

## Práctica 4: Node-RED

Autores:

Alejandro Serrano Fernández

Pedro Antonio Navas Luque

## Introducción:

Para el correcto funcionamiento de los flujos hay que tener instalado Node-RED en nuestro sistema junto al paquete de email que será necesario para el flujo 3. He aquí los siguientes comandos que debemos de introducir previamente en nuestra terminal de Linux:

```
$ sudo apt-get update
$ sudo apt-get install nodejs
$ sudo apt-get install npm
$ sudo npm install -g --unsafe-perm node-red node-red-admin
$ cd ~/.node-red
$ npm i node-red-node-email
```

## Flujo 1: Calculadora de Vectores mediante API REST

Inicialmente usamos un nodo http in, para realizar una petición GET en los diferentes endpoints. Éste recibirá el json que nos envía el cliente y posteriormente pasará a una función que determinará la longitud de los vectores y realizará las operaciones que sean necesarias.

Mejora: En nuestro caso está diseñado de tal manera que ésta pueda realizar las operaciones con cualquier tipo de longitud de los vectores, tal como se muestra en el algoritmo de la [Figura 1](#).

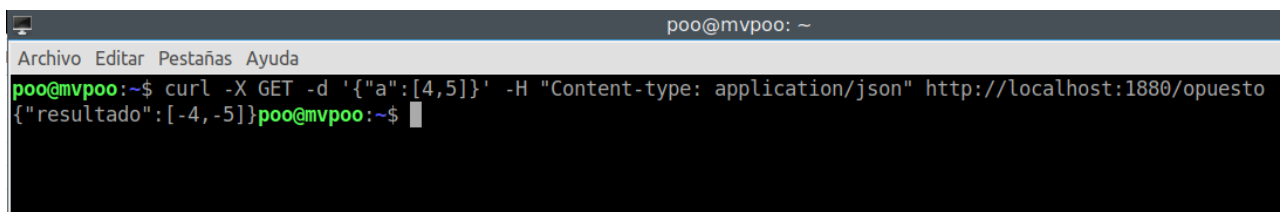
Tras determinar el resultado final, éste se enviará mediante un nodo http response, para que el cliente pueda recibir el resultado de la operación introducida.

Mejora: Además, el resultado final, se almacenará en un fichero que mantiene todos los logs de las peticiones recibidas junto con los resultados. (Nota: el fichero se almacenará en /home/<user>, siendo user el nombre del usuario del sistema.)

Mejora: Finalmente, hemos añadido tres endpoint más con el que se puede **multiplicar** vectores de cualquier longitud (Ej:  $a[1,2,1]$ ,  $b[2,3] = [2,6,1]$ ), realizar el **producto escalar** de dos vectores (Ej:  $a[1,2,1]$ ,  $b[2,3] = 9$ ) y realizar el **opuesto** de un vector (Ej:  $a[2,3] = [-2,-3]$ ). Para realizar éste último tan solo habría que especificar un vector en el comando, tal como se muestra en la [Figura 2](#).

```
1 var array = []
2
3 var longitud_a = msg.req.body.a.length;
4 var longitud_b = msg.req.body.b.length;
5
6 if(longitud_a >= longitud_b){
7
8     for(var i = 0; i < longitud_a; i++){
9         if(i > longitud_b - 1) array[i] = msg.req.body.a[i]
10        else array[i] = msg.req.body.a[i] + msg.req.body.b[i];
11        console.log(array[i])
12    }
13 }
14 }
15 else{
16
17     for(var j = 0; j < longitud_b; j++){
18         if(j > longitud_a - 1) array[j] = msg.req.body.b[j];
19         else array[j] = msg.req.body.a[j] + msg.req.body.b[j];
20    }
21 }
22
23 msg.payload = {"resultado":array}
24 return msg;
```

Figura 1: Algoritmo para determinar la suma de dos vectores de cualquier longitud



A terminal window titled 'poo@mvppoo: ~' showing a command and its output. The command is: `curl -X GET -d '{"a":[4,5]}' -H "Content-type: application/json" http://localhost:1880/opuesto`. The output is: `{"resultado":[-4,-5]}`.

Figura 2: Comando para calcular el opuesto de un vector

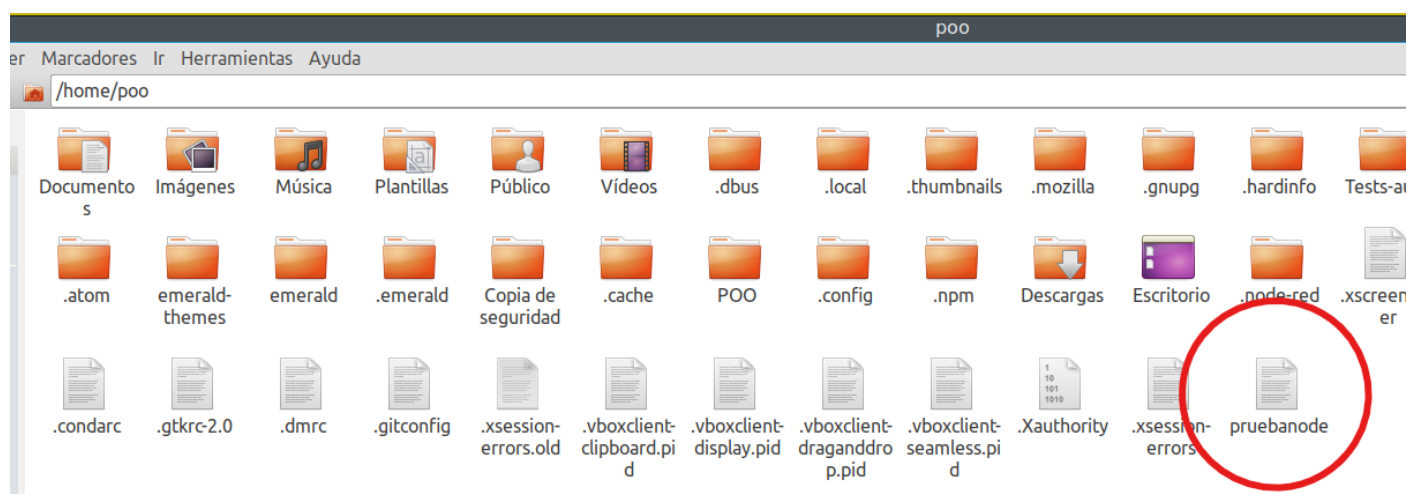


Figura 3: Fichero en el que se almacenan los resultados

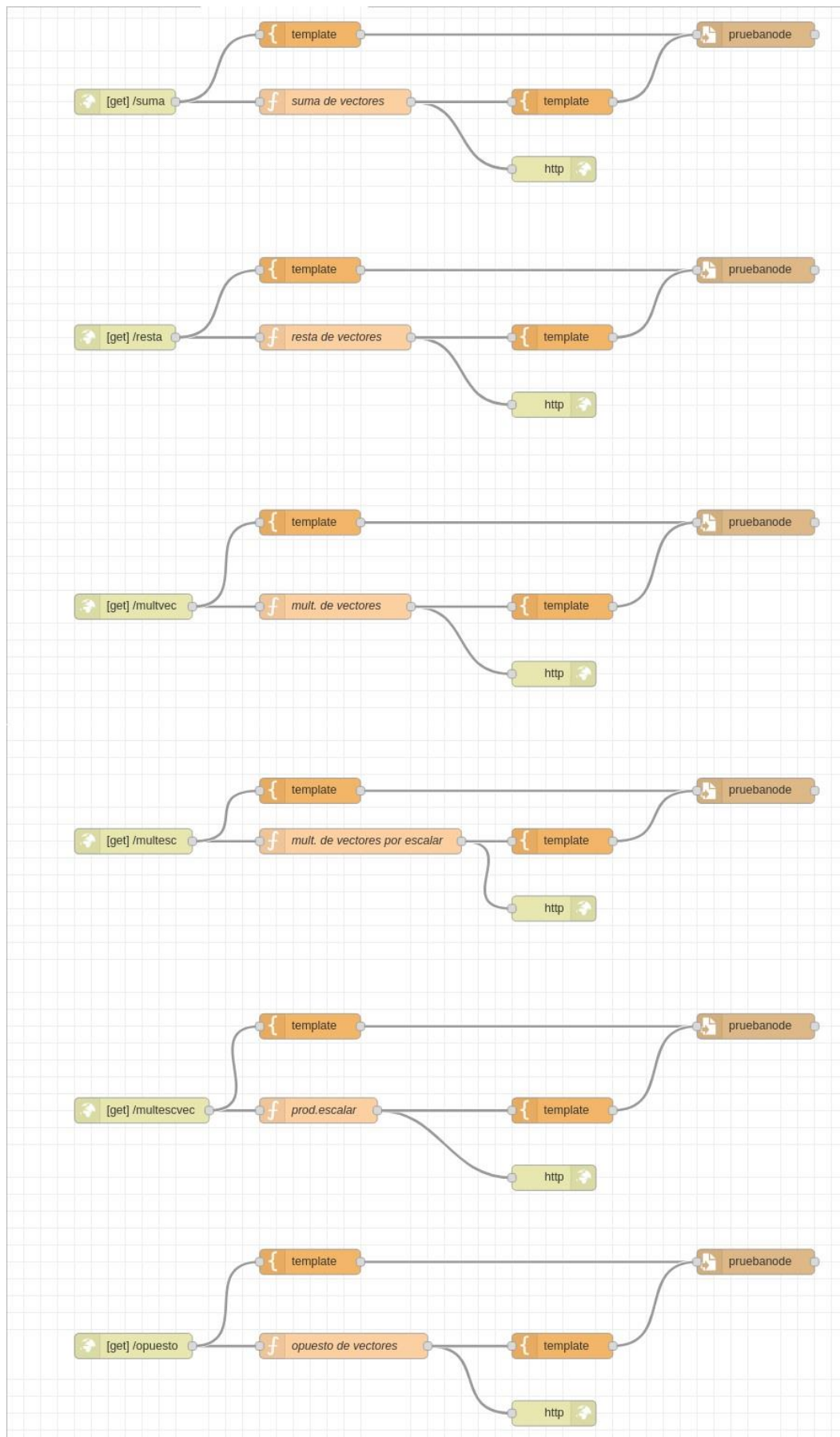


Figura 4: Flujo del problema 1

## Flujo 2: MQTT

Siguiendo el enunciado del problema 2, utilizaremos un nodo **mqtt in** para subscribirnos al enlace que se indica en el enunciado de la práctica (dirección: <https://test.mosquitto.org/> , tópico: `/merakimv/Q2GV-Y4R8-5Z3L/light`) del cual recibiremos la luminosidad que registra un cámara de seguridad.

Tras el nodo **mqtt in**, hemos puesto un nodo **delay** para poder observar los datos con mayor claridad.

Seguidamente transformamos el string recibido, a un objeto Javascript Object con un nodo de la clase parser, en concreto el nodo **JSON**.

Una vez transformado, determinaremos el máximo (que se almacenará en un “variable” contexto) a través de una función, y lo mostrará en el debugger (Se mostrará el máximo y el valor recibido).

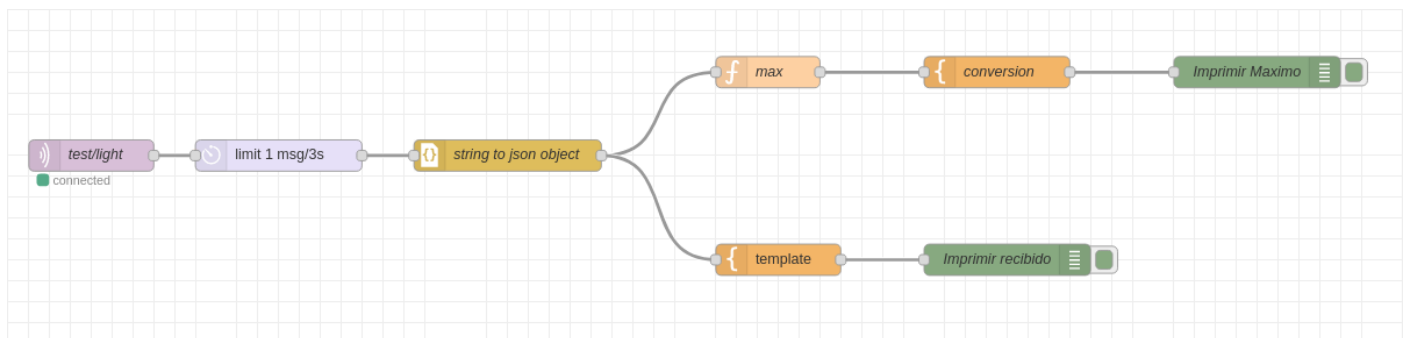


Figura 5: Flujo del problema 2

```

26/5/2020 20:13:27 node: Imprimir recibido
/merakimv/Q2GV-Y4R8-5Z3L/light : msg.payload :
string[25]
"El valor recibido es: 8.4"

26/5/2020 20:13:27 node: Imprimir Maximo
/merakimv/Q2GV-Y4R8-5Z3L/light : msg.payload :
string[30]
"Actualmente el máximo es: 8.4!"

26/5/2020 20:13:30 node: Imprimir recibido
/merakimv/Q2GV-Y4R8-5Z3L/light : msg.payload :
string[25]
"El valor recibido es: 8.2"

26/5/2020 20:13:30 node: Imprimir Maximo
/merakimv/Q2GV-Y4R8-5Z3L/light : msg.payload :
string[30]
"Actualmente el máximo es: 8.4!"

26/5/2020 20:13:33 node: Imprimir recibido
/merakimv/Q2GV-Y4R8-5Z3L/light : msg.payload :
string[25]
"El valor recibido es: 8.2"

26/5/2020 20:13:33 node: Imprimir Maximo
/merakimv/Q2GV-Y4R8-5Z3L/light : msg.payload :
string[30]
"Actualmente el máximo es: 8.4!"
    
```

Figura 6: Muestra de los valores recibidos y el máximo de ellos

```

1 var max = context.get('maximo') || 0;
2 console.log(msg.payload.lux)
3
4 if(msg.payload.lux > max ){
5     max = msg.payload.lux;
6     context.set('maximo',msg.payload.lux);
7 }
8
9 msg.max = max;
10 return msg;

```

Figura 7: Algoritmo para determinar el máximo de los valores recibidos

## Flujo 2: Mejora

Para publicar y recibir eventos, hemos utilizado un nodo **mqtt out** con la dirección <https://test.mosquitto.org/> donde serán publicados los eventos que seleccionaremos en el flujo (Temperatura, Coche, String) con el tópico /serfer/navluq. El nodo **mqtt in** recibirá el evento que hayamos seleccionado anteriormente y lo mostrará por el debugger.

Nota: Hemos utilizado un nodo switch para imprimir de diferente manera los distintos eventos que recibamos.

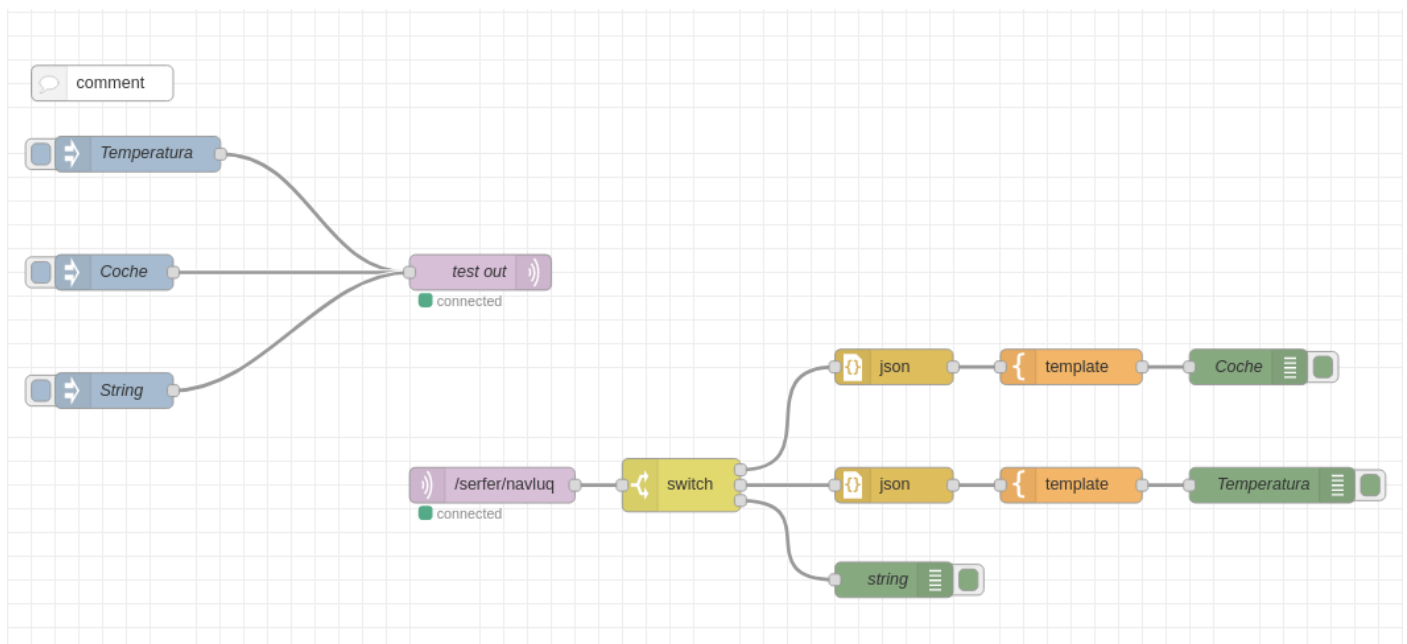


Figura 8: Flujo de la mejora del problema 2

## Práctica 4: Node-RED

En las siguientes imágenes, se muestra el contenido de cada evento:



Figura 9: Evento temperatura

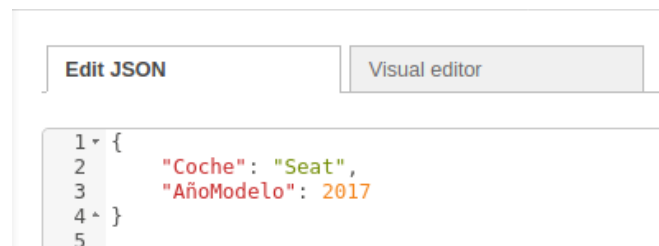


Figura 10: Evento Coche

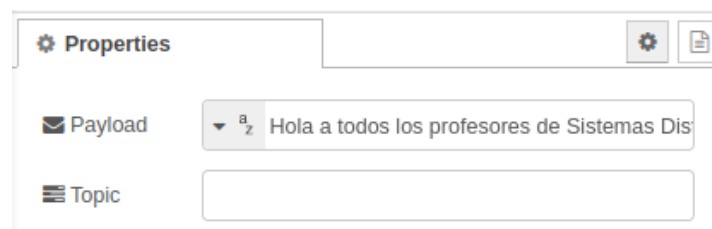
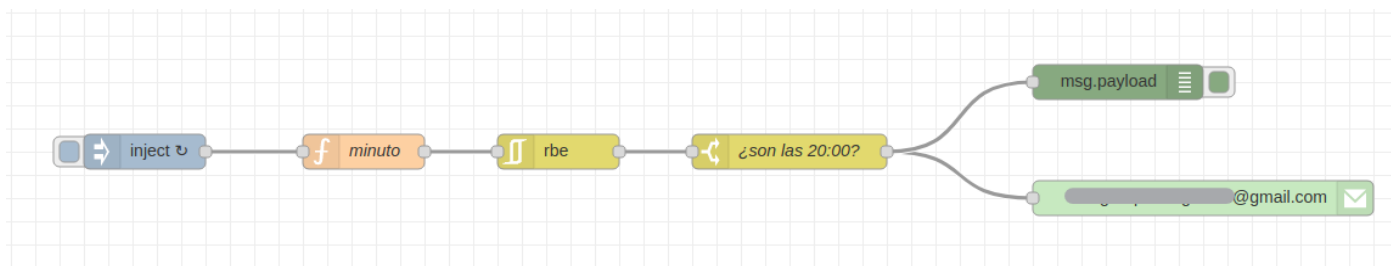


Figura 11: Evento String



Figura 12: Mensajes recibidos del servidor mqtt

## Flujo 3: Email que se envía todos los días a las 8 de la tarde



Hemos diseñado un flujo, utilizando el nodo **email** que previamente habremos de instalar, con el que recibiremos un mensaje personalizado al email que hemos configurado, a una hora establecida, en nuestro caso a las 20:00.

El flujo comienza inyectando cada 1 minuto el mensaje que queramos enviar. La función que le procede determinará la hora del sistema y la devolverá. En la salida, le sigue un nodo **rbe** (report by exception node), este nodo se encargará de pasar el mensaje de la función si el mensaje que devuelve la función es distinto, por ejemplo, si la función devuelve cada 1 minuto la hora, pongamos las 2, la función dejará pasar el mensaje, cuando sea distinto a 2, es decir, que sean las 3. Cuando se recibe una hora distinta, el switch valorará si la hora recibida coincide con las 8 de la tarde, en tal caso, enviará un email al correo configurado.

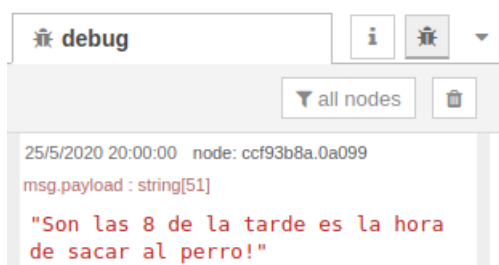


Figura 13: Muestra del mensaje por el debugger

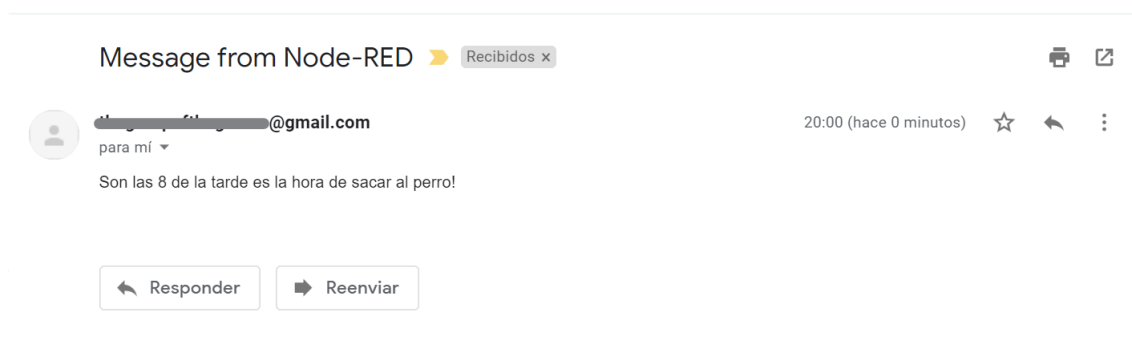


Figura 14: Muestra del mensaje recibido en el correo