



Università
di Genova

DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Smart Jugs - Documentation

Internet of Things - University of Genova - a.a 2023/24

Authors:

Cattaneo Kevin - S4944382

Isola Riccardo - S4943369

Index

Index.....	2
Overview.....	3
Sensors.....	3
Arduino.....	3
Simulator in TypeScript.....	3
Simulator in Godot - Gamification process.....	4
Sensor Data.....	5
Platforms.....	5
Node-RED - “Fog”	5
ThingWorx.....	5
Redis.....	5
Dashboards.....	6
Client-Side (Mobile application).....	6
Enterprise-Side (Node-RED).....	6
Overview.....	6
Company Objectives.....	6
Location.....	7
Architecture.....	8
Opinions.....	8

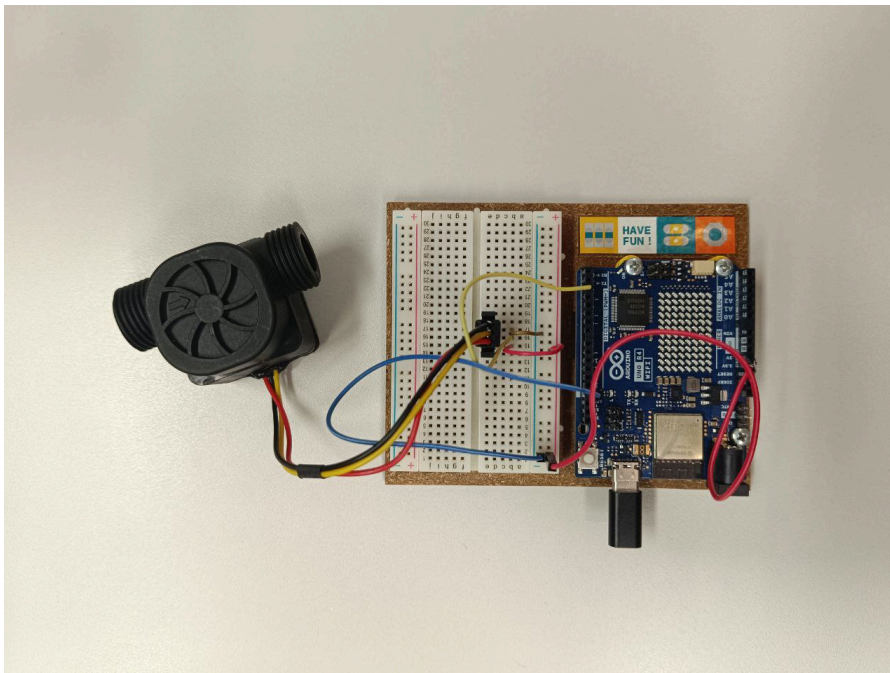
Overview

Our project involves the connectivity of “Smart (Water) Jugs” and the tracking of the fluid filtered each second. We also compute some aggregate information like the total number of liters filtered, the amount of water consumed each day and the quality of the filter. Our implementation interfaces with both a dashboard from an enterprise view and one from a client view; for the latter case we use an application developed in parallel with the Mobile Development course, that provides the same amount of information gathered and personalized per each jug and also a localization.

Sensors

Arduino

We implemented a version of the sensor via Arduino hardware, and a script that allows us to pair the jug to the phone to link the jug to a user and then data via MQTT protocol to ThingWorx.

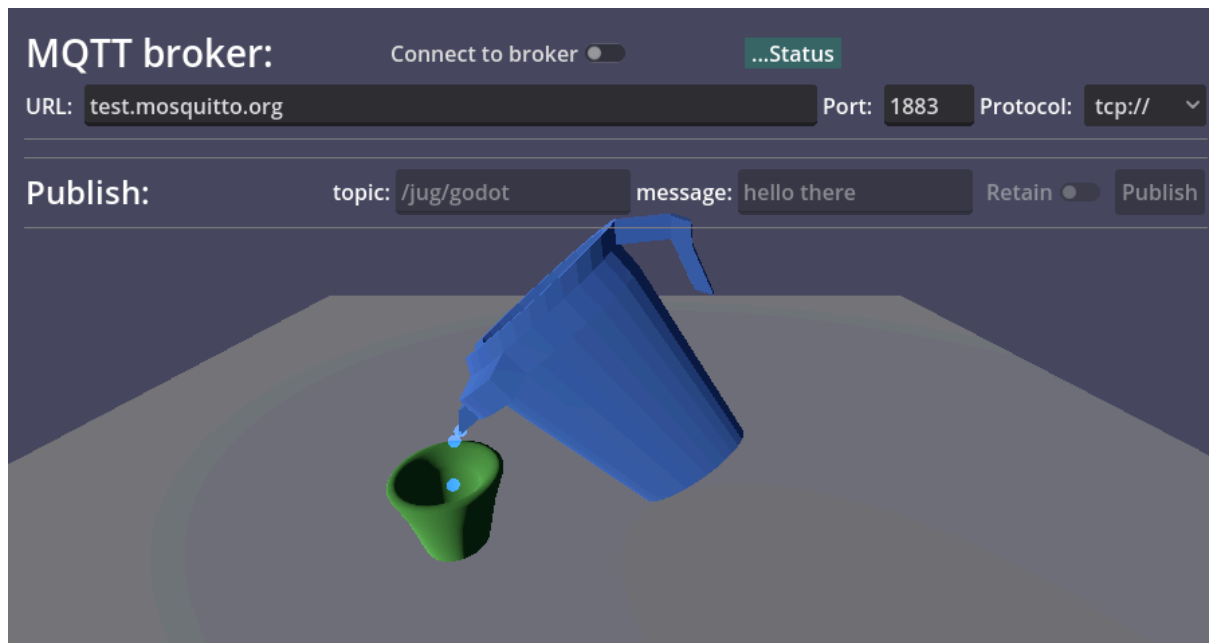


Simulator in TypeScript

We implemented a version of the sensor via a simulator that allows us to scale freely the number of sensors and so the amount of data that they send. This simulator performs different calls just like the arduino does. Just like that, the data that are simulated (randomly) are also sent on the ThingWorx platform.

Simulator in Godot - Gamification process

We implemented a version of the sensor also via Godot Software. In the spirit of gamification we tried to see how the MQTT works also in this case. The main idea is to simulate the pouring of water via 3D Models done by us and to see the data that are sent. Notice that those data will not be part of the pool included in ThingWorx and are just listened for testing purposes on Node-Red; but surely can be shifted to ThingWorx. Notice also that the implementation of MQTT script in GDScript (Godot Script) is not made by us, we used the following add-on by <https://github.com/goatchurchprime/godot-mqtt.git>. We then personalized it following our needs, without delving too much or refining the original implementation.



Sensor Data

The main sensor we use is a fluxometer, some data is also taken from the mobile device. The data we retrieve and show in the NodeRED dashboard represents the following parameters:

- Total consumption of water from all the jugs
- Quantity of plastic saved (by using a jug)
- Counting the number of low-filter alerts (notification sent to mobile device)
- Positions of jugs (taken via mobile device)

Platforms

Node-RED - “Fog”

The “fog” of the backend is implemented entirely via Node-RED. Here we handle the request of pairing done by the jug to the application and the other APIs that interfaces with it. For some data, e.g. user data and their jugs, we save those data into a PostGres database. We pushed Node-RED to its extremes, by implementing also all the APIs needed for interfacing with the mobile devices.

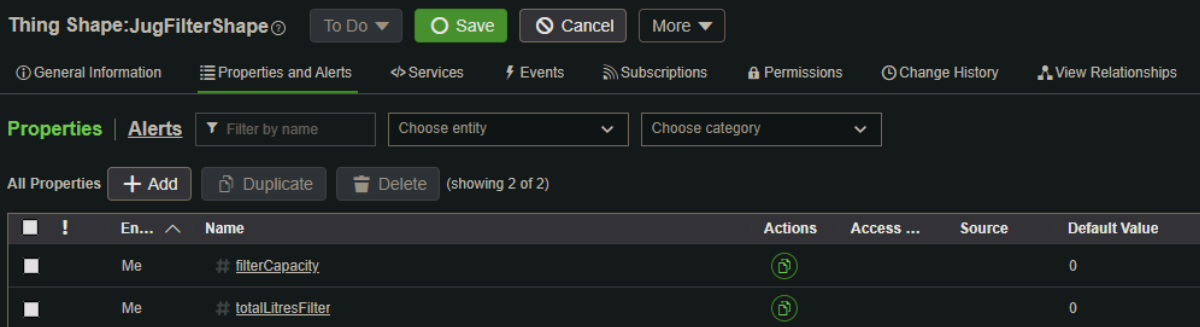
The enterprise dashboard is implemented here and polls directly from the ThingWorx platform.

ThingWorx

We use ThingWorx mainly as a database to store things (jugs) data that arrive each second. This platform presents the properties of each jug, implemented as a Template for the properties about the fluid (e.g. total liters filtered) that extends a Shape that holds the filter properties (e.g. filter capacity).

We also have a Value Stream that sends the amount of total liters filtered correlated with a timestamp that we use to implement a time series in the mobile application.

Shape



The screenshot shows the 'Thing Shape: JugFilterShape' configuration page in ThingWorx. The interface includes a top bar with 'To Do', 'Save', 'Cancel', and 'More' buttons. Below this is a navigation menu with 'General Information', 'Properties and Alerts' (selected), 'Services', 'Events', 'Subscriptions', 'Permissions', 'Change History', and 'View Relationships'. The 'Properties and Alerts' section is active, showing a 'Filter by name' dropdown and 'Choose entity' and 'Choose category' dropdowns. Below these are buttons for '+ Add', 'Duplicate', and 'Delete', with a note '(showing 2 of 2)'. The main table lists two properties:

	En...	Name	Actions	Access ...	Source	Default Value
■	Me	# filterCapacity				0
■	Me	# totalLitresFilter				0

Template (implements the shape)

Thing Template:JugTemplate

To Do Save Cancel More

General Information Properties and Alerts Services Events Subscriptions Configuration Permissions Change History View Relationships

Properties Alerts Filter by name Choose entity Choose category

All Properties Add Duplicate Delete (showing 10 of 10)

	En...	Name	Actions	Access ...	Source	Default Value
Me	##	litresPerSecond				0
Me	##	totalLitresJug				0

Redis

We use Redis as a cache service for storing jugs data without requesting each time the entire stream of data from the remote server of ThingWorx. Doing so we speed up the process by contacting the Redis server (in our self-made enterprise). After a certain amount of time via Node-RED we update the Redis data for each jug by polling ThingWorx.

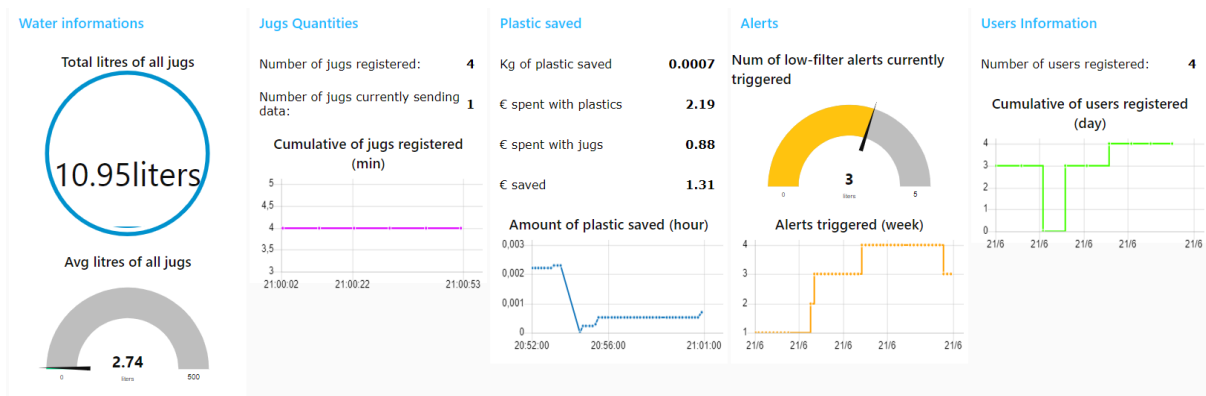
Dashboards

Client-Side (Mobile application)

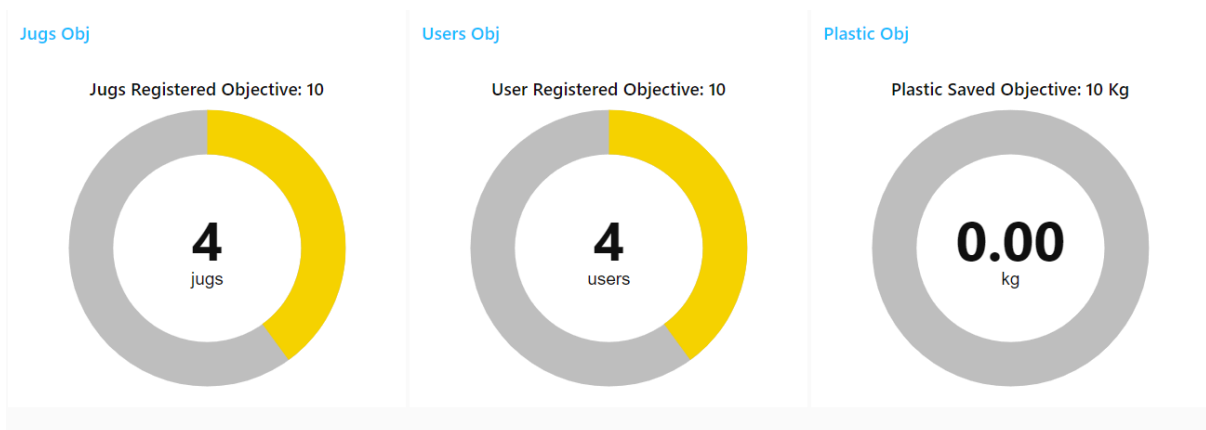
<photo>

Enterprise-Side (Node-RED)

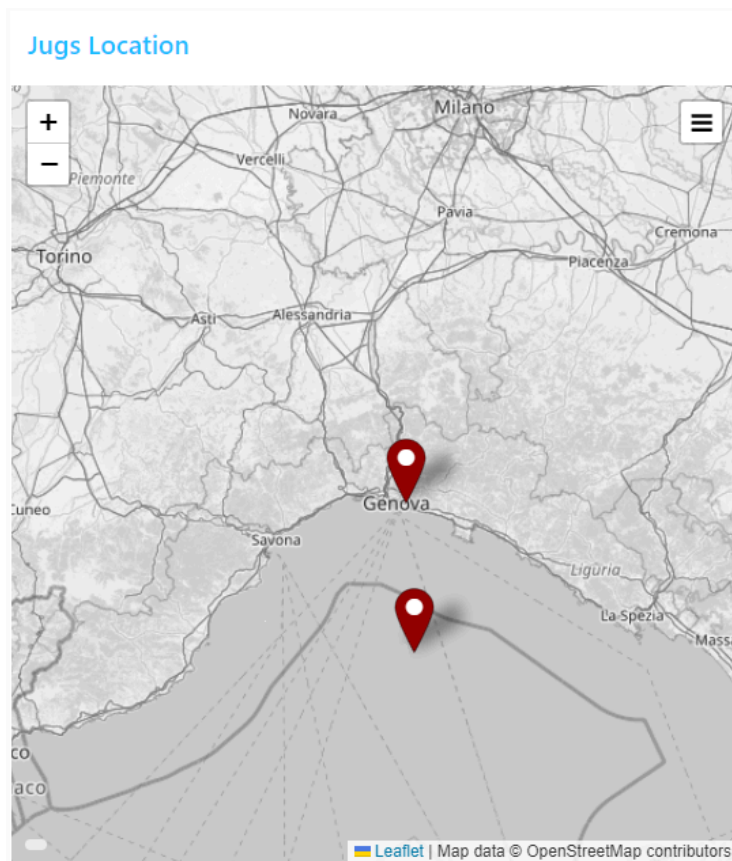
Overview



Company Objectives



Location



The position of the device was located in the University of Genova.

In the previous photo we can see the difference between **precise location** (mark straight on the university) and **approximate one** (in the sea, that is in a ray around the precise one).

Architecture

- TODO: do high-level design via mermaid

Opinions

- NodeRED is an easy platform to use until the system we want to represent grows in size and complexity: this makes even some steps of computation represented into a great amount of blocks.
- ThingWorx documentation usually was poor or inaccessible since it was necessary to pay a license.
- ThingWorx provided server was usually down, so it slowed up the implementation process. On the other hand, it gave us the idea of implementing a Redis cache to limit the continuous traffic to ThingWorx.
- Arduino libraries usually were abandoned: there is a case of needing to modify the header of an arduino library to make it work (see comment in the script).