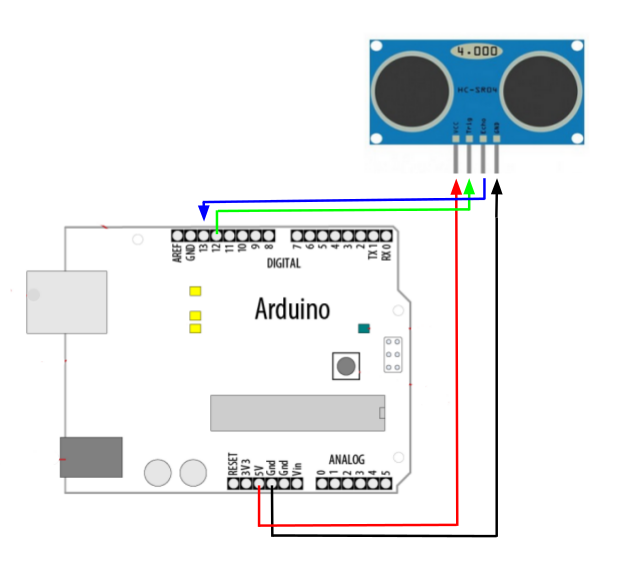
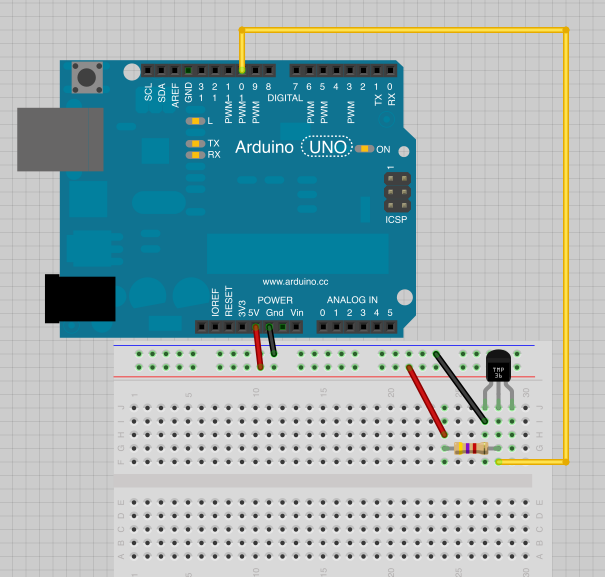
# Proyecto Final

Medición de Temperatura y Distancia en Tiempo Real usando las Tecnologías de: Arduino, Dweet, Node.js y Spring Framework MVC aplicadas a la tecnología WebSocket sobre OpenShift como PAAS.

## DIAGRAMA ESQUEMÁTICO

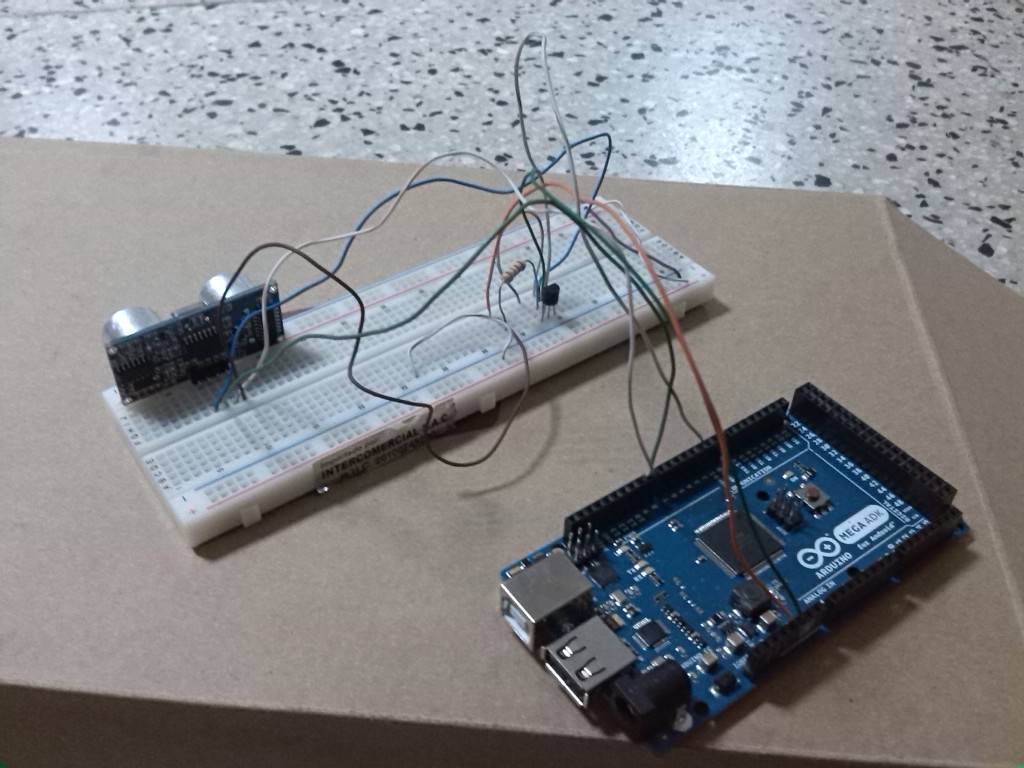
Sensor HC-SR04



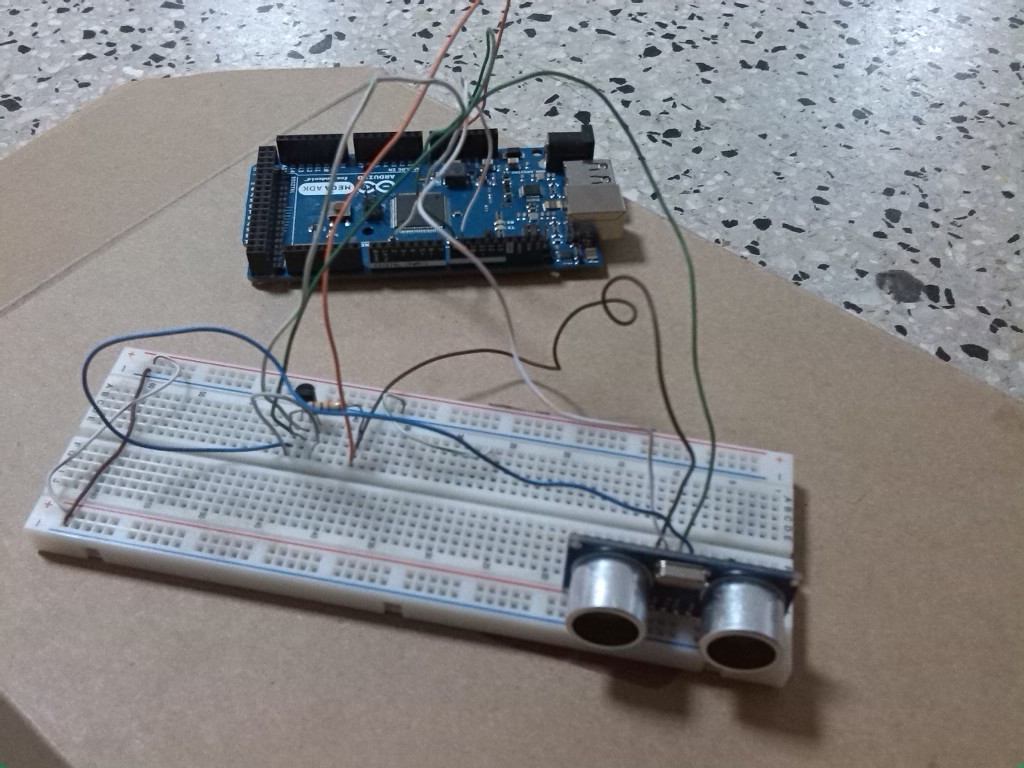
Sensor DS18B20

## DESARROLLO y DIAGRAMA DE LA IMPLEMENTACION

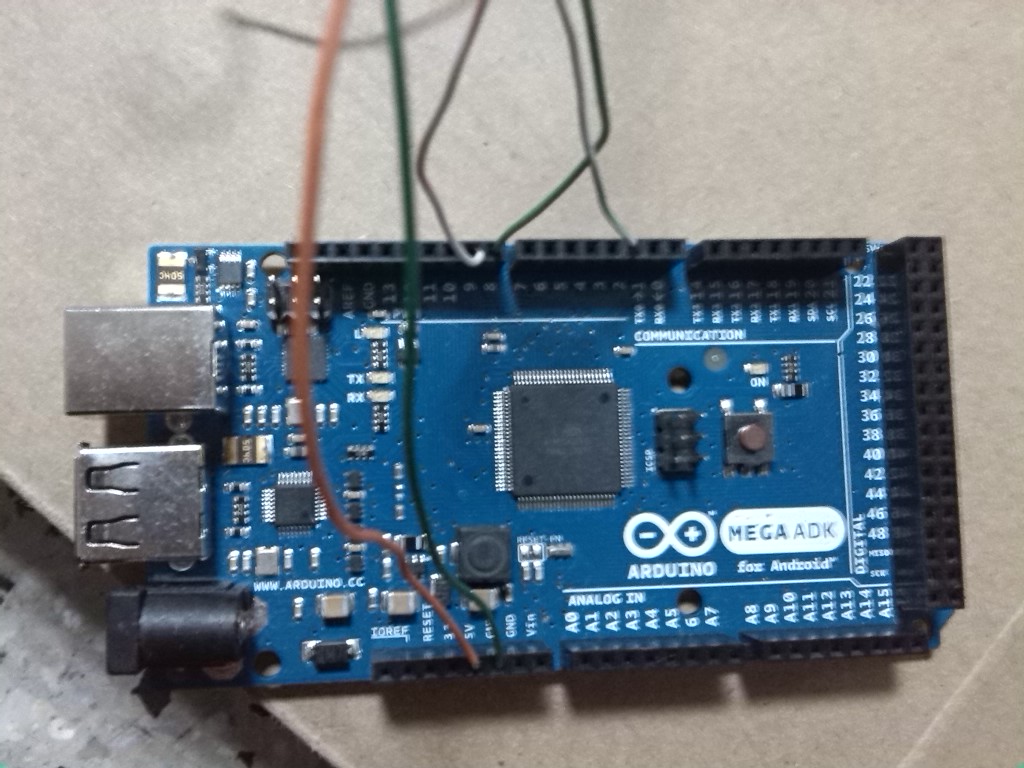
Se ubica el sensor de temperatura DS18B20 en un entorno abierto con el fin de medir la temperatura del ambiente.



Se ubica un sensor de distancia HC-SR04, el cuál emite una onda ultrasonido imperceptible por el humano y que chocará con el primer cuerpo que encuentre volviendo al sensor y de esta manera calcula la distancia.



Se ha utilizado un Arduino MEGA ADK, en la siguiente imagen se muestran los pines a los que están conectados los sensores para poder transmitir los datos de éstos a la computadora.



## CODIGO DESARROLLADO

**El código Arduino**

**Aquì se incluyen las librerías DallasTemperature y OneWire para poder obtener los datos del sensor de Temperatura y Distancia que serán enviados por el puerto serial. Esta información es enviada cada segundo.**

#include <DallasTemperature.h>

#include <OneWire.h>

#define Pin 2 //Se declara el pin donde se conectarᠬa DATA

OneWire ourWire**(**Pin**);** //Se establece el pin declarado como bus para la comunicaci󮠏neWire

DallasTemperature sensors**(&**ourWire**);** //Se instancia la librerDallasTemperature

long distancia**;**

long tiempo**;**

void setup**()** **{**

delay**(**1000**);**

Serial**.**begin**(**9600**);**

pinMode**(**9**,** OUTPUT**);** /\*activaci󮠤el pin 9 como salida: para el pulso ultras󮩣o\*/

pinMode**(**8**,** INPUT**);** /\*activaci󮠤el pin 8 como entrada: tiempo del rebote del ultrasonido\*/

sensors**.**begin**();** //Se inician los sensores

**}**

void loop**()** **{**

sensors**.**requestTemperatures**();** //Prepara el sensor para la lectura

digitalWrite**(**9**,**LOW**);** /\* Por cuesti󮠤e estabilizaci󮠤el sensor\*/

delayMicroseconds**(**5**);**

digitalWrite**(**9**,** HIGH**);** /\* envdel pulso ultras󮩣o\*/

delayMicroseconds**(**10**);**

tiempo**=**pulseIn**(**8**,** HIGH**);** /\* Funci󮠰ara medir la longitud del pulso entrante. Mide el tiempo que transcurrido entre el env

del pulso ultras󮩣o y cuando el sensor recibe el rebote, es decir: desde que el pin 12 empieza a recibir el rebote, HIGH, hasta que

deja de hacerlo, LOW, la longitud del pulso entrante\*/

distancia**=** int**(**0.017**\***tiempo**);** /\*f󲭵la para calcular la distancia obteniendo un valor entero\*/

/\*Monitorizaci󮠥n centtros por el monitor serial\*/

Serial**.**print**(**sensors**.**getTempCByIndex**(**0**));** //Se lee e imprime la temperatura en grados Celsius

Serial**.**print**(**"/"**);**

Serial**.**println**(**distancia**);**

delay**(**1000**);** //Se provoca un lapso de 1 segundo antes de la pr󸩭a lectura

**}**

**El código Node.js**

**Cumple el rol de intermediario entre el Arduino y el Dweet, se hará uso de las librerías “SeriaPort” y “Node-Dweet.io. Se recogerá los datos del arduino enviados por el puerto serial y a través de las funciones del SeriaPort que se activan a través de los eventos se enviará la información a Dweet.io usando el formato JSON.**

***var*** dweetClient **=** require**(**"node-dweetio"**);**

***var*** dweetio **=** ***new*** dweetClient**();**

***var*** serialport **=** require**(**'serialport'**),**

SerialPort **=** serialport**.**SerialPort**,**

portName **=** "COM12"

***var*** myPort **=** ***new*** SerialPort**(**portName**,** **{**

baudRate**:** 9600**,**

parser**:** serialport**.**parsers**.**readline**(**"\r\n"**)**

**});**

myPort**.**on**(**'open'**,** showPortOpen**);**

myPort**.**on**(**'data'**,** saveLatestData**);**

myPort**.**on**(**'close'**,** showPortClose**);**

myPort**.**on**(**'error'**,** showError**);**

***function*** showPortOpen**()** **{**

console**.**log**(**'port open. Data rate: ' **+** myPort**.**options**.**baudRate**);**

**}**

***function*** saveLatestData**(**data**)** **{**

console**.**log**(**data**);**

***var*** temp**=**""**;**

***var*** dist**=**""**;**

***var*** dim**;**

***for*** **(*var*** i **=** 0**;** i **<** data**.**length**;** i**++)** **{**

***if*(**data**.**charAt**(**i**)** **==** '/'**){**

dim**=**i**;**

**}**

**}**

***for*(*var*** j**=**0**;** j**<** dim**;** j**++){**

temp**=** temp**+** data**.**charAt**(**j**);**

**}**

***for*(*var*** k**=**dim**+**1**;** k**<** data**.**length**;** k**++){**

dist**=** dist **+** data**.**charAt**(**k**);**

**}**

dweetio**.**dweet\_for**(**"SSDDFISI"**,** **{**temperatura**:** temp**,** distancia**:** dist**},** ***function*(**err**,** dweet**){**

**});**

**}**

***function*** showPortClose**()** **{**

console**.**log**(**'port closed.'**);**

**}**

***function*** showError**(**error**)** **{**

console**.**log**(**'Serial port error: ' **+** error**);**

**}**

**La clase Principal HomeController.java**

**Mediante esta clase se implementará el WebSocket usando la librería Javax.websocket, que permitirá la muestra de la información actualizada en tiempo real. También en esta clase se hará uso de un objeto de la clase Dweet y haciendo uso de sus funciones se accederá a los datos en formato JSON de los sensores.**

@ServerEndpoint**(**value**=**"/ws"**)**

@Singleton

@Controller

public class HomeController **{**

@Autowired

private InsEducService instEducService**;**

private static final Set**<**Session**>** userSessions **=** Collections**.**synchronizedSet**(new** HashSet**<**Session**>());**

private TaskScheduler scheduler **=** **new** ConcurrentTaskScheduler**();**

private static List**<**Mensaje**>** mensajesC **=** **new** ArrayList**<**Mensaje**>();**

private static List**<**Mensaje**>** mensajesS **=** **new** ArrayList**<**Mensaje**>();**

private static Cronometro miCronometro **=** **new** Cronometro**(**999999999**);**

static final int SERVIDOR **=** 999999999**;**

static final int CLIENTE **=** 30**;**

@OnOpen

public void onOpen**(**Session userSession**)**

**{**

System**.**out**.**println**(**"Usuario nuevo: " **+** userSession**.**getId**());**

userSessions**.**add**(**userSession**);**

**}**

@OnClose

public void onClose**(**Session userSession**)** **{**

System**.**out**.**println**(**"Connection closed. Id: " **+** userSession**.**getId**());**

userSessions**.**remove**(**userSession**);**

**}**

@OnMessage

public void onMessage**(**String message**,** Session userSession**)** **throws** IOException **{**

//guardo los mensajes en una lista estatica

String miTurno**=**turno**(**message**);**

String mensaje**=**mensaje**(**message**);**

**if(!**miTurno**.**equals**(**"tres"**)){**

Mensaje mensj **=** **new** Mensaje**();**

mensj**.**setMensaje**(**mensaje**);**

**if(**miTurno**.**equals**(**"uno"**)){**

mensj**.**setTurno**(**1**);**

**}else{**

mensj**.**setTurno**(**0**);**

**}**

mensajesC**.**add**(**mensj**);**

mensajesS**.**add**(**mensj**);**

**}**

**if(**miTurno**.**equals**(**"uno"**)){**

updatePriceAndBroadcast**(**CLIENTE**,**message**,**userSession**);**

miCronometro**.**setTiempo**(**CLIENTE**);**

**}else{**

**if(**miTurno**.**equals**(**"cero"**)){**

updatePriceAndBroadcast**(**SERVIDOR**,**message**,**userSession**);**

miCronometro**.**setTiempo**(**SERVIDOR**);**

**}else{**

updateInfoSensores**(**message**,**userSession**);**

**}**

**}**

**}**

public static String mensaje**(**String Mensaje**)**

**{**

String mensaje**=**Mensaje**.**substring**(**12**,** Mensaje**.**length**());**

int coma**=**mensaje**.**lastIndexOf**(**","**);**

mensaje**=**mensaje**.**substring**(**0**,**coma**-**1**);**

**return** mensaje**;**

**}**

public static String turno**(**String Mensaje**)**

**{**

int coma**=**Mensaje**.**lastIndexOf**(**","**);**

String turno**=**Mensaje**.**substring**(**coma**+**10**,** Mensaje**.**length**()-**2**);**

**return** turno**;**

**}**

/\*\*

\* Iterates stock list, update the price by randomly choosing a positive

\* or negative percentage, then broadcast it to all subscribing clients

\*/

private void updatePriceAndBroadcast**(**int solicitante**,** String nuevoMensaje**,** Session userSession**)** **{**

int msjC **=** 0**;**

int msjS **=** 0**;**

**for(**Mensaje msj **:** mensajesC**){**

**if(**msj**.**getTurno**()==**1**){**

msjC**++;**

**}**

**if(**msj**.**getTurno**()==**0**){**

msjS**++;**

**}**

**}**

**if((**msjC**==**1 **&&** msjS**==**0**)** **||** **!**miCronometro**.**isContinuar**()){**

miCronometro **=** **new** Cronometro**(**solicitante**);**

miCronometro**.**start**();**

**}**

**if(**msjC**>**0**){**

System**.**out**.**println**(**"cantidad de usuario: " **+** userSessions**.**size**());**

**for** **(**Session session **:** userSessions**)**

**{**

session**.**getAsyncRemote**().**sendText**(**nuevoMensaje**);**

**}**

**}else{**

mensajesC**.**clear**();**

mensajesS**.**clear**();**

**}**

**}**

public void updateInfoSensores**(**String nuevoMensaje**,** Session userSession**){**

**for** **(**Session session **:** userSessions**)**

**{**

session**.**getAsyncRemote**().**sendText**(**nuevoMensaje**);**

**}**

**}**

/\*\*

\* Invoked after bean creation is complete, this method will schedule

\* updatePriceAndBroacast every 1 second

\*/

@PostConstruct

private void broadcastTimePeriodically**()** **{**

scheduler**.**scheduleAtFixedRate**(new** Runnable**()** **{**

@Override public void run**()** **{**

// System.out.println("el tiempo es: "+ miCronometro.getTiempo());

**if(**miCronometro**.**getTiempo**()==**1**){**

mensajeNoDisponibilidad**();**

**}**

mostrarInformacion**();**

**}**

**},** 1000**);**

**}**

public void mensajeNoDisponibilidad**(){**

String mensajeN **=** "SERVIDOR: Disculpenos por la demora; ahora no tenemos solucion a su consulta"**+**

" pero cuando la tengamos responderemos inmediatamente. Muchas Gracias\\n\\n"**;**

String nuevoMensaje **=** "{\"mensaje\":\""**+**mensajeN**+**"\",\""**+** "turno\":\"cero\"}"**;**

Session userSession **=** **null;**

**try** **{**

onMessage**(**nuevoMensaje**,** userSession**);**

**}** **catch** **(**IOException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

public void mostrarInformacion**(){**

Dweet dweetio **=** **new** Dweet**();**

String infoDist **=** dweetio**.**obtenerValorDist**();**

String infoTemp **=** dweetio**.**obtenerValorTemp**();**

String nuevoMensajetemp **=** "{\"mensaje\":\""**+**infoTemp**+**"\",\""**+** "turno\":\"tres\"}"**;**

String nuevoMensajedist **=** "{\"mensaje\":\""**+**infoDist**+**"\",\""**+** "turno\":\"tres\"}"**;**

Session userSession **=** **null;**

**try** **{**

onMessage**(**nuevoMensajetemp**,** userSession**);**

onMessage**(**nuevoMensajedist**,** userSession**);**

**}** **catch** **(**IOException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

**}**

/\*\*

\* Serve the main page

\*/

@RequestMapping**(**value**=**"/instEduc.do"**,** method**=**RequestMethod**.**GET**)**

public String setupForm**(**Model modelo**){**

modelo**.**addAttribute**(**"listacoles"**,** instEducService**.**obtenerlistaInst**());**

modelo**.**addAttribute**(**"colegio"**,** **new** InstEducativa**());**

**return** "instEducativa"**;**

**}**

@RequestMapping**(**value**=**"/instEduc.do"**,** method**=**RequestMethod**.**POST**)**

public String doACtions**(**@ModelAttribute**(**"colegio"**)** InstEducativa colegio**,**

BindingResult resultado**,** @RequestParam String accion**,** Model modelo**){** /\*\*/

InstEducativa coleResultado **=** **new** InstEducativa**();**

List**<**InstEducativa**>** institucionBuscada **=** **new** ArrayList**<**InstEducativa**>();**

**switch(**Integer**.**parseInt**(**accion**)){**

**case** 1**:**

coleResultado**=**colegio**;**

institucionBuscada **=**instEducService**.**obtenerlistaInstN**(**colegio**.**getInstitucion**());**

**if(**institucionBuscada**.**size**()<**1**){**

institucionBuscada**.**add**(new** InstEducativa**());**

modelo**.**addAttribute**(**"mensaje"**,** "ColegioNoregistrado"**);**

**}**

**break;**

**case** 2**:**

coleResultado**=**colegio**;**

**if(**esNumero**(**colegio**.**getInstitucion**()))**

institucionBuscada**=** instEducService**.**obtenerlistaInstU**(**Integer**.**parseInt**(**colegio**.**getInstitucion**()));**

**else{**

institucionBuscada**.**add**(new** InstEducativa**());**

modelo**.**addAttribute**(**"mensaje"**,** "Noesentero"**);}**

**break;**

**case** 3**:**

coleResultado**=**colegio**;**

institucionBuscada**=** instEducService**.**obtenerlistaInstD**(**colegio**.**getInstitucion**());**

**if(**institucionBuscada**.**size**()<**1**){**

institucionBuscada**.**add**(new** InstEducativa**());**

modelo**.**addAttribute**(**"mensaje"**,** "ColegioNoregistrado"**);**

**}**

**break;**

**case** 4**:**

coleResultado**=**colegio**;**

**if(**esNumero**(**colegio**.**getInstitucion**()))**

institucionBuscada**.**add**(**instEducService**.**obtenerInst**(**Integer**.**parseInt**(**colegio**.**getInstitucion**())));**

**else{**

institucionBuscada**.**add**(new** InstEducativa**());**

modelo**.**addAttribute**(**"mensaje"**,** "Noesentero"**);}**

**break;**

**}**

modelo**.**addAttribute**(**"colegios"**,** coleResultado**);**

modelo**.**addAttribute**(**"listacoles"**,** institucionBuscada**);**

**return** "instEducativa"**;**

**}**

public boolean esNumero**(**String cadena**){**

boolean esN**;**

**try{**

int numero **=** Integer**.**parseInt**(**cadena**);**

esN**=** **true;**

**}catch(**NumberFormatException e**){**

esN**=false;**

**}**

**return** esN**;**

**}**

@RequestMapping**(**value **=** "/"**)**

public String principal**()** **{**

**return** "inicio"**;**

**}**

@RequestMapping**(**value **=** "/inicio.html"**)**

public String retorno**()** **{**

**return** "inicio"**;**

**}**

@RequestMapping**(**value **=** "/servicio.html"**)**

public String servicio**()** **{**

**return** "servicio"**;**

**}**

@RequestMapping**(**value **=** "/cliente.html"**)**

public String cliente**()** **{**

**return** "cliente"**;**

**}**

@RequestMapping**(**value **=** "/servidor.html"**)**

public String servidor**()** **{**

**return** "servidor"**;**

**}**

@RequestMapping**(**value **=** "/proyecto.html"**)**

public String proyecto**()** **{**

**return** "Proyectos"**;**

**}**

**}**

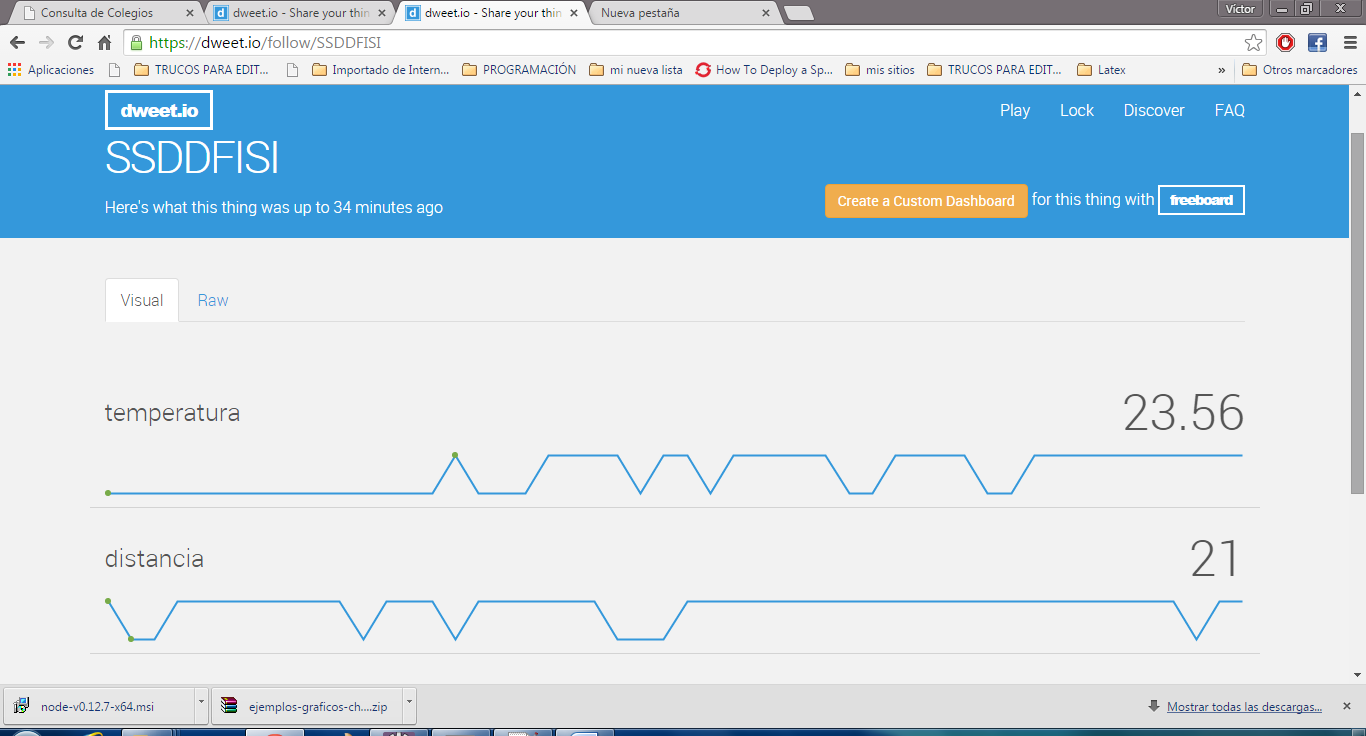
# Dweet:

Dweet es un repositorio de información en formato JSON enviada por sensores, que se está haciendo uso para el almacenamiento de los datos enviados por los sensores de temperatura y distancia.

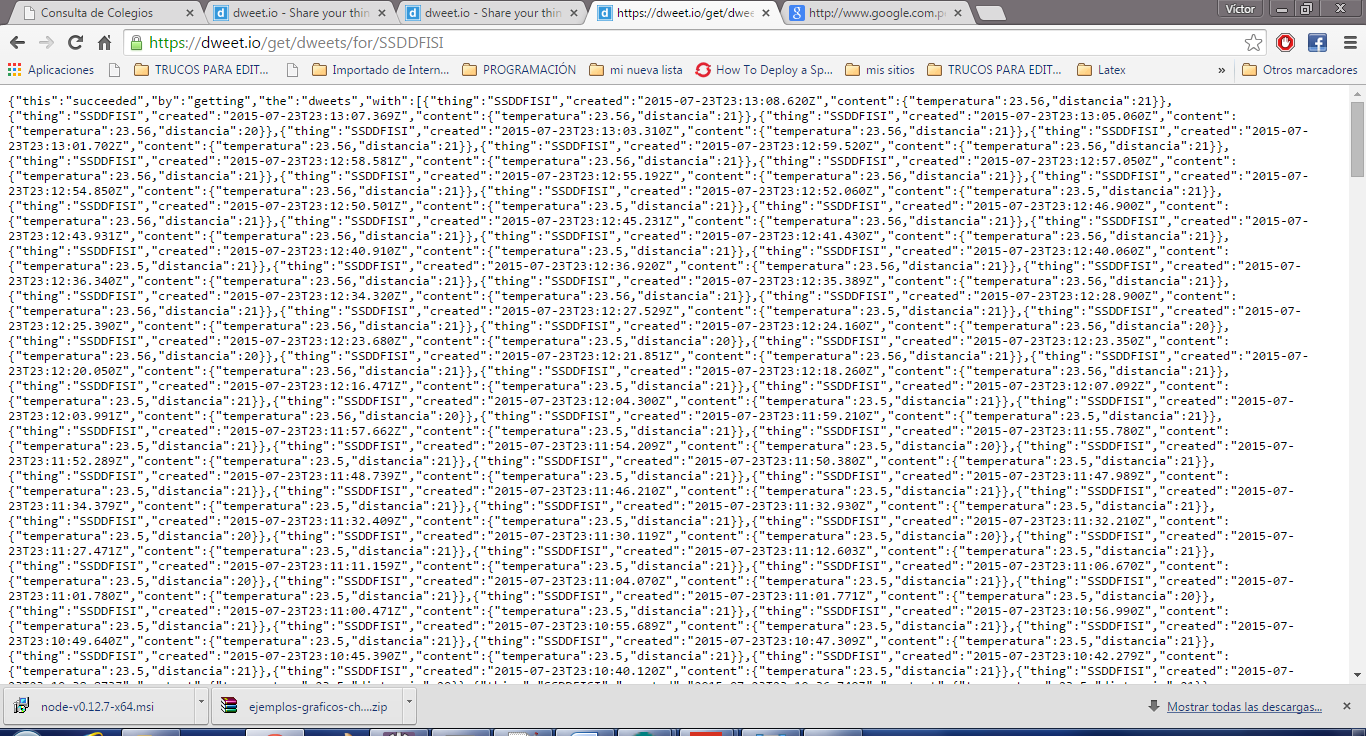
Gracias a que es una tecnología gratuita se puede utilizar para obtener los datos guardados en este repositorio en línea y poder acceder a ellos mediante Openshift.

A continuación, mostramos la lectura de nuestros datos en la interfaz que proporciona el dweet.io:

Cuando empece a generarse cambios estos se mostraran en tiempo real en los dispositivos que pueda usar el usuario.



Los cuales son almacenados por la misma en formato JSON:



Y el resultado se muestra en la aplicación almacenada en la nube de OpenShift.

