

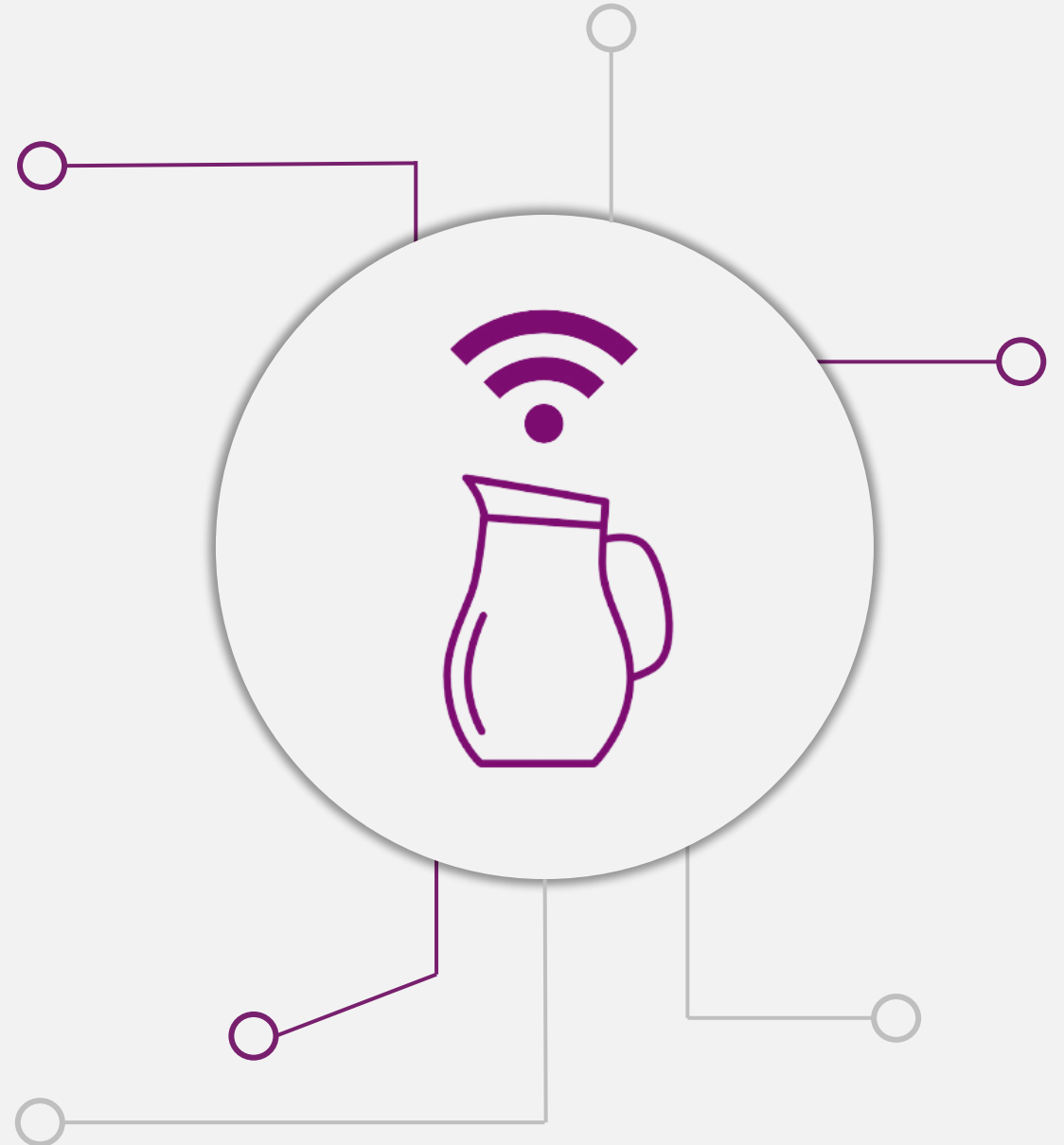
# Smart Jugs

## IoT - Project

### Authors

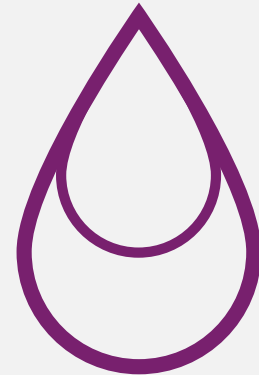
Kevin Cattaneo – S4944382

Riccardo Isola – S4943369



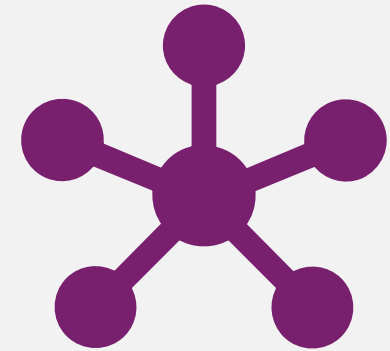
# Overview

- Our project involves the connectivity of “Smart (Water) Jugs” and the tracking of the **fluid filtered** each second.
- We track:
  - The total number of liters filtered
  - The amount of water consumed each day
  - The filter usage
  - The kilograms of plastic saved
  - The location
- Coupled with an application developed in parallel with the **Mobile Development** course. The application part represent the user / client dashboard.



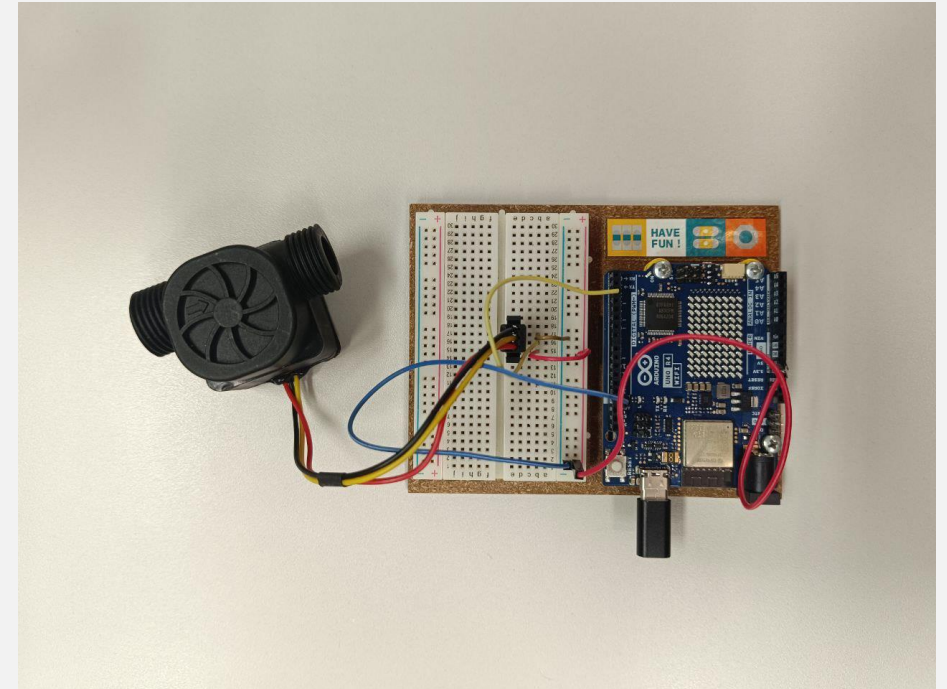
# Technologies

- The technologies we adopted are the following ones:
  - **Arduino** – as hardware for emulating a sensor
  - **TypeScript** – to have a scalable simulated number of sensors
  - **Godot** – to have a gamified version of the simulated sensor
  - **Node-RED** – representing the middleware between ThingWorx and the mobile devices, also used for dashboards. Authentications done via JWT Tokens.
  - **ThingWorx** – used mainly as remote database to save things data persistently
  - **Redis** – cache service used to limit the traffic of requests to ThingWorx
- We will discuss each one in the following sections.



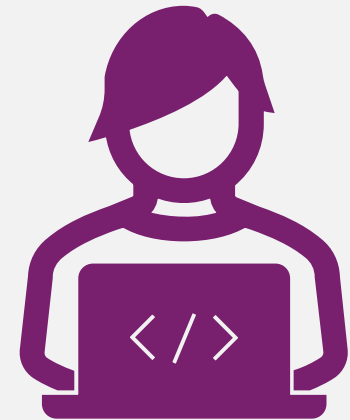
# Arduino

- Arduino represents the **hardware** version of the sensor and represents the **physical jug** itself or an “**adapter**” that can be put on the pouring part of the jug (with a smaller version of course).
- The script that executes does the following:
  - Open a **Wi-Fi hotspot** self-hosted by the Arduino
  - Pair the sensor with the mobile device and link itself to a user
  - After the first step, to send data via **MQTT** directly to ThingWorx
- The pairing is done via a self-hosted Wi-Fi from the Arduino itself, on which the client Wi-Fi credentials can be transmitted: in this way the Arduino connects to Internet and start publishing via MQTT



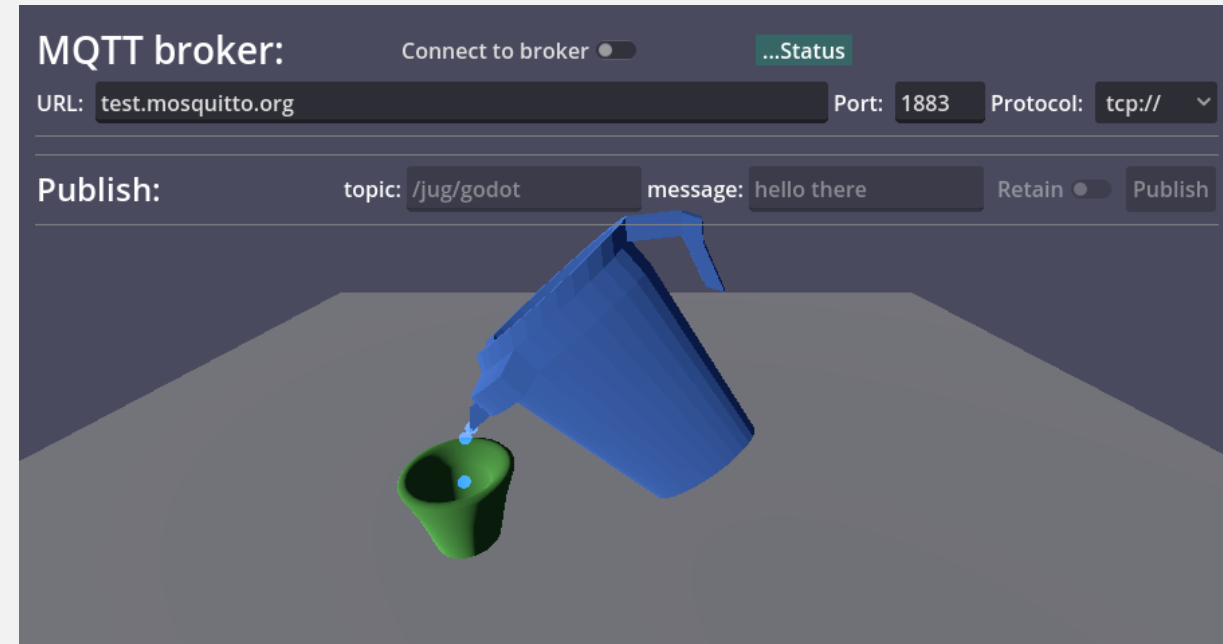
# TypeScript

- TypeScript allowed us to write some APIs with which both the sensor and mobile device could interface
- The peculiarity of the code is the fact that allowed us to test our implementations on a **scalable number of sensors** (since, for example, owning 10 Arduino cards would be infeasible)
- This kind of simulator behaves exactly like it's the hardware counterpart does
- We developed using **Bun** instead of NodeJS, a popular JavaScript runtime that provides faster performance, simpler APIs, and better development experience



# Godot

- We want also to spice the project with something extra, like the interiorization of the **gamification** process in these topics
- We delve into a **3D implementation** of a jug and tried to simulate the behavior of the pouring of a jug
- Please note that while the entire scene has been made by us, the MQTT implementation of this part has been taken externally from GitHub, and adapted to our objectives, since knowing how Godot works with MQTT was not our primary focus. Thanks to: <https://github.com/goatchurchprime/godot-mqtt?tab=readme-ov-file>

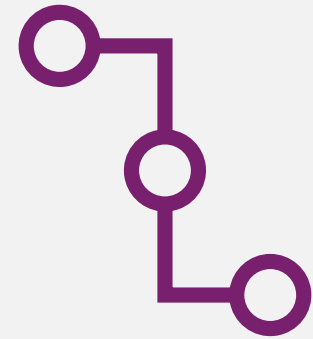


*Note also that this implementation doesn't send data to ThingWorx, we just implemented the Node-RED listener on MQTT.*

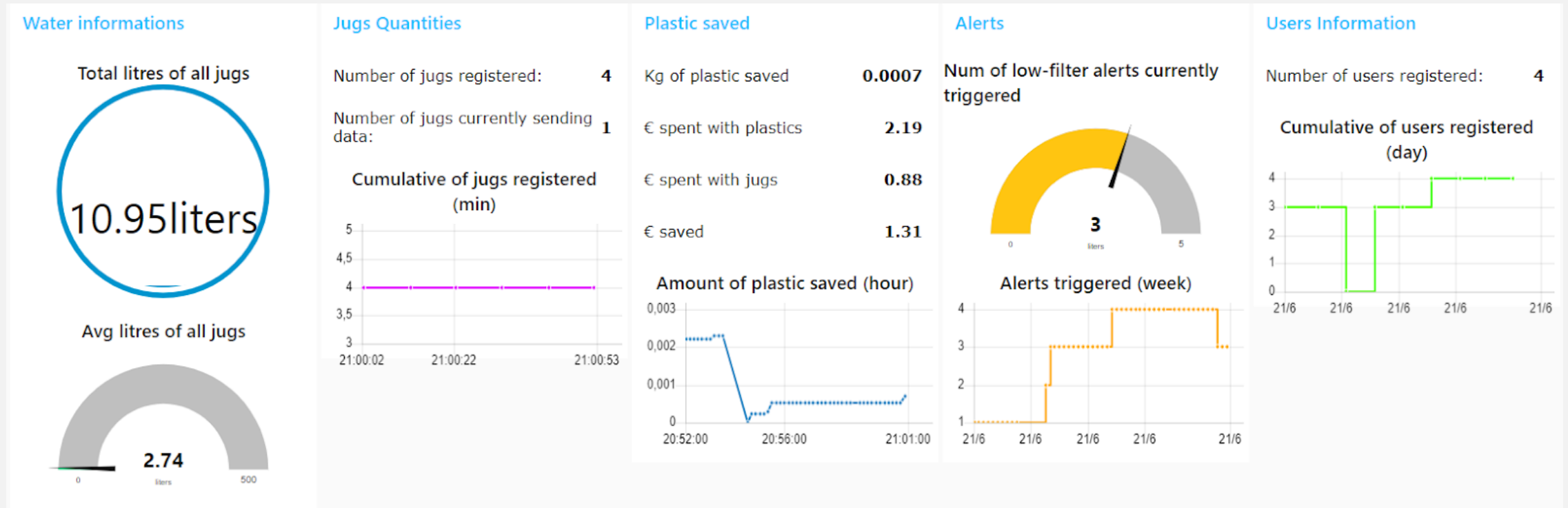
*With the correct parameters, this could be also made work to send data to ThingWorx.*

# Node-RED

- Represents the **middleware** (“fog”) that implements all the APIs needed to make the mobile application works with the IoT system.
- Contains also the APIs needed to the jug sensor to enable pairing.
- We also implemented the enterprise **dashboard** with it and split it in different categories.
- Any operation involving the authentication to perform operations on jugs and so to verify the effective owning of a jug from a user has been made by verifying a **JWT Token** assigned univocal upon login to the user.



# Dashboard #1 - Overview



This is the main dashboard that the **employee** of SmartJugs can see and have an overview about the current registered jugs and the liters filtered **in real-time**.



# Dashboard #1 - Details

In this dashboard the following data are displayed:

- **Water information:** real-time information about the total and the avg liters poured until now from the jugs
- **Jug Quantities:** how many jugs have been paired by the users of SmartJugs
- **Plastic saved:** a metric to observe how much kilograms we saved instead of use plastic water bottles (see also next slide)
- **Alerts:** those alerts are implemented application-side and marks the number of mobile notification sent because the filter of a certain jug is about to expire
- **User information:** we track how many users registered and the curve in the current day



# Plastic saving

## An ecological and financial parenthesis

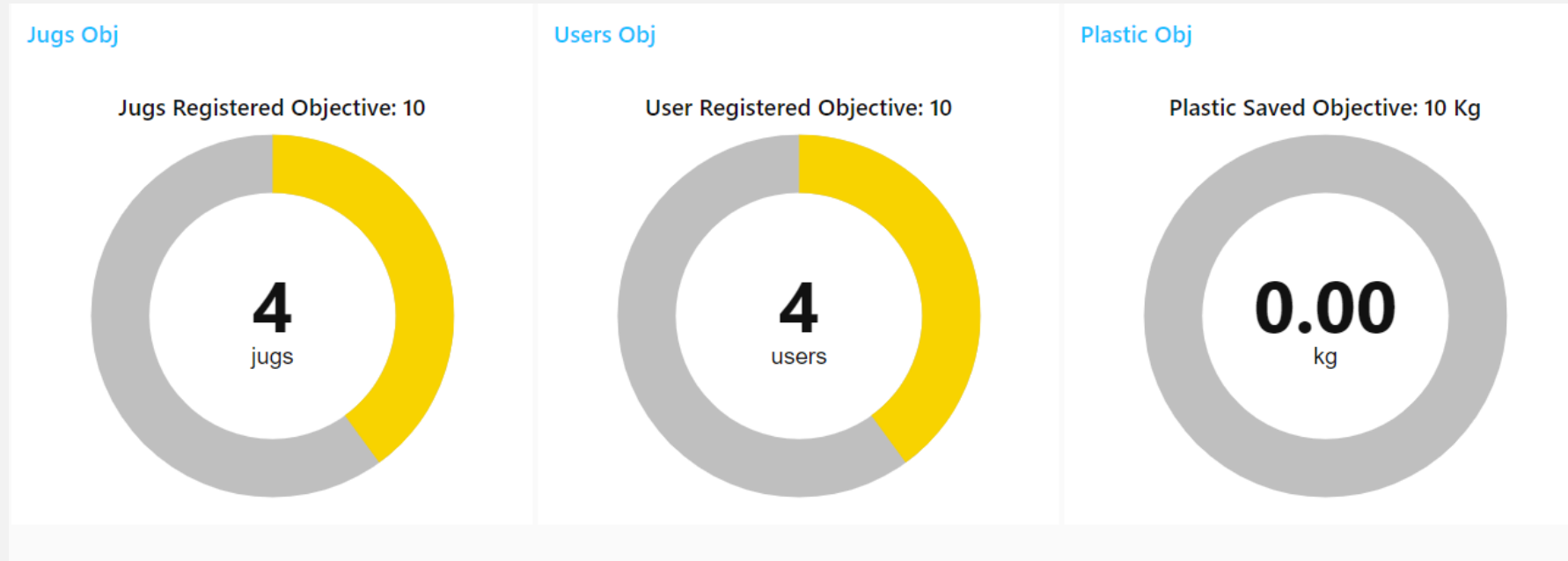
Since we are talking of water and we know the **problem of plastic** in the modern days, we also felt the necessity to talk about plastic.

In our dashboard we provide two main categories of metrics for plastics, the first that weight the **environment** and the latter that weight the **money** topic:

- **Kg of plastic saved:** with this metric, we computed the equivalent of plastic water bottles weight in terms of water consumed
- **€ saved by using jugs:** with this metric, we observe the difference into use jugs compared with buying the equivalent in plastic water bottles



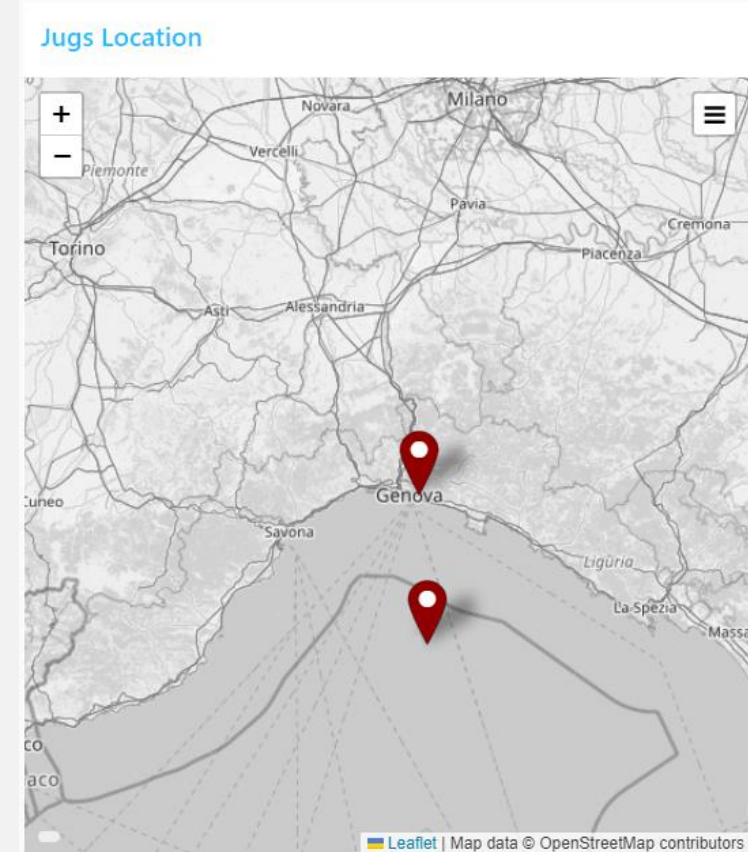
# Dashboard #2 - Objectives



We also thought about the possibility to fix some **objectives** in the company and see how far we have got.

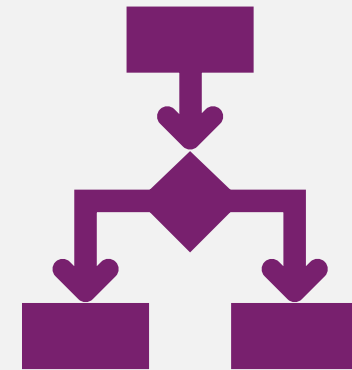
# Dashboard #3 - Location

- We decided to **track the jug position** on pairing process, by taking the position from the mobile device and to show it on a map. We do not track the user movement. The user can choose if share the approximate position or no position at all.
- On the map provided as example, the jug was located at DIBRIS (University of Genova), we can see:
  - The **approximate** position, on the bottom
  - The **precise** position, on the Genova label, set just for presentation purposes (not included in implementation)
- Each mark holds the coordinates and the jug name.



# ThingWorx

- We use the ThingWorx platform to create the things and **save persistently** the data.
- In particular of ThingWorx we used:
  - **Properties:** to save things details (e.g. liters per second, total liters)
  - **Shape:** to represent the jug filter details
  - **Template:** to represent in a scalable and efficient way each thing, implements the interface above
  - **Value Stream:** to log the total liters filtered and retrieve it to achieve aggregation per day, per week etc. on Node-RED



# Redis

- We implement a Redis **cache service** (as a separated server) to achieve for efficiency in our implementation
- The main objective was to **limit the HTTP traffic** to ThingWorx when getting the data stream, since we observed a certain **fragility** when we scale to a very big number of things



**Thanks for your attention**

