Intelligent Irrigation System for Low-cost Autonomous Water Control in Small-scale Agriculture

Deliverable D2.1a

First report on specifications & functionalities of the edge-enabled sensor-gateway framework for smart irrigation system

Responsible Editor:	WAZIUP
Contributors:	UPPA
Document Reference:	INTEL-IRRIS D2.1a
Distribution:	Public
Version:	1.1
Date:	February 2022

Contributors table

DOCUMENT SECTION	AUTHOR(S)	REVIEWER(S)
SECTION 1	C. Pham & A. Rahim	F. Markwordt
SECTION 2	F. Markwordt & C. Pham	A. Rahim
SECTION 3	Johann Forster	C. Pham
SECTION 4	C. Pham	F. Markwordt
SECTION 5	F. Markwordt & Johann Forster	C. Pham
SECTION 6	C. Pham	A. Rahim

Document revision history

Version	Date	Changes
V1.1	Feb 7th, 2022	Public release
V1.0	Feb 7th, 2022	First draft version for internal approval
V0.1	Feb 3rd, 2022	First release for review

Executive Summary

Deliverable D2.1a describes the edge-enabled IoT gateway framework that will be used in the INTEL-IRRIS project. These results are the initial outcomes of Task 2.1 in WP2.

Table of Contents

1. INTRODUCTION

1.1. Edge-enabled sensor-gateway framework for smart irrigation system

The objective at the Edge-enabled IoT gateway is to have an autonomous gateway capable of collecting data from the soil devices and capable of displaying these data in a user-friendly manner. When targeting the smallholder farmers, the gateway should be able to simply display irrigation notifications on a small embedded device/screen, in addition to the more advanced and embedded web interface that would be accessible through a smartphone or a tablet. An example is provided below in the form of an illustration where the edge-enabled IoT gateway is the control part.

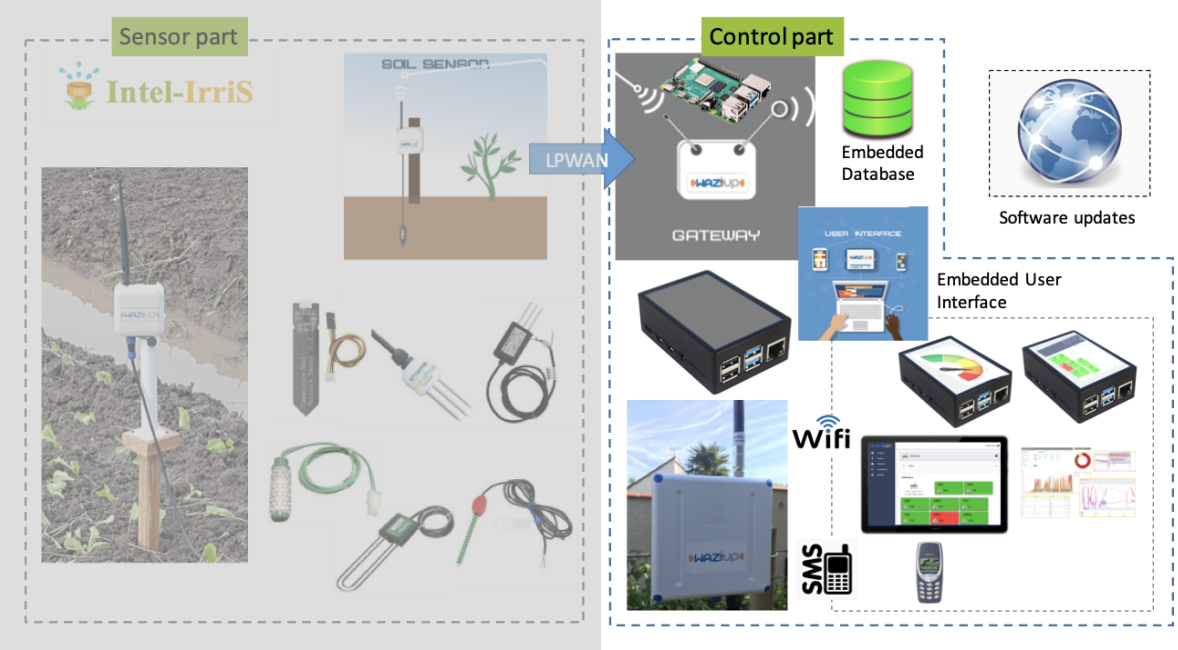


Figure 1 – The control part represented by the edge-enabled gateway

1.2. Specification & Functionalities

The INTEL-IRRIS's edge-enabled IoT gateway will be built on the low-cost, versatile, embedded and open IoT gateway expertise from both UPPA and WAZIUP. The main desirable features of the IoT gateway in INTEL-IRRIS, besides receiving sensor data from deployed soil devices, are:

- Implement the "Intelligent Irrigation in-the-box" with "plug-&-sense" approach
  - take into account the complex water-soil-plant interaction
  - embed Decision Support System (DSS), disruptive Artificial Intelligence (AI)
  - integrate various knowledge streams to present relevant indication and recommendations

    - Integrate of irrigation-specific software application modules including adaptable User Interface depending on smallholder farmers profile to provide all the necessary "control" components of a smart irrigation system
- Provide display capability of irrigation data for the end-user
- Integration of software modules for gateway maintenance and administration

2. The WaziGate framework

In the following the WaziGate framework, with all its components is described. Further or more detailed information is available on WAZIUP technology website[1].

2.1. General description & main features

WaziGate is an IoT LoRa Gateway developed over the years by WAZIUP in the context of several international research & development projects, further developed and enhanced under INTEL-IRRIS. The WaziGate framework is suitable to implement the edge-enabled IoT gateway approach: customized applications can be hosted in the gateway and the gateway can easily work without Internet connectivity and still provides data to end-users through its embedded database and web-based visualization module. WaziGate features are:

- Edge capacity to host your applications
- LoRa communication up to 10-12 Km
- Permanent WiFi hotspot enabling easy access to the gateway
- WiFi/3G/Ethernet internet connection
- Data upload with HTTP, MQTT or even SMS
- Low power consumption
- Automation
- Remote management

You can find all the WaziGate documentation on the WaziGate documentation page[2]. There are 4 main sections describing the WaziGate main features:

- Quick start[3]
- Installation[4]
- LoRaWAN[5]
- WaziGate Apps[6]

[1] https://www.waziup.io
[2] https://www.waziup.io/documentation/wazigate/
[3] https://www.waziup.io/documentation/wazigate/v2/quick_start/
[4] https://www.waziup.io/documentation/wazigate/v2/install/
[5] https://www.waziup.io/documentation/wazigate/v2/lorawan/
[6] https://www.waziup.io/documentation/wazigate/v2/waziapps/

2.2. Edge-enabled feature: WaziGate Apps

WaziGate software framework uses a microservice architecture to manage its Apps. This architecture allows each application to operate independently in isolation which makes the development and maintenance much simpler than a monolithic architecture. In the following, there is a component diagram of the WaziGate software architecture given. The different components represent or are realized with help of docker containers, the connections between them show the network protocols. Located in the top one can find the userdomain, the middle part represents our software on the gateway and the bottom shows the hardware.

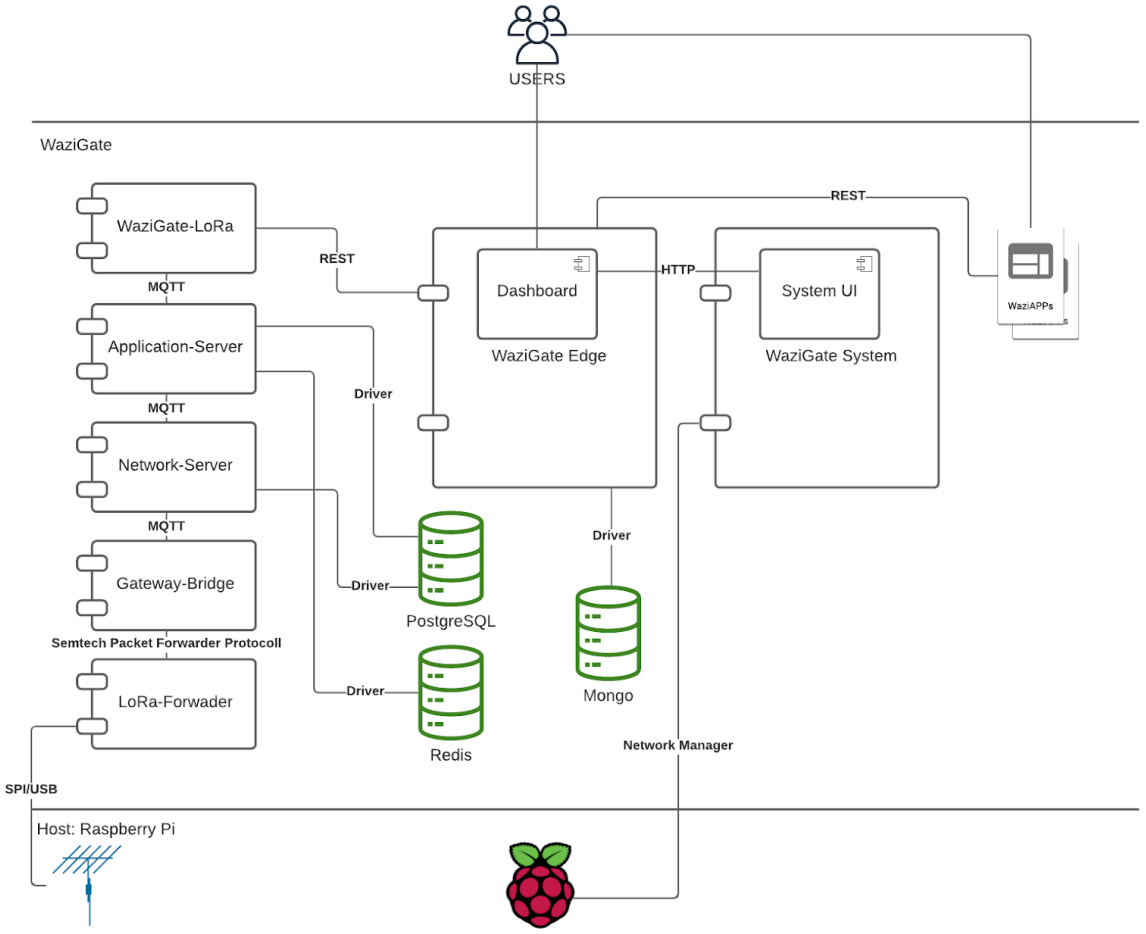


Figure 2 – Component Diagram of the WaziGate

Apps can be written in different programming languages. Apps communicate with other Applications through a set of restful APIs, the used data format is json. On top of the Raspbian OS is a Docker Engine, which handles all necessary images/containers that represent our platform. The app component is isolated in a docker container, the docker

images are uploaded and available on dockerhub[7]. The big advantage is the isolation of different components, every container can be updated independently.

The Wazigate App Manager pulls compatible Docker images from dockerhub and integrates them via the Docker Engine.

Detailed steps on how to create your own applications are available from WaziGate Apps.

2.3. Local data visualization

The gateway holds its data inside a local MongoDB database. The user interface is able to visualize readings of sensors and actuators values. The following screenshot gives a quick example about the simple functionality of the data visualization components.

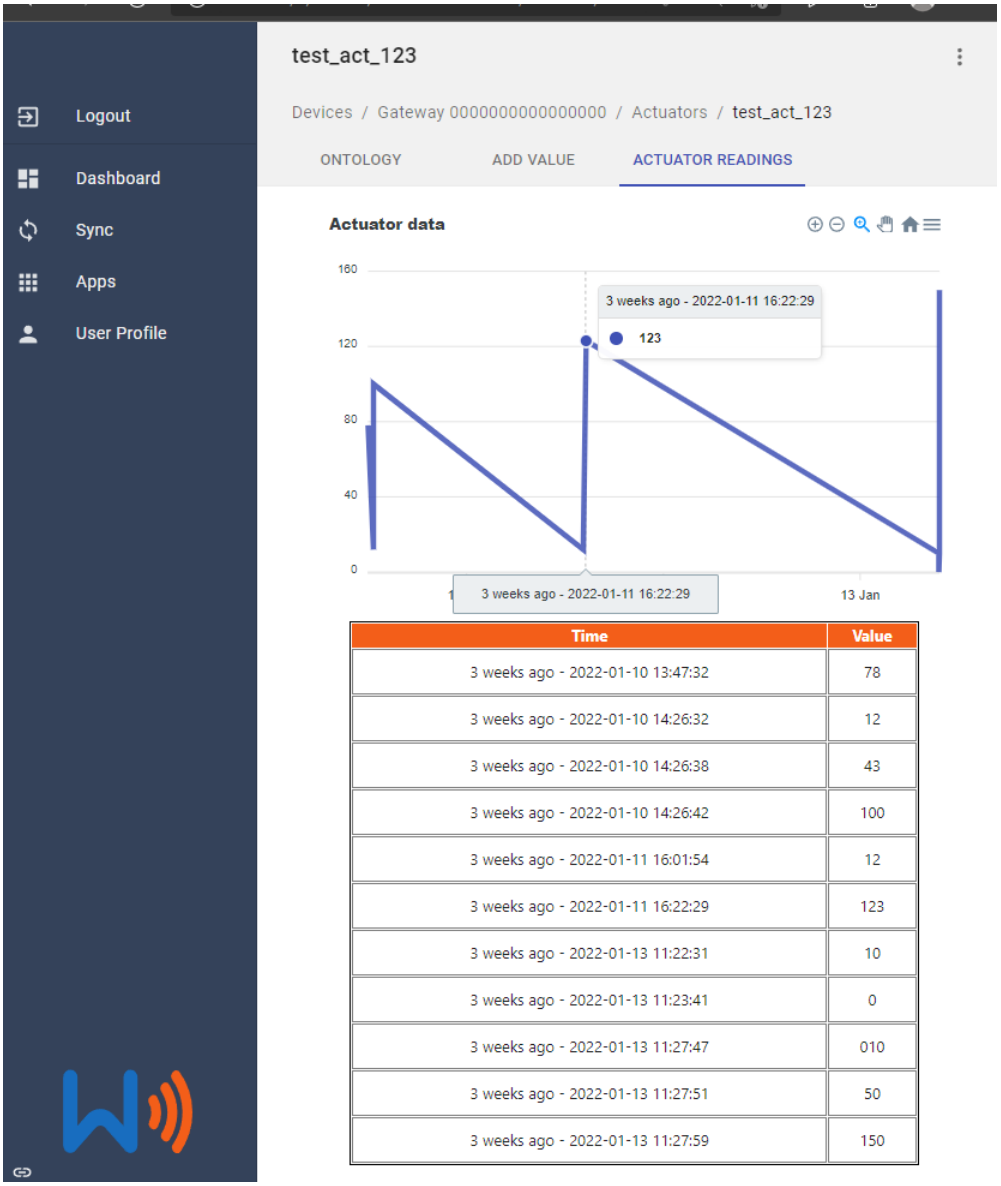
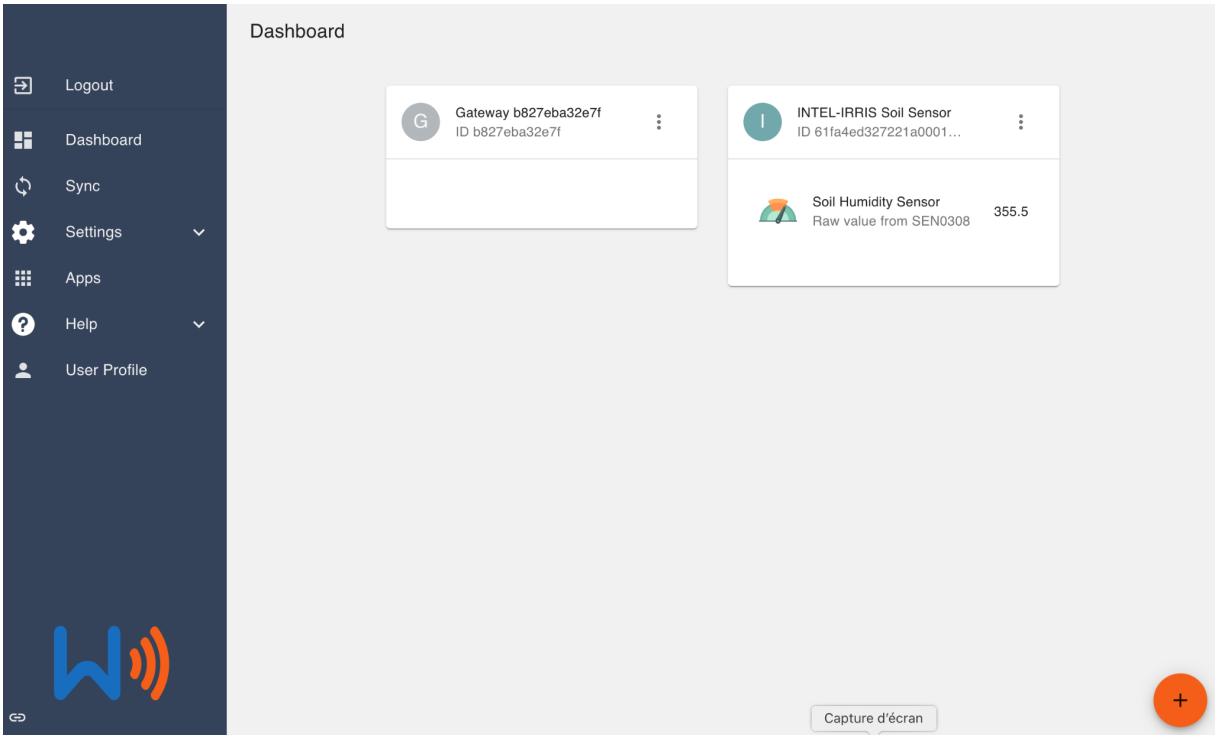


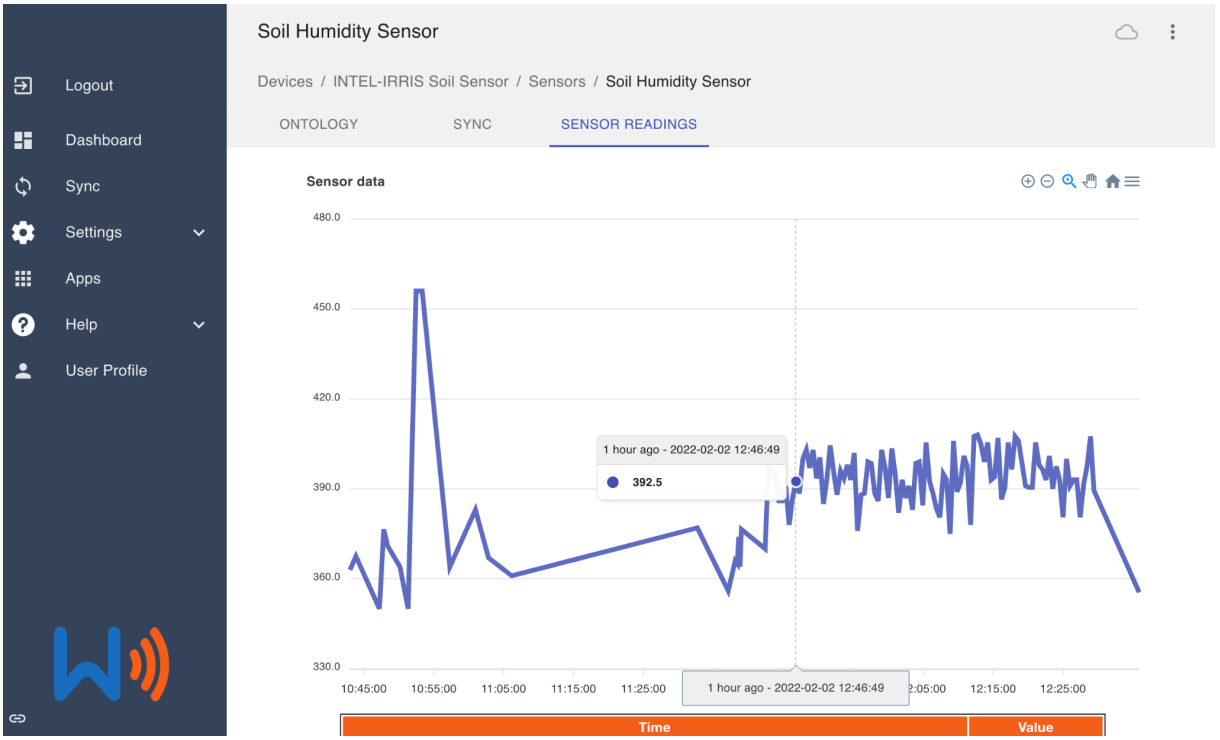
Figure 3 – Readings of an actuator on the WaziGate

[7] https://hub.docker.com

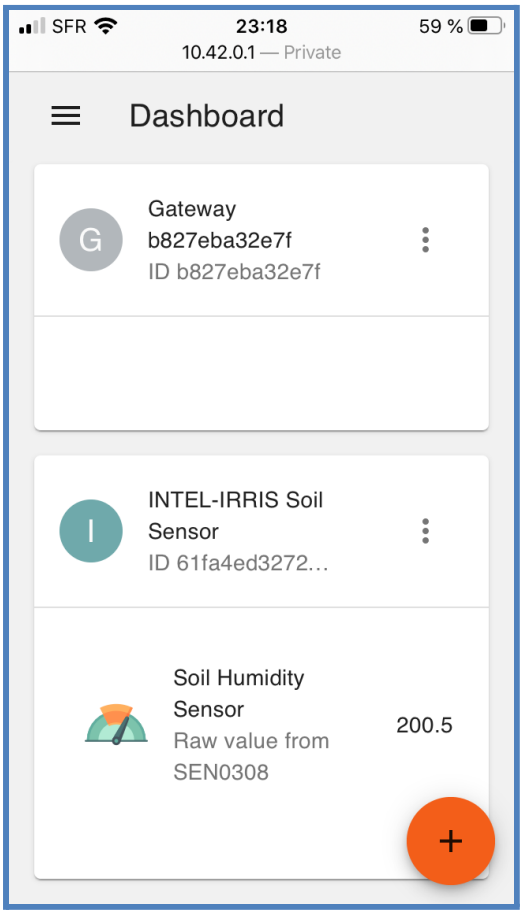
In the figure above the graph and the table show the values of the actuator “test_act_123”. The graph allows you to zoom in and out, visualizing the data points with values and timestamps. There is an option to export the diagram as an image, as pixel or vector graphic, or as plain data or CSV spreadsheet format. The table shows all the timestamps and values at a glance.

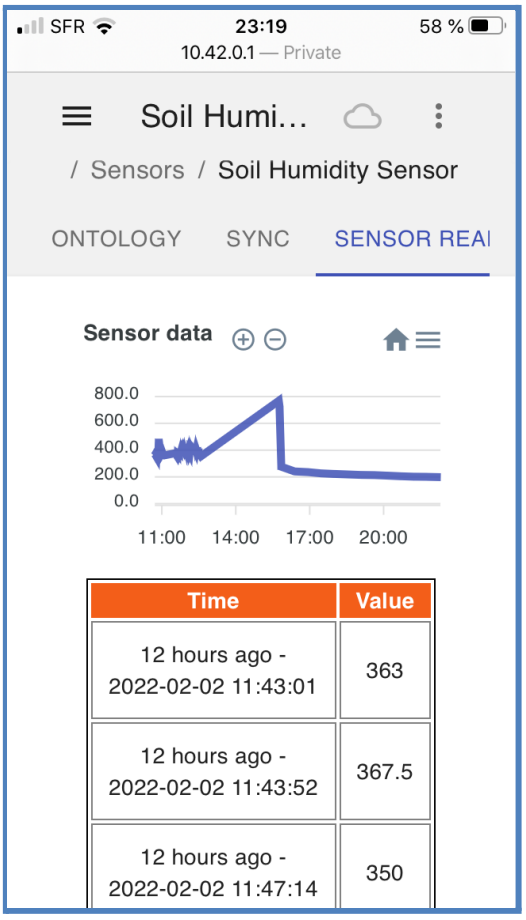
Here is another screenshot showing the early tests with the INTEL-IRRIS Soil Sensor.

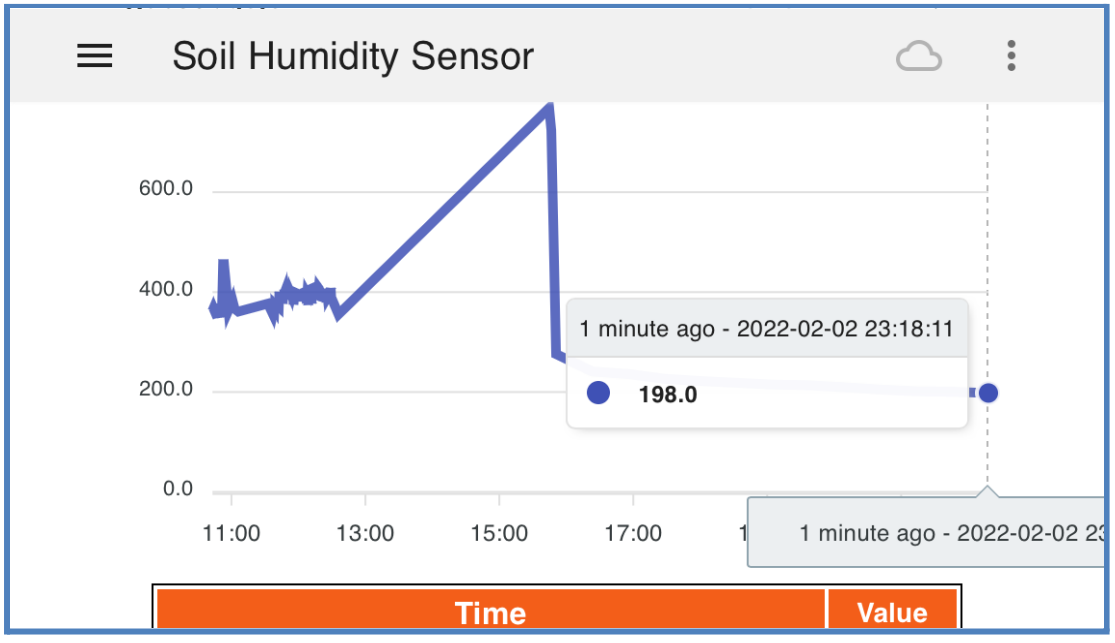




The whole embedded dashboard being responsive can be accessed from a smartphone. The following screenshots show how sensor data can be visualized quickly from a smartphone or tablet.







2.4. Installing WaziGate software

This is a short explanation, it is assumed you already have some experience with Raspberry PI. The following things are needed to work with our software:

- A Raspberry PI 3b or 4
- LoRa Hat (WaziHat, RAK, or HT-M01) with the antenna attached.

If you are unsure, please follow our more detailed step by step guide[8].

Step #1: Download and flash the latest Wazigate ISO image[9].

Step #2: Once the WaziGate is flashed, boot it up and connect to its UI. You can connect to the UI using:

- The WaziGate Hotspot.
- An ethernet cable connected to a local router.
- An HDMI screen (you need to connect it before switching on the RPI).

The WaziGate UI will be available in your browser at http://wazigate.local. Please allow at least 10 min for the first boot, before connecting.

Step #3: Once inside the UI, configure your WiFi if necessary.

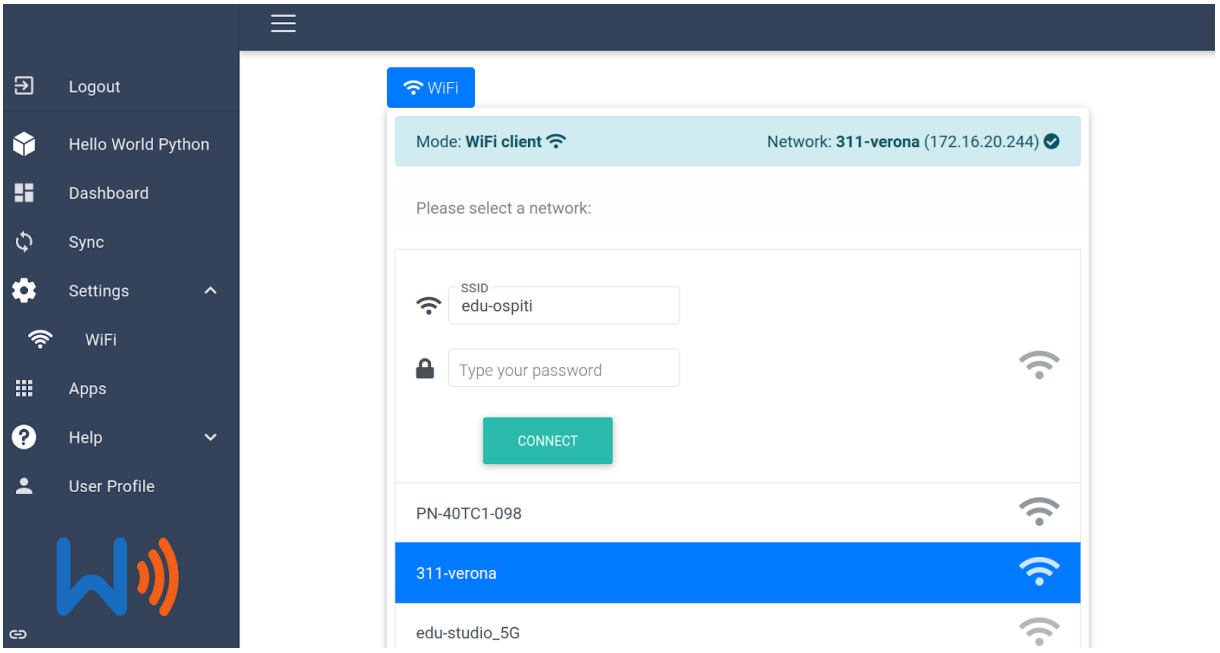


Figure 4 – Managing WiFi connectivity

Step #4: Optionally, you can configure also your Cloud credentials in the Sync panel. You can get your Cloud credentials at http://dashboard.waziup.io[10].

[8] https://www.waziup.io/documentation/wazigate/v2/install/
[9] https://downloads.waziup.io/WaziGate_latest.zip
[10] http://dashboard.waziup.io/

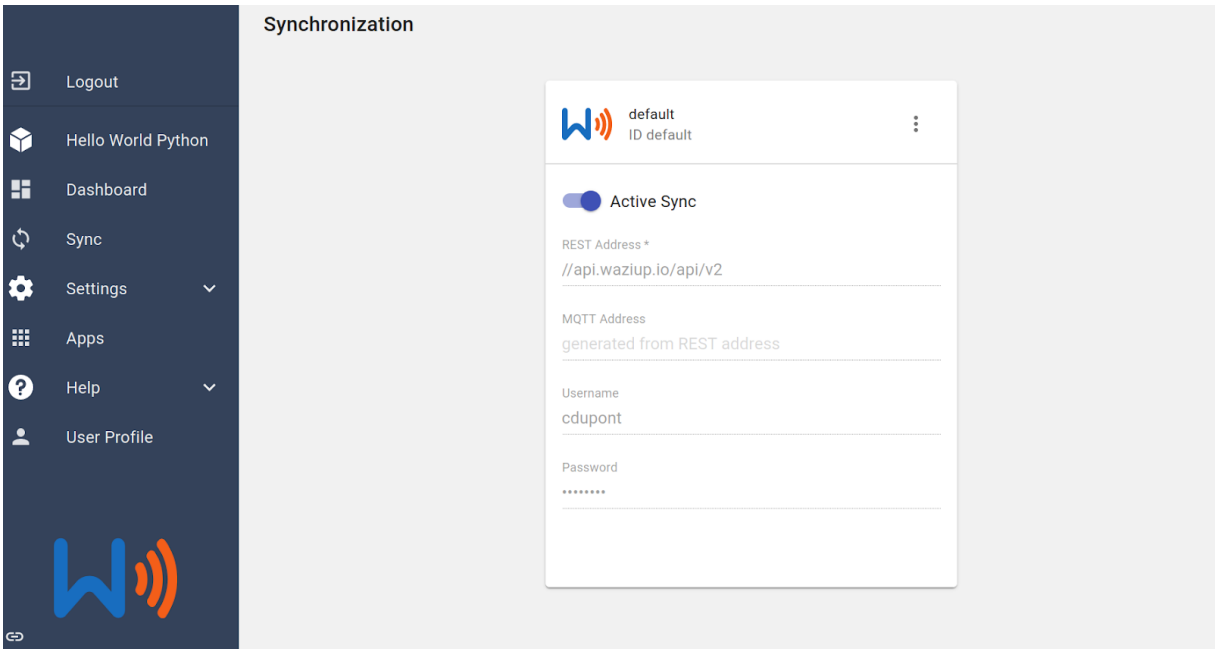


Figure 4 – Optional active sync with the cloud

After the synchronisation is active, the sensor or actuator data is also available via the WaziCloud. This enables a comprehensive overview about different gateways and their attached hardware. This comes in handy, when there are more than one Raspberry PIs active, all information is available at a glance.

2.5. Documentation & tutorial materials

There are detailed documentations and courses on various WAZIUP's IoT components available on WAZIUP website[11].

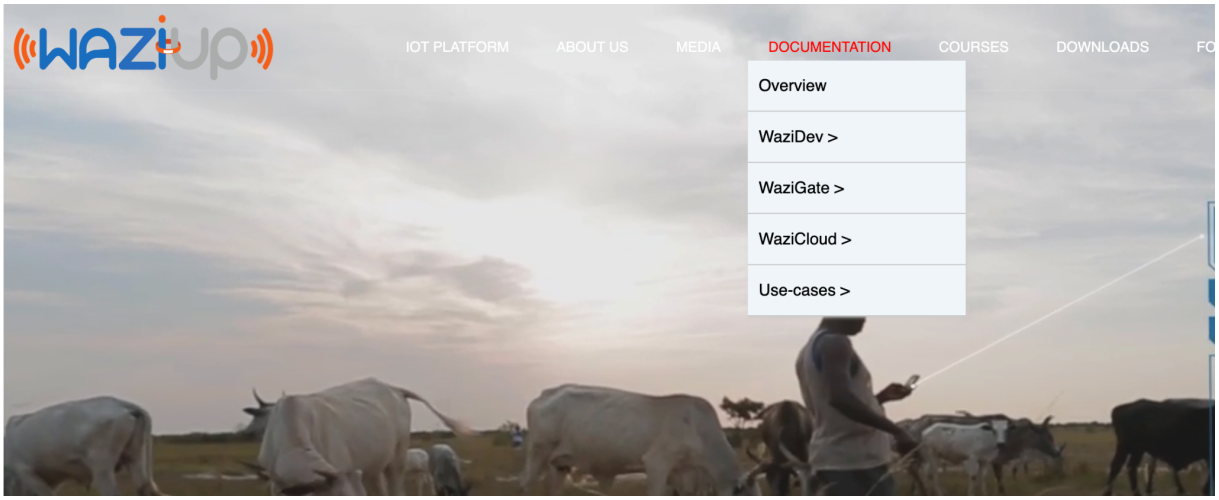


Figure 5 – Documentions on https://www.waziup.io

[11] https://www.waziup.io

Figure 6 –Courses on https://www.waziup.io

3. A PROOF-OF-CONCEPT SMART-IRRIGATION WAZIAPP

3.1. Presentation of the smart-irrigation WaziApp

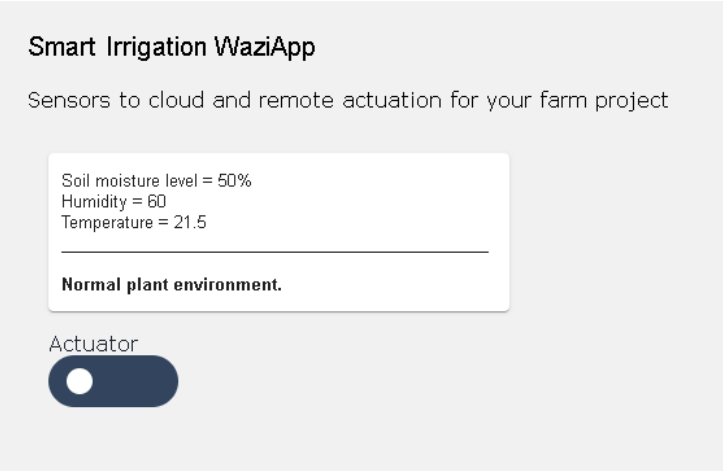
The WaziApp for Smart Irrigation is a proof-of-concept application developed by WAZIUP. It runs locally on the WaziGate. The goal is to prototype an IoT based system that can monitor the moisture content of a farm's soil, current temperature and humidity readings; and also, to be able to control supply of water to a crop. The WaziApp thus provides a local version of IoT Cloud services thus reducing the cost for internet requirements.

The application has been developed with Python, HTML, JavaScript and containerized with Docker. This architecture allows it to be built for a specific sensor device using its `device_id` and `actuator_id`. This enables visualization of the different sensor data of that device, together with an actuation mode. Once the application gets the sensors' data, it analyzes them and displays relevant actions that the user can take.

With this, other features can be added such as visualization with graphs, charts, and analytics using Machine Learning. Developed using Python too.

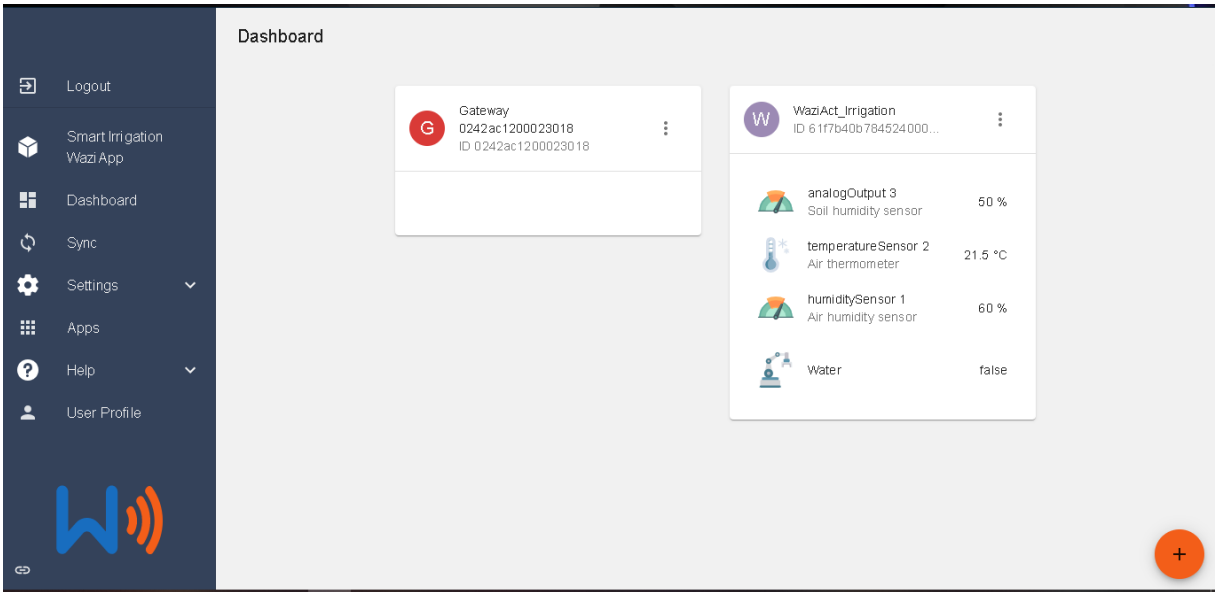
3.2. General description & main features

The irrigation WaziApp allows visualization of a device's sensor and also setting its actuator values.



The application's architecture enables the user to select a particular device that they want to link the WaziApp with, using the device id.

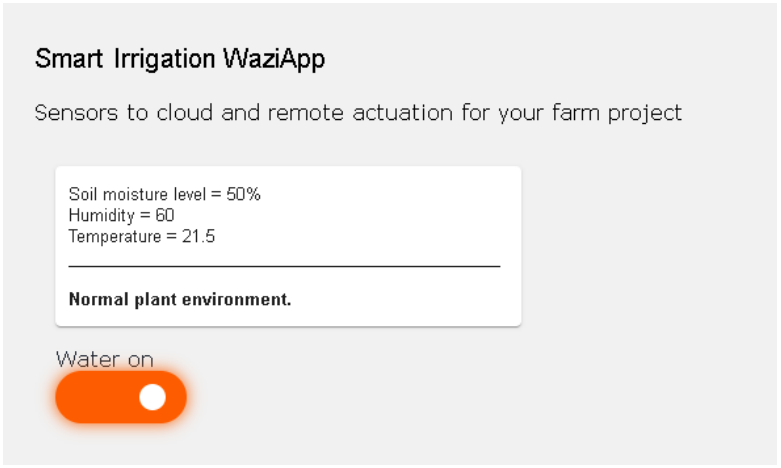
Let's consider a WaziApp for the Device: WaziAct_Irrigation:

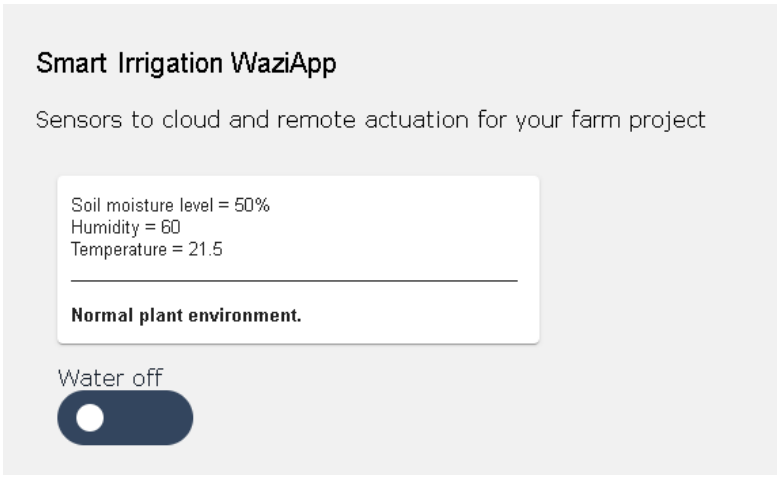


The WaziApp first gets the data: Soil humidity sensor, Air temperature and Air humidity sensor. An advantage of having an application is that the sensor data can be analyzed and we can get relevant information from them. Henceforth, the WaziApp informs the user to take various actions based on the insights. The provided insights and actions are:

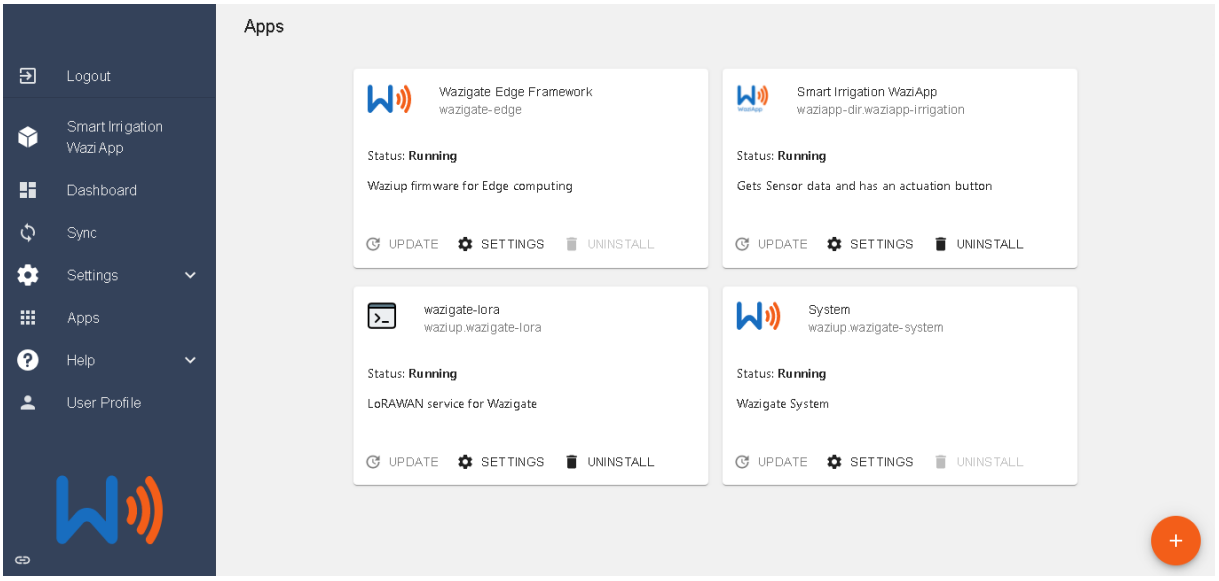
a) Watering required. Moisture content is low!
b) Watering is required very soon.
c) Normal plant environment.
d) Good plant environment.

With this information, a user can then take the required actions such as watering the plant if informed to:



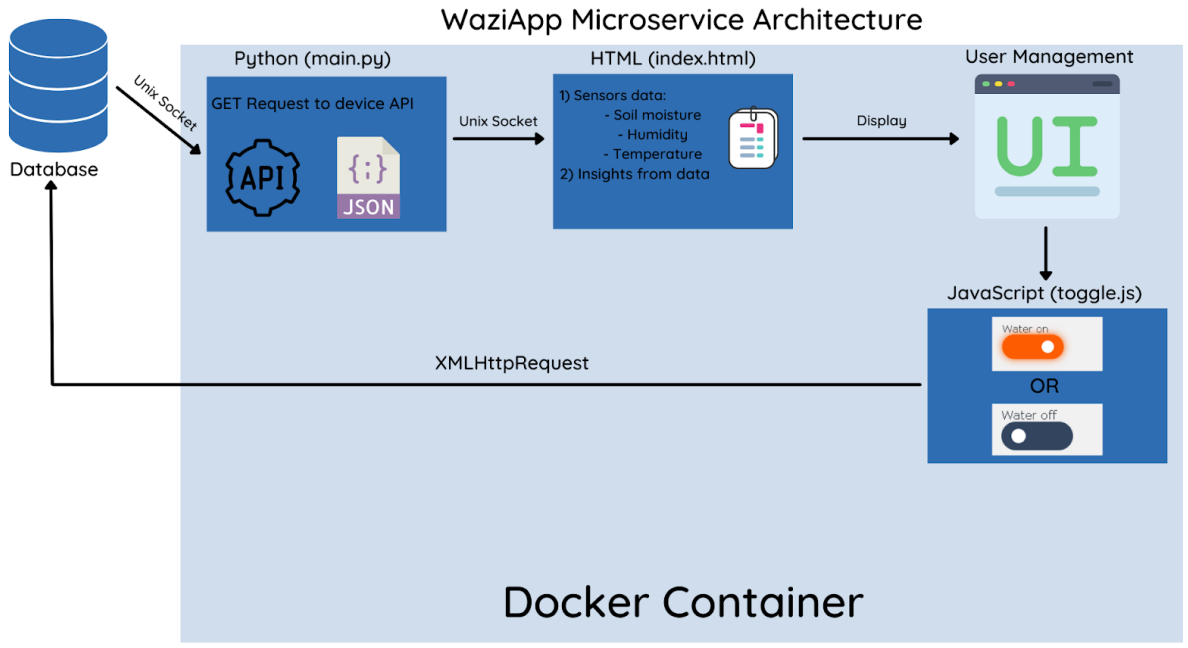


Once installed on the WaziGate, the WaziApp's status can also be seen from the Apps menu with other installed applications.



3.3. Architecture

The WaziApp is built as a Python microservice with Docker. It primarily includes a Python program, HTML/CSS for user-interface, and a JavaScript that handles setting actuator states. The other files are for Docker and Unix socket configuration.



1) The Python program has a function that handles GET requests for obtaining the sensors data from the device's local API using a Unix socket.
2) This function makes a request to the device's API and obtains data in JSON format.

A device's API needs to have sensors data of : soil moisture, humidity and temperature; in the listed order. Also, an actuator is needed to have been created from the dashboard. This requirement is so that the Python's function can correctly obtain the 3 sensors data using their known positions in the JSON.

3) Once the sensors data is obtained, the Python program then analyses the data and gives the described insights.
4) This Python function is then used by the HTML program to obtain the sensor data together with the insights and print them on a Web browser.
5) To set the actuator modes, the HTML page and a JavaScript program are used. The HTML dashboard has a toggle switch that triggers different events in the JavaScript program. These events use XMLHttpRequest to POST different actuator states: true or false, to a device's local API. This eliminates the need of a form or refreshing the page when the actuator switch is toggled.

4. The INTEL-IRRIS WaziApp

4.1. General description

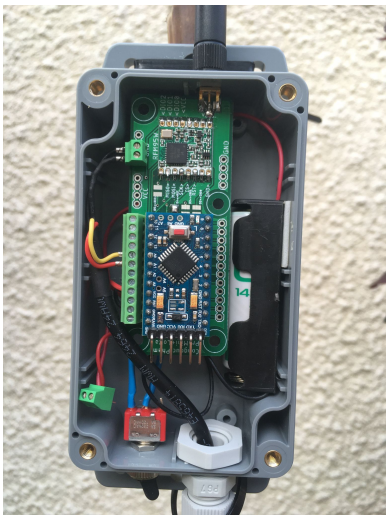
The INTEL-IRRIS WaziApp (IIWA) will be based on the current smart-irrigation WaziApp architecture and approach.

IIWA has additional requirements that will be listed in the following paragraphs.

4.2. Requirements & Specifications

IIWA focuses on processing a soil parameter from a single sensor. This soil sensor could be either a simple capacitive sensor (SEN0308 is the main target for INTEL-IRRIS, left figure) or a tensiometer (WATERMARK 200SS is the main target for INTEL-IRRIS, right figure).

The INTEL-IRRIS soil sensor (IISS) will then be a soil sensor device as illustrated below.



A list of requirements and specifications are shown below:

A/ If there is another soil sensor deployed, it is better to have another instance of IIWA to link with the data of this other soil sensor.

B/ For each soil sensor, it should be possible to specify a set of parameters such as:

- sensor type (capacitive, tensiometer)
- plant/crop type
- soil salinity
- soil bulk density
- soil type (arid, semi-arid,...)
- …

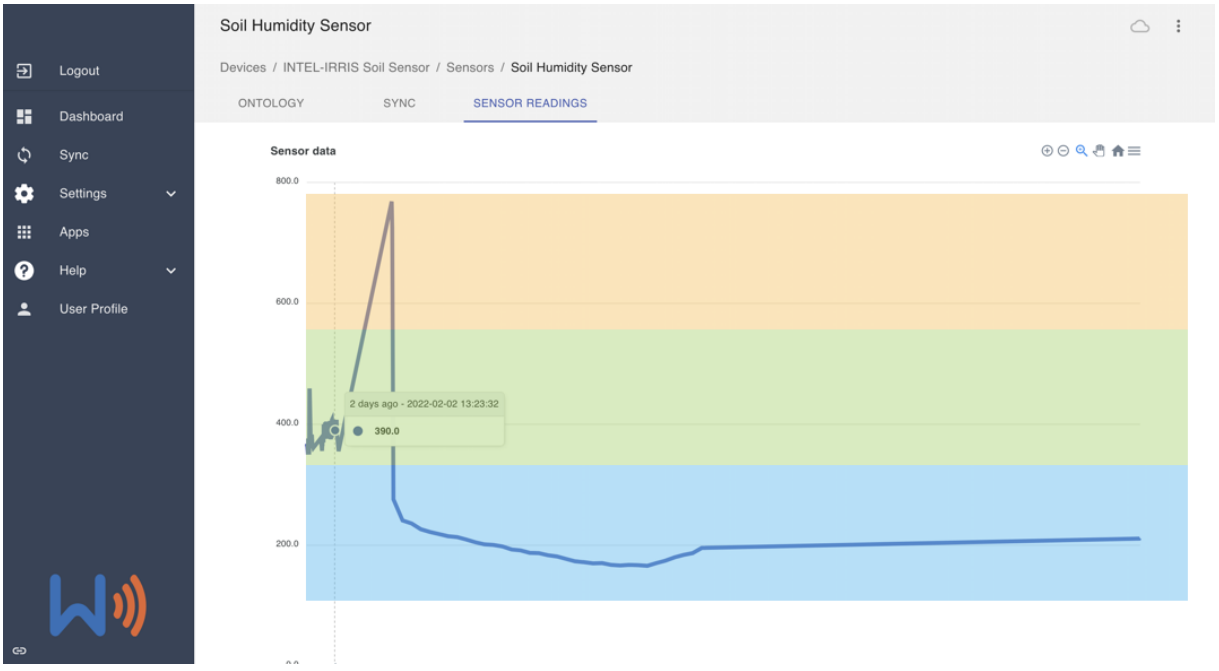
C/ There should also be the same set of parameters defined at a global scope. Then, for a given soil sensor, the user should be able to indicate for each parameter whether it is the default value, a specific value or whether it is disabled.

D/ The structure of a configuration file needs to be defined. For instance, it could be a global .json file with a global section and a dedicated section for each sensor.

E/ To display sensor value, there should be the choice of several graph type:

- line plot when a range of values is required
- gauge when a single value is required

F/ It should be possible to define intervals with color code so that sensor value could be visualized with these colors. See an illustration below: orange (dry), green (ok), blue (too wet)



G/ Intervals can be defined by taking the maximum value and the minimum value received from the sensor, then the interval range will be computed and associated with the colors.

I/ IIWA should also be able to drive a small OLED screen. It has already been tested that the OLED screen can be programmed with Python code on the WaziGate, as illustrated below.

4.3. Data structures and files

In addition to the generic sensor data database provided by the generic IoT gateway framework, the INTEL-IRRIS WaziApp will need:

- a configuration file to store the various parameters for the complex water-soil-plant interaction model
  - json format: `intel-irris-conf.json`
- an irrigation information status file to store relevant information and irrigation recommendations that can be displayed to the end-user
  - json format: `intel-irris-output.json`

5. EMBEDDED AI FRAMEWORK

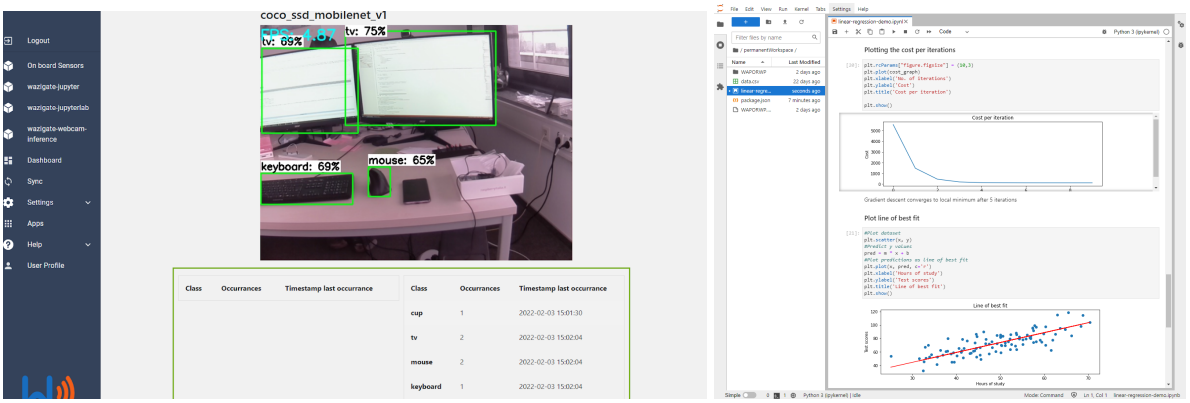
The AI Framework is a new component of the WaziGate's ecosystem, at the moment WAZIUP is working on extending its functionality for INTEL-IRRIS project. The Embedded AI framework will include the following features and funtionalites.

- The AI edge-processing framework should be applicable for for smart irrigation application
- The framework will also able to test and validate various AI models
- The framework will also include both online and offline AI training capabilitiy

Currently, the framework has three major components.

The first component consists in an environment for developers/users to conduct artificial intelligence and machine learning tasks. For that reason we included a dedicated system-oriented WaziApp with the full Jupyterlab functionality. Different machine learning packages can be installed that help the developer with various data science tasks such as regression models for instance. It is possible to use different programming languages, for example Python is preinstalled, and the functionality can be enhanced with different kernels which are available. There are numerous pre-cooked notebook files available, for a variety of applications, to choose from. You can query the WaziGate's data of different sensors and actuators, from inside the WaziApp, to conduct different tasks on it.

Then, to demonstrate the embedded AI features, a computer vision WaziApp was created as an example to dynamically introduce this functionality into the Gateway. With help of a RaspiCam or another camera, attached to the WaziGate via I2C bus, USB or IP, the WaziGate is able to carry out the inference on a videostream. The WaziApp has different models, which have differences in mean average precision and runtime. The models are trained with the COCO dataset, it includes 80 different classes, so the network can distinguish between 80 different objects, for example: persons, cars, animals or even a toothbrush!



Another feature we want to add to our environment of applications is a component to judge on future values of sensors and actuators. This dedicated WaziApp will contain algorithms to compare different models, to find the most suitable one for the given task. In a first step, input data from the WaziGate or from other sources must be available for model training. Then the

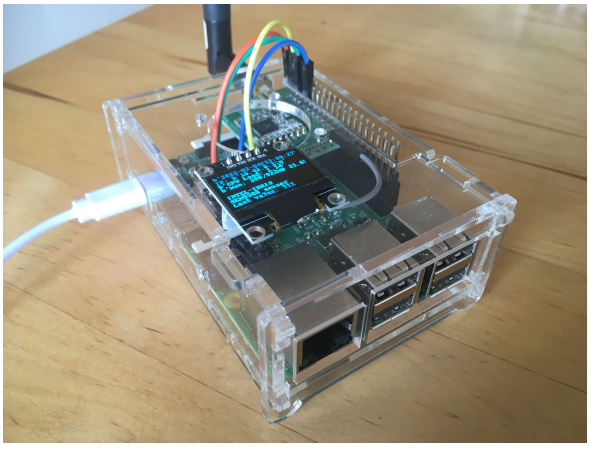
WaziApp will use these training data with different learning methods and models to build a variety of trained models with different parameters. Afterwards, tests can be conducted on a test dataset, split from actual data. Finally, the objective is to allow users to choose a specific model according to the metric that is most important for the final application. This WaziApp is in an early stage of development. The future development of the AI framework will be included in the next deliverable D2.1b.

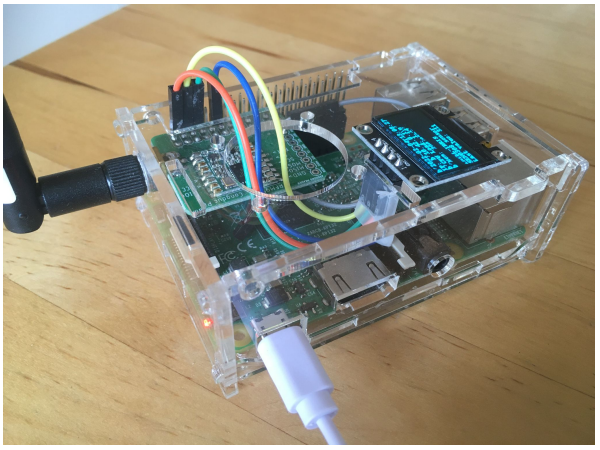
6. PACKAGING THE INTEL-IRRIS GATEWAY

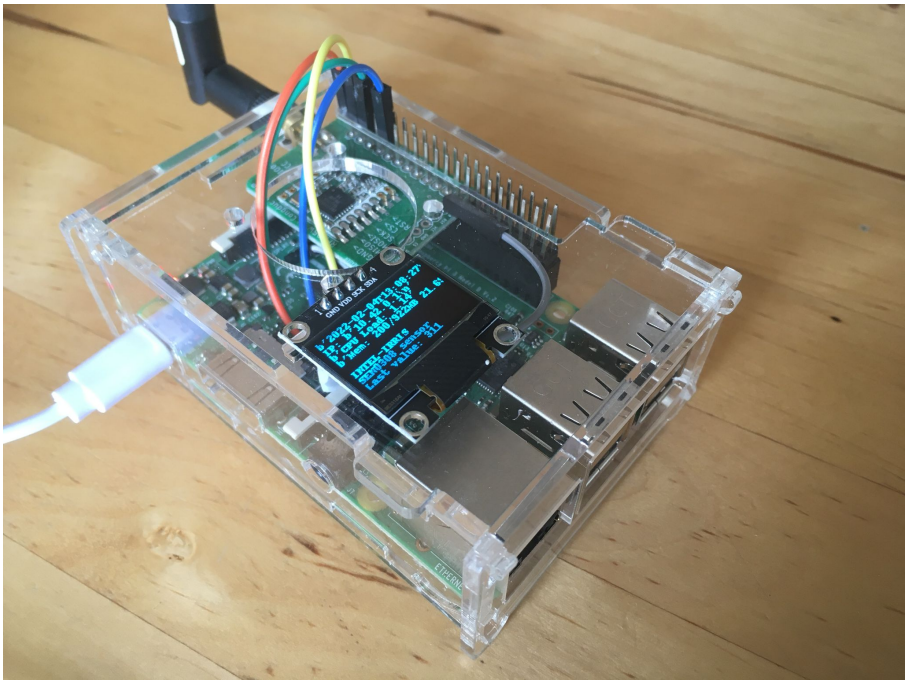
The INTEL-IRRIS IoT gateway builds on our previous expertise as described in the WaziGate section. The core hardware platform will be the RaspberryPi 3B+ or RaspberryPi 4 depending on memory requirements. The first version will be based on the less expensive RPI3B+ that can still be able to support 64-bit OS.

The RPI3B+ will come with a single-channel LoRa radio hat where a small OLED screen can be connected.

The whole system will be packaged in a low-cost plastic enclosure as the IoT gateway is mainly targeting an indoor deployment scenario performed by smallholders themselves. The pictures below illustrate one of the enclosures that could be used. The objective being to be as simple as possible.







REFERENCES

Acronyms List

Acronym	Explanation
MQTT	Message Queue Telemetry Transport
OS	Operating System

Project Co-ordinator Contact

Pr. Congduc Pham

University of Pau

Avenue de l'Université

64000 PAU

FRANCE

Email: Congduc.Pham@univ-pau.fr

ACKNOWLEDGEMENT

This document has been produced in the context of the PRIMA INTEL-IRRIS project. The INTEL-IRRIS project consortium would like to acknowledge that the research leading to these results has received funding from the European Union through the PRIMA program.