Práctica 4: Node-RED

Sistemas Distribuidos

Ejercicios

1. Crear un flujo en Node-RED con 3 nodos de tipo *inject*, que llevarán a cabo la siguiente funcionalidad:

   - El primer nodo, generará un mensaje automático con el número 3 cada 2 segundos.
   - El segundo nodo, generará un mensaje automático con el número 4 cada 3 segundos.
   - El tercer y último nodo, generará un mensaje con la cadena "restart" que se insertará (lanzará) manualmente.

   Posteriormente, se incluirá un nodo *function* que irá restando los números pares y sumando los impares. El resultado de cada operación debe almacenarse en una variable acumulador que debe ser persistente.

   - Tanto la operación realizada como el resultado de la misma, deben almacenarse en un fichero (a modo de log). Por ejemplo, si se recibe el número 4 que es par, siendo el valor de la variable acumulador 0, en el fichero se debe almacenar algo similar a: *Resta 4 Resultado -4*.
   - Si se recibe la cadena "restart" en el fichero debe almacenarse esta cadena y el acumulador debe reiniciarse a 0. Ejemplo de este caso: *restart Resultado 0*.

   El flujo generado debe exportarse completo.

   Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

2. Crear un flujo en Node-RED con 3 nodos de tipo *inject*, que llevarán a cabo la siguiente funcionalidad:

   - El primer nodo, denominado Euro, que generará el mensaje JSON: "moneda":"euro", "cambio": 0.000038. Este mensaje se insertará (lanzará) manualmente.

- El segundo nodo, denominado Bitcoin, que generará el mensaje JSON: “moneda”:”bitcoin”, “cambio”: 26621.05. Este mensaje se insertará (lanzará) manualmente.
- El tercer y último nodo, generará un mensaje automático con la cadena “restart” cada 5 segundos.

Posteriormente, se incluirá un nodo *function* cuya funcionalidad es la de convertir siempre 10 unidades a euros o Bitcoins según la elección del usuario, y también ir contabilizando el número de operaciones realizadas, de cada tipo (euro y Bitcoin). La función debe coger el “cambio” del nodo *inject*.

Se incluirá otro nodo *function* cuya funcionalidad es la de reiniciar los contadores de las operaciones cuando reciba la cadena “restart”.

Para mostrar la salida, se requiere utilizar al menos un nodo *template*.

Ejemplos de salida:

- Si el usuario selecciona Bitcoin la salida será: 10 unidades equivalen a “resultado” Bitcoins (“contador” op.).
- Si el usuario selecciona Euros la salida será: 10 unidades equivalen a “resultado” euros (“contador” op.).
- Cuando se inserte “restart”, la salida será: Reiniciar “contador”Op. Bitcoins y “contador” Op. euros.

El flujo generado debe exportarse completo.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

3. Crear un flujo en Node-RED que deberá cubrir la funcionalidad que se describe a continuación. Deberá proporcionar un endpoint `/visita` mediante el nodo *http-in* al que se enviará un JSON con el siguiente formato:

```
{ "usuario": "Pedro", "mensaje": "contenido del mensaje" }
```

Cuando el endpoint reciba un mensaje, éste deberá responder con un JSON con el formato:

```
{ "respuesta": "Gracias X, es usted el visitante número Y" }
```

donde X deberá reemplazarse con el nombre del usuario indicado, e Y deberá reemplazarse con el número de visitas que ha habido al endpoint hasta ese momento, que deberá ir aumentando progresivamente. De igual

modo, por cada mensaje recibido por el endpoint se deberá añadir una línea con el formato:

`"Visita recibida número X, visitante: USUARIO, mensaje: CONTENIDO"`

a un fichero *visitas.txt*, reemplazando X por el número de visitas, USUARIO por el nombre del usuario y CONTENIDO por el mensaje recibido. Para este procedimiento será necesario hacer uso de los nodos *file* y *template*.

Así mismo, deberá haber un nodo de tipo *inject* que, en caso de ser activado manualmente, borrará el fichero *visitas.txt*.

El flujo generado debe exportarse completo.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

4. Crear un flujo en Node-RED con los siguientes nodos:

Tres nodos de tipo *inject*:

- El primer nodo debe inyectar el número 5 cada 10 segundos.
- El segundo nodo debe inyectar el número 1 cada 5 segundos.
- El tercer nodo debe inyectar el número 3 cuando el usuario lo active manualmente.

Un nodo función que recibirá los mensajes de los tres nodos anteriores y debe hacer lo siguiente:

- Calcular y almacenar la sumatoria de los números recibidos.

Por último, se debe mostrar por la consola de depuración el número recibido cada vez que se reciba un nuevo mensaje, junto con el valor actual de la sumatoria. Adicionalmente, si la sumatoria es par, se debe imprimir (mostrar por la consola de depuración) el mensaje “¡Es par!” y en caso contrario, el mensaje “¡Es impar!”.

El flujo exportado debe tener el nombre ejNodeRed.json.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

5. Diseñe un flujo de Node-RED donde existan dos nodos función. El primero de ellos, llamado `acumulador`, sumará el número que tome por entrada a un acumulador global al flujo, si el número es par; y lo restará al acumulador, si es impar. El segundo de los nodos, llamado `máximo`,

almacenará en una variable global al flujo el número máximo de los que haya ido recibiendo o recibirá por entrada.

Como entrada del flujo, se utilizará un nodo que genere un número aleatorio, el cual llegará a ambos nodos función.

Cada vez que se modique el acumulador o se encuentre un nuevo máximo, se deberá mostrar un mensaje indicando el valor del acumulador o del máximo, mediante un nodo de depuración.

Por último, escriba el código correspondiente a ambos nodos función.

Nota: El nodo `random`, al activarse genera un número aleatorio entre dos valores.

Debe explicar las características de los nodos utilizados y configuraciones aplicadas.

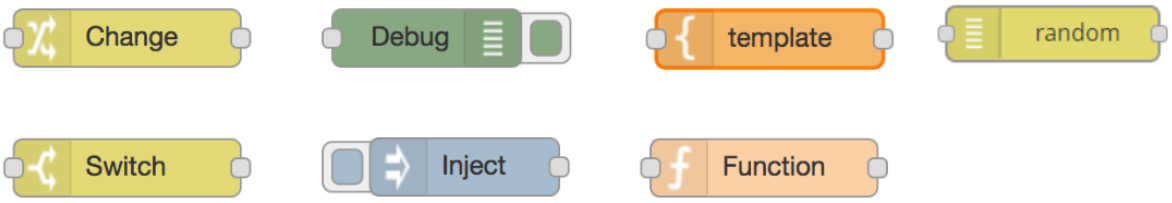

Fig 1. Lista de nodos que pueden utilizarse para crear el flujo.

6. Implementar en Node-RED una API REST que ofrezca, al menos, tres endpoints para realizar operaciones sobre vectores de enteros. Para ello se será necesario utilzar, entre otros, nodos de tipo "`http in`" y "`http response`".

Los datos se pasarán a los endpoints en formato JSON, y las operaciones a implementar serán:

- `/suma`: sumar dos vectores de tres posiciones. Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"a":[1,2,3],"b":[4,5,6]}'
-H "Content-type: application/json"
http://localhost:1880/suma

{"resultado":[5,7,9]}
```

- `/resta`: restar dos vectores de tres posiciones, en el orden vector "`a`" - `vector` "`b`". Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"a":[1,2,3],"b":[4,5,6]}'
```

```
-H "Content-type: application/json"
http://localhost:1880/resta

{"resultado":[-3,-3,-3]}
```

- `/multesc`: multiplicar un vector de tres posiciones por un escalar. Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"a":[1,2,3],"n":5}'
-H "Content-type: application/json"
http://localhost:1880/multesc

{"resultado":[5,10,15]}
```

Como mejoras puede practicar lo siguiente:

- Añadir `endpoints` para operaciones adicionales.
- Permitir realizar las operaciones con cualquier longitud de vector.
- Utilizar un nodo de tipo "`file`" donde se mantenga un `log` de todas las peticiones que se reciben. Cada vez que se reciba una petición, se almacenará en el fichero una nueva línea de texto, indicando el tipo de operación solicitada, parámetros y resultado.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

7. Implementar en Node-RED una API REST que ofrezca, al menos, tres `endpoints` para realizar operaciones sobre cadenas. Para ello se será necesario utilzar, entre otros, nodos de tipo "`http in`" y "`http response`".

Los datos se pasarán a los `endpoints` en formato JSON, y las operaciones a implementar serán:

- `reverse`: calcular la inversa de de una cadena. El formato del JSON de entrada deberá ser:

```
{
      "cadena": "Hello, world",
}
```

El formato devuelto por el endpoint será similar al de entrada. Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"cadena": "Hello, world"}'
```

```
-H "Content-type: application/json" \
http://localhost:1880/reverse

{"cadena":"dlrow ,olleH"}
```

- `/change_case`: pasar una cadena a minúsculas o mayúsculas. El formato del JSON de entrada deberá ser:

```
{
      "cadena": "Hello, world",
      "modo": "mayus"
}
```

Donde el valor del elemento modo puede ser mayus o minus según queramos pasar la cadena a mayúsculas o minúsculas.

`El formato devuelto por el endpoint será el siguiente:`

```
{
      "cadena": "HELLO, WORLD"
}
```

Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"cadena": "Hello world", "modo": "minus"}'
-H "Content-type: application/json"
http://localhost:1880/change_case

{"cadena":"hello world"}
```

- `/random`: generar una cadena aleatoria de la longitud indicada. El formato del JSON de entrada deberá ser:

```
{
      "longitud": 12,
}
```

El formato devuelto por el endpoint será el siguiente:

```
{
      "cadena": "kUqHWtaXFQBV"
}
```

Ejemplo de interacción con el endpoint mediante `cURL`:

```
$ curl -X GET -d '{"longitud": 12 }'
-H "Content-type: application/json"
http://localhost:1880/random

{"cadena":"U28QKFmX5Ian"}
```

Como mejoras puede practicar lo siguiente:

- Añadir `endpoints` para operaciones adicionales.
- Añadir soporte de errores cuando el formato del JSON de entrada no es correcto.
- Utilizar un nodo de tipo "`file`" donde se mantenga un *log* de todas las peticiones que se reciben. Cada vez que se reciba una petición, se almacenará en el fichero una nueva línea de texto, indicando el tipo de operación solicitada, parámetros y resultado.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

8. MQTT. Suscribirse a un broker MQTT mediante un nodo de tipo "`mqtt in`". El broker al que os debéis suscribir está en la dirección https://test.mosquitto.org/, puerto 1883. Éste es un servidor público y gratuito para realizar pruebas con el protocolo MQTT.

Entre los distintos eventos disponibles, deberéis suscribiros a aquel con topic `/merakimv/Q2GV-Y4R8-5Z3L/light`

El evento nos dará de forma regular la luminosidad en luxes que registra una cámara de seguridad. La estructura del evento (payload del mensaje) es de la forma "{"lux": 80.9}". El payload llegará en formato `string`, deberéis utilizar un nodo de la clase `parser` para transformarlo a un `JavaScript Object` y poder trabajar fácilmente con esta información.

Mediante un nodo `function`, deberéis almacenar el valor máximo registrado de todos los eventos recibidos. Para ello será necesario utilizar el `contexto` de Node-RED.

Cada vez que se reciba un evento, se debe mostrar, por la consola de depuración, tanto el valor de luminosidad recibido como el máximo registrado hasta este momento.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

9. MQTT. En este flujo tendremos que crear, de forma paralela, un emisor y un suscriptor que se conecten a un broker MQTT. El broker al que os debéis suscribir está en la dirección https://test.mosquitto.org/, puerto 1883. Éste es un servidor público y gratuito para realizar pruebas con el protocolo MQTT.

El `topic` en el que escribir será `uca/sd/YEAR/contador/IDENTIFICADOR`, donde YEAR será el año actual e IDENTIFICADOR será una cadena aleatoria que el alumno elija para evitar colisiones con otros compañeros.

El emisor deberá publicar una vez por segundo un JSON con un campo contador, que contendrá un entero que irá aumentando en cada publicación.

El receptor se suscribirá al `topic`, recibiendo las notificaciones periódicas, y deberá mostrar el mensaje.

`El contador tiene el valor X` a través del nodo `debug`.

Consejos:

- La comunicación con el broker `mqtt` se hace mediante los nodos `mqtt in` y `mqtt out`.
- Para realizar una publicación periódica, revisar las opciones de repetición del nodo `inject`.
- Para mantener un contador, revisar el uso de los `contextos` en funciones.

Como mejoras puede practicar lo siguiente:

- En lugar de mantener un contador en memoria, leer el dato de un fichero en el emisor, y guardarlo en otro fichero en el receptor.

Por último, explicar las características de los nodos utilizados y configuraciones aplicadas.

Información adicional:

Puede consultar los recursos disponibles en:

- https://nodered.org/docs/user-guide/
- https://www.w3schools.com/js/default.asp
- https://www.postman.com/