Laboratorio 1

Tecnólogo en Informática | *Java EE*

Esteban Leivas
Luis Pagola
Luciano Porta

Estudio previo

Base de datos relacional

MySQL

Si bien la decisión de utilizar MySQL no la tomamos nosotros ya que este gestor de base de datos era un requisito del proyecto, tiene sentido usarlo por su escalabilidad y flexibilidad. Como todas las tecnologías que estamos usando, es opensource y gratis. Además, dispone gran compatibilidad con las tecnologías que vamos a trabajar en este proyecto y tiene una buena documentación propia y un soporte enorme en Internet.

Simulación de ventiladores

PHP

Dado que el propósito de este software va a ser una especie de stub que vendría a proveer los datos que los ventiladores mecánicos enviarian si estuviéramos en un caso real, acordamos utilizar PHP porque es un lenguaje fácil de desarrollar y tiene una excelente integración con MySQL y soporte de la comunidad.

Plataforma de desarrollo

Java EE

Esta parte también era un requisito propio del proyecto, sin embargo, destacamos que Java EE es la integración de un grupo de especificaciones para construir software empresarial en Java. Dentro de estas especificaciones, encontramos soluciones enfocadas a solucionar desarrollo de aplicaciones web, persistencia de datos, construcción API basadas en REST o WebSockets, entre otras. Es de código libre y gratuita. A pesar de que Oracle ya no se hace cargo de esta tecnología, su gestión y desarrollo se ha pasado a The Eclipse Foundation quienes con la versión 8 de la plataforma han incluido muchas mejoras como soporte a entornos en la nube y arquitectura de microservicios. Además, Java EE ha incorporado tecnologías fuera de Java como HTML 5 y HTTP/2. Otra ventaja que incluye Java EE es la de simplificar la plataforma Java al incluir anotaciones, inyección de dependencias y facilitando la configuración y flexibilidad con archivos XML.

Servidor de aplicaciones

GlassFish

Es una implementación de servidor para Java EE desarrollada por Oracle. De hecho, la mayoría de las otras opciones que se encuentran disponibles utilizan las implementaciones de GlassFish como ejemplo, por lo tanto sabemos que es una buena solución en cuanto a funciones y compatibilidad con las últimas versiones de Java EE. También viene incluida

con NetBeans lo cuál nos viene bien ya que será la IDE que utilizaremos para este proyecto.

Framework

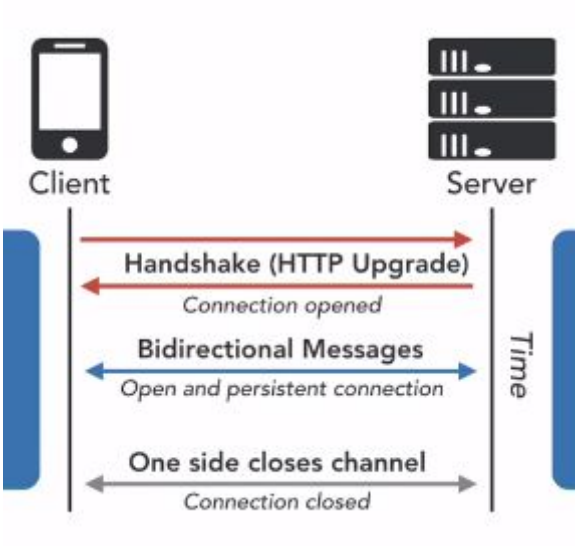
Spring

Spring es un framework para construir aplicaciones web que utiliza el patrón de diseño MVC (Model - View - Controller).

Protocolo de comunicación

WebSocket

Aplicaciones en tiempo real están implementadas usando WebSocket, una tecnología introducida a HTML en 2011. Esta solución alivia varios problemas al momento de desarrollar aplicaciones que requieren interactividad en tiempo real entre cliente y servidor web. Cada solicitud XHR viene con headers de HTTP lo cual supone un gasto importante de recursos. Con la introducción de WebSockets, ahora se tiene una conexión persistente y bidireccional.



Además, dicha conexión no necesita estar enviando headers de forma innecesaria.

Por estas razones, decidimos utilizar esta tecnología para transmitir los datos de las VeMecs en tiempo real.

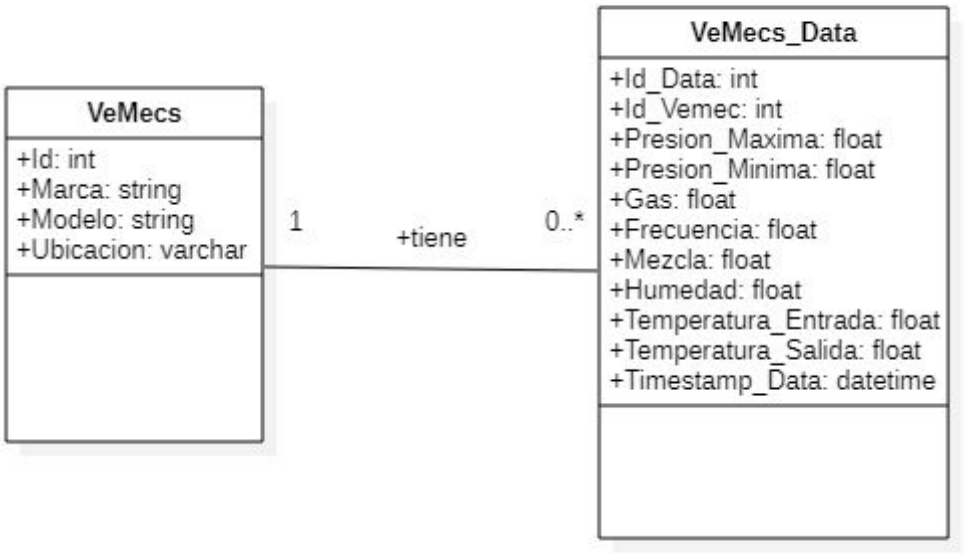
Protocolo de comunicación

REST

REpresentational State Transfer (REST) es un estilo arquitectonico para proveer estandares entre servidores y clientes haciendo que sea más facil comunicar sistemas entre ellos. Proveé una serie de comandos comúnmente usados en transacciones web y como al contrario de WebSocket no tiene conexiones persistentes, pensamos utilizarlo para tareas sencillas como generar altas de VeMecs, por ejemplo.

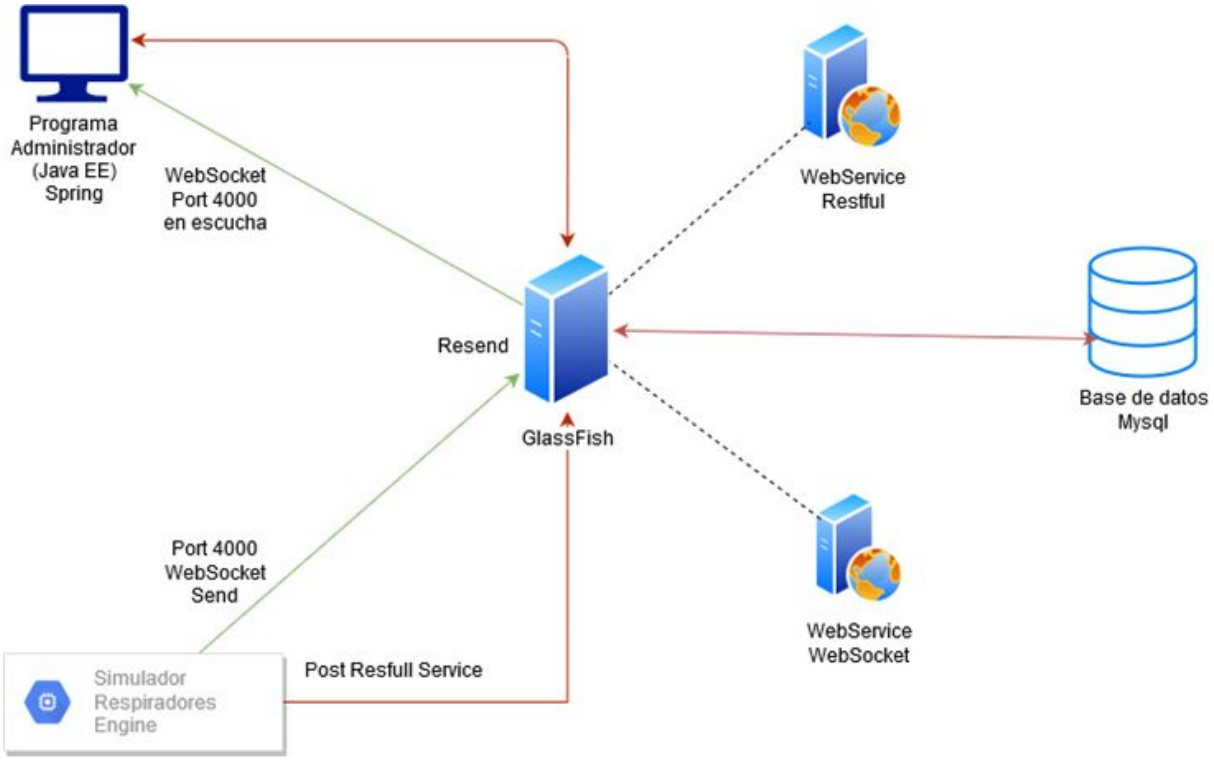
Documentación

Diagrama de clases



Implementación

Simulador



El simulador de respirador, el cual cuenta con varias instancias, envía la información de forma constante por medio de un servicio websocket al servidor por el port 4000, luego el servidor se encarga de hacer un resend al mismo port de los demás clientes conectados al mismo, con la id del simulador en su nombre (ej: datosVeMec4).

Para un manejo más eficiente utilizamos una api llamada *socket.io*, la cual está instalada en todos los clientes y en el servidor, de esta manera se genera algo así como “subcanales con un nombre específico” (ej: envio_datosVeMec) por lo que el programa administrador puede quedarse en escucha de solo datos que nos interesen, le llegara toda la información solo al canal que tenga de nombre su id y recibir así toda la data permitiendo al programa de administración mostrar esta información en tiempo real, incluyendo una gráfica de la presión de entrada y salida.

server.js:

```
socket.on ('envio_datosVeMec', function (msg) {
        console.log('data: '+msg);
    var json = JSON.parse(msg);
        io.sockets.emit('datosVeMec'+json.Id, json); //envía con la id
});
```

Luego cada 60 segundos los simuladores envían la data, además de al websocket, a un servicio Restful que se encarga de hacer un insert en la base de datos con dichos datos.

```
...
socket.emit("envio_datosVeMec", JSON.stringify(data.getData()));
console.log("> Esto es el ventilador " + _this.id + "ts" + data.time_Stamp);
segs += 1;

if (segs > 60) {
        segs = 0;
        console.log("Enviando Respaldo a Bd");

        console.log(JSON.stringify(data.getJSON()));
        let json_data = JSON.stringify(data.getJSON());
        request(
        {
                uri:
"http://localhost:8080/RESTapi/webresources/entities.vemecsdata",
        method: "POST",
```

```
        body: JSON.stringify(data.getJSON()),
        headers: { "Content-Type": "application/json" },
        },
        function (error, response, body) {
          console.log(body);
        }
        );
 }
...
```

El programa administrador para tener el historial entero de datos también se fija en la bsd y para arrancar a graficar utiliza todos los datos previos que se tienen del respirador del cual se desea ver su información, luego se va actualizando a tiempo real tomando en cuenta los datos recibidos por el websocket.

```
<script language = "javascript" type = "text/javascript">

...

socket.on('datosVeMec'+${id}, (res) => {

        console.log("recibiendo datos de vemec...");

        var json = res;

        document.getElementById('p_mx').innerHTML = json.Presion_Maxima;

        document.getElementById('p_mn').innerHTML = json.Presion_Minima;

        document.getElementById('gas').innerHTML = json.Gas;

        document.getElementById('frec').innerHTML = json.Frecuencia;

        document.getElementById('mez').innerHTML = json.Mezcla;

        document.getElementById('hum').innerHTML = json.Humedad;

        document.getElementById('t_in').innerHTML = json.Temperatura_Entrada;

        document.getElementById('t_out').innerHTML = json.Temperatura_Salida;

        document.getElementById('p_in').innerHTML = json.Presion_Entrada;

        document.getElementById('p_out').innerHTML = json.Presion_Salida;


        console.log("agregando datos a la gráfica...");

        segundos++;

        ...

        arrayDatos.push(['${segundos}' , json.Presion_Entrada, json.Presion_Salida]);

});

...

</script>
```

El programa administrador está creado en Java EE usando spring cómo framework.

Los simuladores y el websocket Service están ambos hechos en NodeJs, ya que nos pareció la opción más dinámica y sencilla logrando una excelente integración con Socket.Io y haciéndonos todas las dependencias necesarias de una forma muy prolija y dinámica con sus Packages de dependencias auto generadas.

El servicio de RESTful fue realizado con Java EE y las librerías JaxRs2.0 que están dentro de las mismas libs de glassfish.