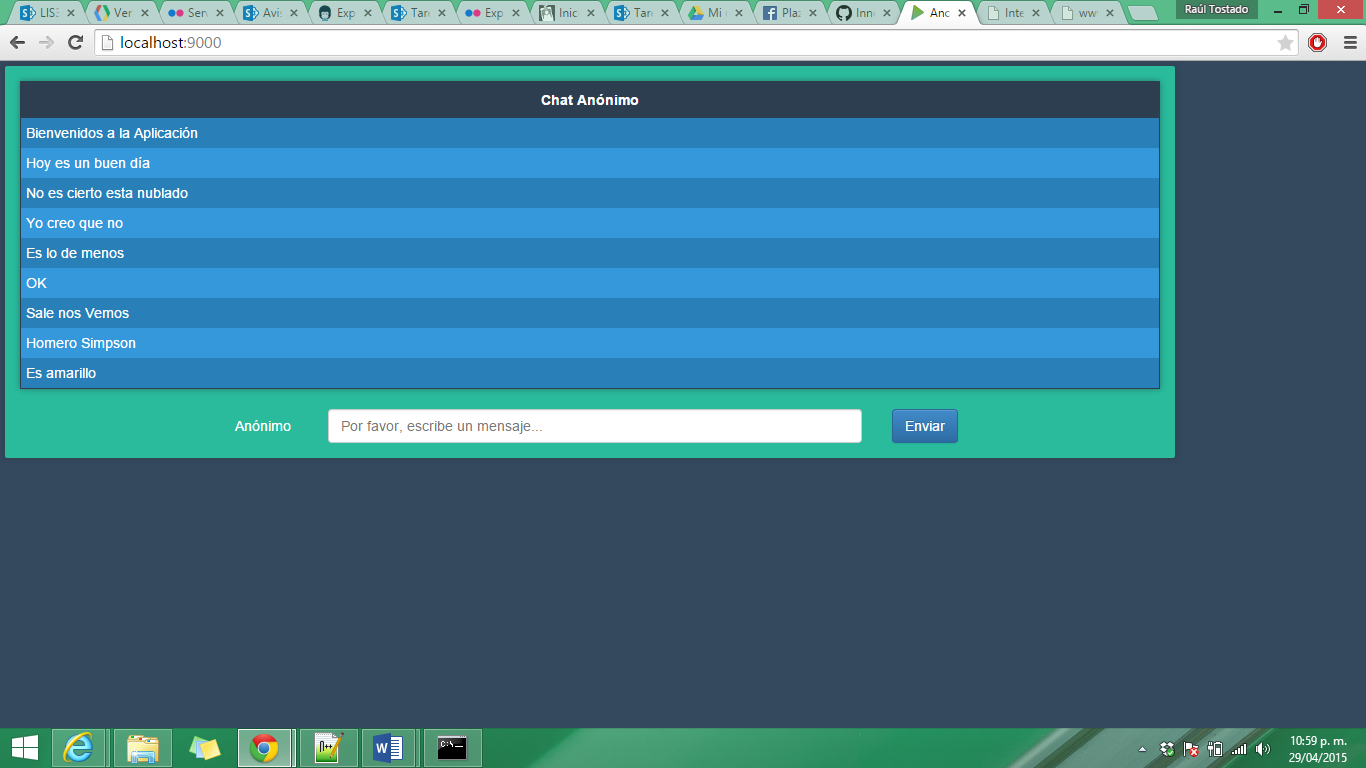
/\*

\* Código Websocket

\*/

}, 4000); // Tiempo en milisegundos

* **Desarrollar** la interfaz de usuario de la aplicación (Con CSS y Html)
  + La aplicación final debería verse más o menos así:
* **Explicar** ampliamente en el reporte los pasos que seguiste para el desarrollo de la aplicación.



R= En la imagen anterior, podemos ver el resultado final del chat, se añaden los elementos de la hoja de estilo, también se creó un nuevo diálogo el cual cada cierto intervalo de tiempo, nos manda mensaje por mensaje, el resultado del chat puede ser visto por todas las personas que estén conectados al mismo, así como los cambios que va a efectuar.

**Actividad 4 - Limpieza**

En el desarrollo de aplicaciones Web es importante mantener un estilo al programar, por esto deberás:

* **Nombrar** adecuadamente las variables y métodos.
* **Agregar** un comentario antes de un método para indicar su funcionamiento.
* **Comentar** las líneas de código que podrían ser no muy claras.
* **Realizar** un identado adecuado del código.
* **Revisar** el código para detectar usos inadecuados de variables ó errores al escribir.

Recordando que el estilo debe ser siguiendo las guías de [Google](https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml) para **JavaScript**.

**Actividad 5 - Preguntas**

Responder ampliamente a las siguientes preguntas en el reporte, tomando en cuenta todas las actividades realizadas en este repositorio.

* ¿Qué es **WebSocket**?

R= Es un protocolo que provee canales de comunicación bidireccional sobre una conexión TCP única. Hay respuesta en el momento exacto, no hay tiempo de espera.

* ¿Cuál es la diferencia entre **WebSocket** y **Servicios REST**?

R= El servicio REST es una estructura muy cerrada a cliente servidor, es decir, se manda una solicitud, hay una espera breve y nos manda una respuesta y no es necesario actualizar nuevamente la página. Websocket lo principal es que la comunicación es bidireccional, es decir en el momento en que se escribe algo en el chat, es visible por todos los usuarios conectados al chat y se actualiza constantemente.

* ¿Cómo se establece una conexión con **WebSocket**?

R= Se va a llamar con el método GET desde el index de la aplicación.

* Explica la integración de datos en la interfaz utilizando **JavaScript**.

R= Definir los lugares donde va a ir la información generada en la aplicación asignando un id a cada uno de los objetos con los que vamos a trabajar.

* ¿Cómo se implementa el **patrón MVC** en esta aplicación?
  + ¿Qué se hace en el **controlador**?
  + ¿Cómo se implementa la **vista**?

R= Similar a las prácticas anteriores, nos ayuda el patrón a poder separar los componentes del programa para que sea más fácil entender uno a uno

* Explica el comportamiento del WebSocket cuando envías mensajes cada 5 segundos al servidor.

R= La página se mantiene y dado el intervalo aparece un nuevo mensaje en todos los clientes que estén conectados al servidor, en este caso los mensajes los genera en automático el servidor.

**Conclusiones**

Durante la práctica aprendimos a usar websockets, un tema muy interesante e importante ahora que empezamos a trabajar con un chat, fuimos capaces de observar cómo se va actualizando en todas las ventanas, un elemento muy importante y necesario en aplicaciones posteriores. También hay que ser cuidadosos en como lo implementamos para que no exista algún error al ejecutar.