# Distributed PUB/SUB Exercise Report

### Javier Rodríguez, 16/07/2018

## Summary

I have implemented a C# producer with a class that generates random numbers and publish them by redis in a JSON format. A Node.js server is subscribed to the same redis channel and get those random numbers. This server communicates with Angular UI by websockets, so TCP connection doesn’t finalize, allowing angular receiving in a non-stopped way that random numbers generated by the producer.

Only one instance (with two axis) of producer is invoked, so all the clients will have the same data. There has not been implemented any secure form, allowing any client to subscribe or publish in the channel.

## System

As it actually is my first C# project, I decided use Visual Studio as IDE, so I use Windows 10 as OS. Due Redis has no official support to Windows, I have use ubuntu bash for windows in order to install redis‑server and use it, that in my opinion is more comfortable than third party supports. I have chosen JSON format for all messages.

## Producer

The C# app has a class that implement the number generator and the main class.

The **number generator class** has as properties configurable variables (min and max range and an extra for time elapsed between generations), a name for console logging purposes, redis subscriber that should be pass by constructor and finally a cancellation token source and a status string for threading purposes.

The class has also three methods:

**Start Method:** If status is STOPPED, then it creates a new task (with a cancellation token) that will generate and send to redis channel numbers each time given by property Time until the token has a cancellation request. It starts that task and change the status property to STARTED.

**Stop Method:** If status is STARTED, then it requests a cancellation to the token. It also changes the status property to STOPPED.

**Update Method:** It update all configuration properties of the class given by a Dictionary, that will be the result of transform an incoming JSON.

The **main class** manage communications and the console logging. It creates the connection with redis server, the both axis instances of the number generator class, and subscribes to the channel of redis. Depending on the action type of the received message, it will do one or another action (update, start or stop any of the two axis).

## WebServer

I have implemented express.js for the web server. This server is an intermediary between Angular UI and redis, and in order to maintain the real-time behavior, I have decided to implement websockets for the communication between this server and Angular UI. Redis, sockey.io and express.js dependencies are very light, so this server is light too (~6MB with node\_modules). It’s only function is publish on redis what it receives from websockets and send by websockets received data from redis subscription.

## AngularUI