D6.4

Final version of Pilot Use Case Implementation

WP6 – Smart Home Pilot Use Case

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

Due date of deliverable: 30/09/2023
Actual submission date: 30/09/2023

Responsible partner: DOMO
Editor: Domenico De Guglielmo;
E-mail address: domenico.deguglielmo@domo-iot.com

30/09/2023
Version 1.0

Project co-funded by the European Commission within the Horizon 2020 Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

The SIFIS-Home Project is supported by funding under the Horizon 2020 Framework Program of the European Commission SU-ICT-02-2020 GA 952652

Authors: Domenico De Guglielmo (DOMO)
Approved by: Joni Jamsa (CEN), Marco Rasori (CNR)

Revision History

Version	Date	Name	Partners	Section Affected Comments
0.1	1/07/2023	Defined ToC	DOMO	All
0.2	15/07/2023	Updates to D6.2 and D6.3	DOMO	All
0.3	20/08/2023	Use Case Description	DOMO, CNR	Section 6
0.9	11/09/2023	Ready for review	DOMO	All
1.0	29/09/2023	Ready for submission	DOMO, CNR	All

Executive Summary

This deliverable reports the details of the final implementation of the SIFIS-Home pilot architecture, which also represents the real-testbed for validation of SIFIS-Home mechanisms.
The deliverable is structured as follows. First, we report the details of the various devices that we used in the pilot. Second, we describe the network and system architecture of our pilot, showing how all the different devices involved in the pilot are interconnected. Third, we introduce the various software components that have been integrated and tested in the SIFIS-Home pilot implementation. Finally, we report the different smart home use cases that have been validated by using our pilot implementation and, for each one of them, we report the results of the GQM-based validation as well as the results of the usability tests that we defined.

Table of contents

Contents

1 Introduction

The smart home pilot is the real testbed used to show the possibility of integration of SIFIS-Home in existing architectures and devices, presenting an architecture, which fully matches with the SIFIS-Home paradigms. In particular, the architecture involves real devices classified in Smart Devices (SDs) and Not So Smart Devices (NSSDs), fully distribution of functionalities among decentralized smart devices to improve reliability and resilience, secure communication, and privacy-aware data management.
This deliverable reports the details of the final SIFIS-Home pilot implementation. It is structured as follows. First, we describe the different smart and not-so-smart devices that are used in the pilot. For each one of them we report its hardware components and describe its specific use in the pilot. Also, we show the specific actions that need to be performed to install the SIFIS-Home software components on the pilot devices. We continue by describing the details of the network architecture of our pilot to show how the devices communicate and are interconnected. The various components that are installed and executed on the various devices are then described. We then explain how the devices and the software components interact by describing a set of different smart home use cases that we validated using our pilot implementation. We conclude the deliverable reporting both the use cases validation results and the usability assessment results.

2 Devices used in the pilot

This section describes the different smart (SD) and not-so-smart (NSSD) devices that are used in the pilot implementation. The main hardware characteristics of the various devices are reported, and their specific use in the pilot is highlighted.

2.1 *Smart Devices*

Smart devices are powerful devices where it is possible to install a number of applications. They execute the set of SIFIS-Home software components that compose the SIFIS-Home Smart Device framework. In the following, we describe the smart devices that have been used in the pilot.

2.1.1 DoMO gateway

Figure 1 shows the DoMO gateway, i.e., the main smart device used in our pilot. The DoMO gateway is a quite powerful device, based on the Banana PI R3 board, which is provided with a Quad Core ARM A53 CPU and 2 GB of DDR RAM. Also, it has 8GB of EMMC flash available and is equipped with a 500 GB NVMEe disk. Regarding network connectivity, the DoMO gateway is equipped with two 4x4 WiFi 6 network chips (2.4Ghz and 5Ghz bands), 5 Gb Ethernet ports and two 2.5 Gb SFP ports. Also, it is provided with a user-accessible USB 3.0 compliant port that allows connecting external USB devices. Additional details of the device are reported in Figure 2.

The DoMO gateways run an OpenWrt Linux distribution. In detail, we prepared a specific OpenWrt distribution for the DoMO gateways that includes the various components that are needed to execute the SIFIS-Home software components to be used. For additional details please refer to section DoMO Gateway OpenWrt distribution.

The DoMO gateway not only runs the vital SIFIS-Home services but also provides WiFi connectivity to the NSSD that are installed in a SIFIS-Home house.

Figure 1: Domo Gateway

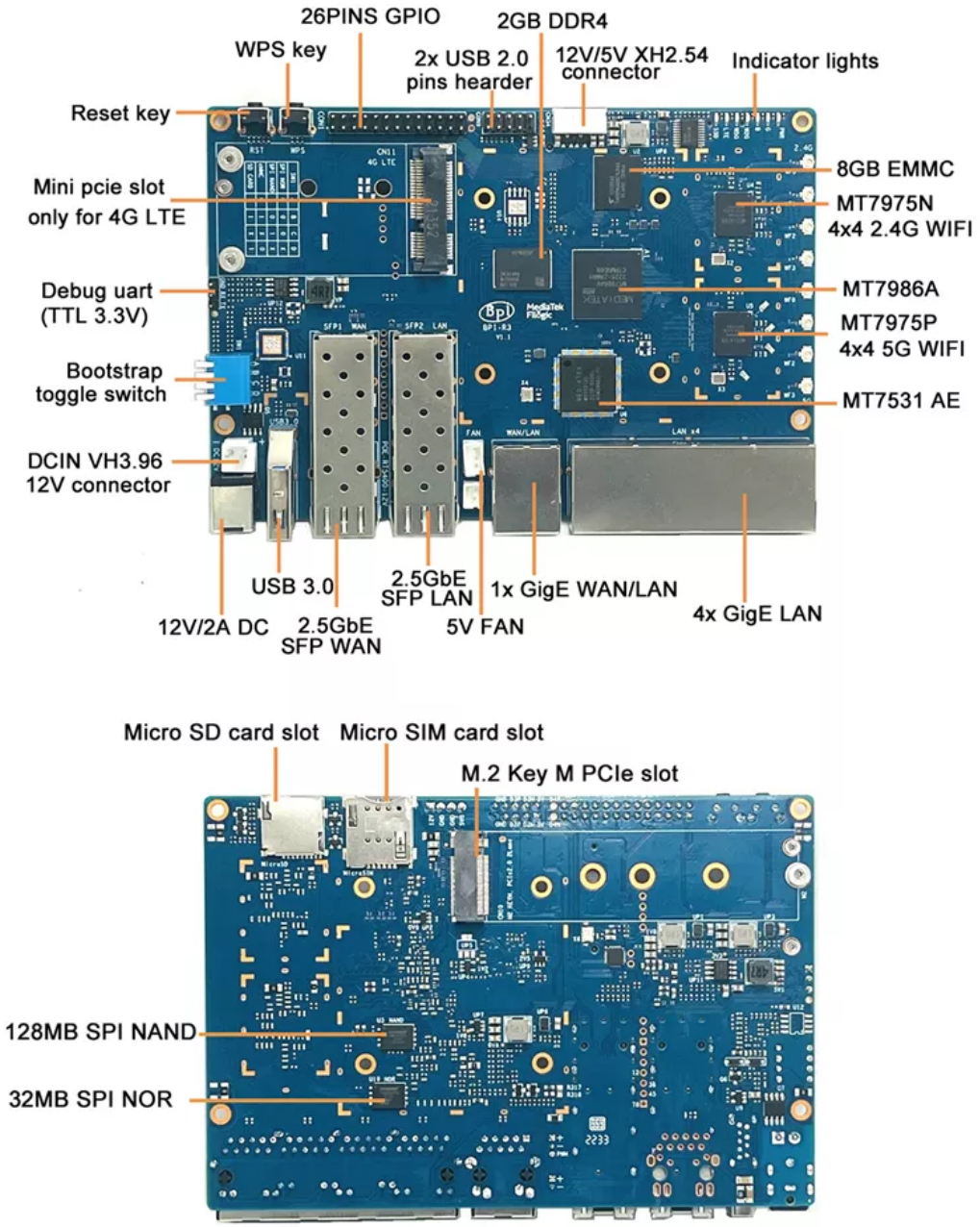


Figure 2: Domo Gateway details

2.1.2 Laptop

In the pilot we use standard laptops as amd64 Smart Devices. In detail, Figure 3 shows the model of the laptop that we used in our pilot implementation. It is a quite powerful device, provided with an Intel i7 CPU, 32GB of RAM and a dedicated Nvidia GPU. Also, it is equipped with a camera and a microphone. We used it to validate the use cases that make use of voice commands and that rely on computer vision tasks.

Please refer to section _Laptop_distribution for additional details of the Linux distribution that we used on our laptops.

Figure 3: Testbed Laptop

2.2 *Not So Smart Devices*

Not so smart devices are small, constrained devices that are mainly used to interact with the physical world. The set of SIFIS-Home software components that they execute is named SIFIS-Home NSSD framework. We report the details of the NSSDs used in the pilot in the following section.

2.2.1 DoMO WiFi actuators

The SIFIS-Home pilot uses different types of WiFi actuators, provided by DoMO, to control and monitor the energy consumption of the lights, sockets, shutters, and appliances installed inside the house. The WiFi actuators are simple devices that provide output and input channels and allow to turn on and off the appliances/light/sockets they are attached to, as well as measure and report their energy consumption. By using the input channels of the actuators, it is also possible to detect the state of attached buttons and bistable buttons, as well as the state of attached window and door contact sensors. All the actuators are equipped with an Espressif chip that provides WiFi/Bluetooth connectivity and that can be flashed with a custom firmware. The NSSDs used in the SIFIS-Home pilot should expose a WebOfThings (WoT) compliant API in order to be controlled and monitored. To this end, we developed a WoT compliant firmware whose details are described in section _DoMO_WiFi_actuators.

In the following, we briefly describe the characteristics of the various types of WiFi actuators that are used in the pilot.

Shelly 1

Figure 4 shows the Shelly 1 WiFi actuator. It provides one input channel and a potential-free output channel. It is not provided with an energy monitoring chip. It can be used to turn on and off lights and appliances as well as heating systems. Also, it can detect state changes of buttons/contacts to which its input channel is connected to.

Figure 4: Shelly 1

Shelly 1PM

Figure 5 shows the Shelly 1PM WiFi actuator. It provides one input channel and one output channel. It can be used to turn on and off lights and appliances and monitor their energy consumption. Also, it can detect state changes of buttons/contacts to which its input channel is connected to.

Figure 5: Shelly 1PM

Shelly 2.5

Figure 6 shows the Shelly 2.5 WiFi actuator. It provides two input channels and two output channels. It can be used to turn on and off light and appliances and monitor their energy consumption. Also, it allows to open/close shutters and curtains. Finally, it can detect changes in the state of buttons/contacts to which its input channels are attached to.

Figure 6: Shelly 2.5

Shelly Dimmer

Figure 7 shows the Shelly Dimmer WiFi actuator. It provides two input channels and one output channel. It can be used to control dimmable lights and monitor their energy consumption. Also, it can detect changes in the state of buttons/contacts to which its input channels are attached to.

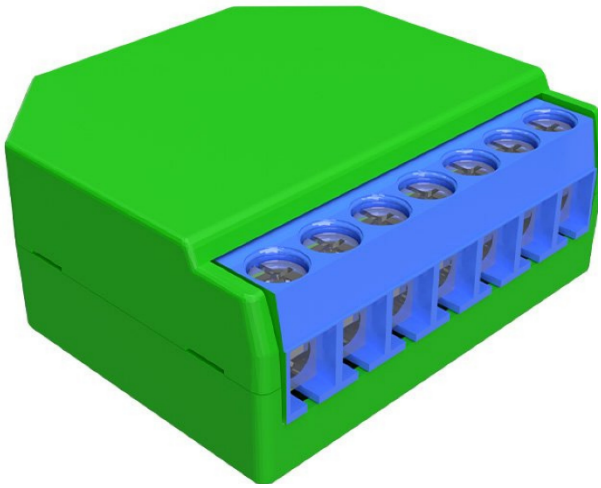


Figure 7: Shelly Dimmer

Shelly RGBW

Figure 8 shows the Shelly RGBW WiFi actuator. It provides one input channel and a number of output channels that can be used to control RGBW led lights. Also, it can detect changes in the state of buttons/contacts to which its input channel is attached to.

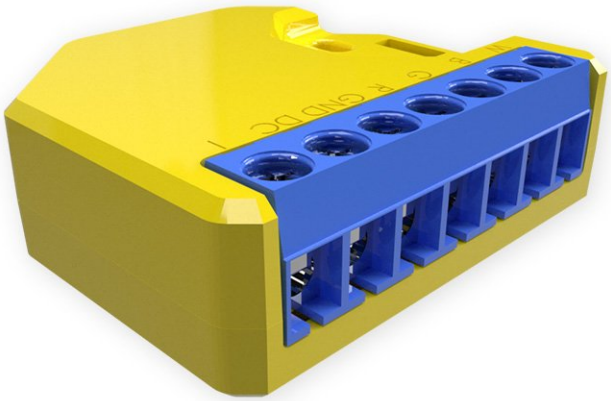


Figure 8: Shelly RGBW

Shelly EM

Figure 9 shows the Shelly EM WiFi actuator. It is a device that can provide the total power and energy consumption of the house where it is installed.

Figure 9: Shelly EM

Shelly 1 Plus

Figure 10 shows the Shelly 1 Plus WiFi and Bluetooth actuator. It provides one input channel and a potential-free output channel. It is not provided with an energy monitoring chip. It can be used to turn on and off lights and appliances as well as heating systems. Also, it can detect state changes of buttons/contacts to which its input channel is connected to. In addition, being equipped with a Bluetooth chip it can be used to control Bluetooth-based actuators as well as to receive data from Bluetooth sensors in its proximity. In the pilot implementation we use this device to detect state changes of Bluetooth Low Energy Contact Sensors (that are described in the next section).

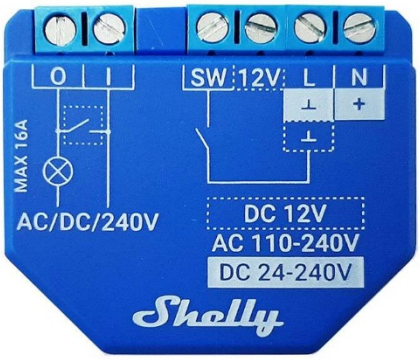


Figure 10: Shelly 1 Plus

Bluetooth Low Energy Contact Sensor

Figure 11 shows the Bluetooth Low Energy (BLE) Door/Window contact sensors that we used in the pilot. They are battery-operated contact sensors that allow to detect when a door/window is open/closed. In detail, whenever an open/close event is detected by the sensor, the on-board Bluetooth chip is used to generate a specific BLE advertisement packet reporting the event. In the pilot implementation, we use the Shelly 1 Plus actuators to receive the BLE advertisement packets generated by the contact sensor and report the door opening/closing event to the SIFIS-Home system.

Figure 11: Bluetooth Low Energy Door/Window contact sensors

Bluetooth Low Energy Temperature and Humidity Sensor

Figure 12 shows the Bluetooth Low Energy Temperature and Humidity sensors that we used in the pilot. They are battery-operated sensors that allow to receive temperature and humidity measurements. In detail, the on-board Bluetooth chip is used to generate periodic BLE advertisement packets that report both the temperature and the humidity values of the room where the sensor is placed. In the pilot implementation we use the Shelly 1 Plus actuators to receive the BLE advertisement packets generated by the sensor and report the measurements to the SIFIS-Home system.

Figure 12: Bluetooth Low Energy Temperature and Humidity Sensor

2.2.2 Riots Devices

Riots Thermostat

Riots Thermostat shown in Figure 13 is a device that measures temperature and humidity and provides means to control room temperature. Thermostat has touch buttons and display that are used to set the desired room temperature. The history of collected sensor data and current settings can be monitored and controlled in realtime using the Riots Cloud on any browser or Riots Mobile app on a smartphone. Riots Thermostat provides proprietary wireless connection that is used to communicate with it inside an apartment. Other Riots devices can connect to it wirelessly enabling freedom regarding the device positioning inside the Apartment if they stay within the range of the wireless connection.

Figure 13: Riots Thermostats

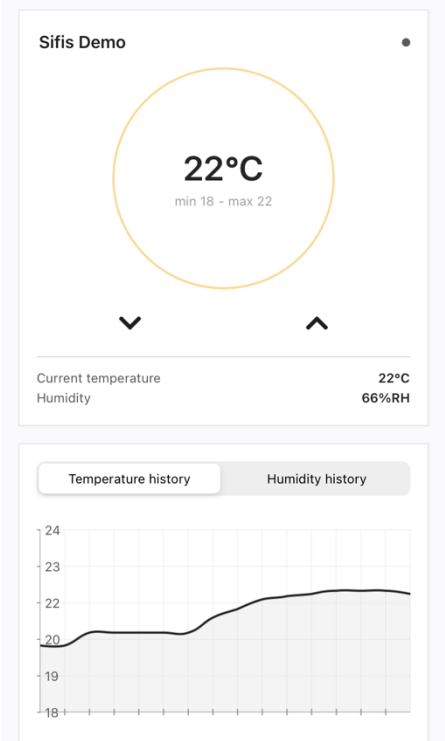


Figure 14: View from RIOTS mobile app

Riots Mama

Riots Mama shown in Figure x is a device that connects the Riots devices to Riots Cloud. Mama creates an AES128 end-to-end encrypted connection between the local Riots network and the Riots Cloud service. All the network traffic goes through Mama, which links a building and its spaces to the Internet. It is a part of every Riots solution, because it enables the spaces and buildings to be turned smart, creating a secure and efficient data transfer from cloud service to the devices in buildings.

Figure 15: Different variations of Riots Mama – GSM (left), LAN (center), USB (right)

Riots Mama has minimal amount of implementation logic included; it is essentially a gateway that converts Riots proprietary wireless connection to the encrypted TCP connection. There are few different variants of Riots Mama device depending on the physical connection type:

- GSM Mama connects to the Internet using IoT SIM card and GPRS data connection
- LAN Mama connects to the Internet via RJ45 port and Ethernet connection
- USB Mama connects to a smart device via integrated USB port.

It provides serial communication and uses host device's internet connection
In this pilot USB Mama is used and the host device's software that is used to communicate with Mama and Riots Cloud is updated to add Sifis-Home compatibility. For more details see chapter *Riots WoT integration* later in this document.

3 Pilot architecture

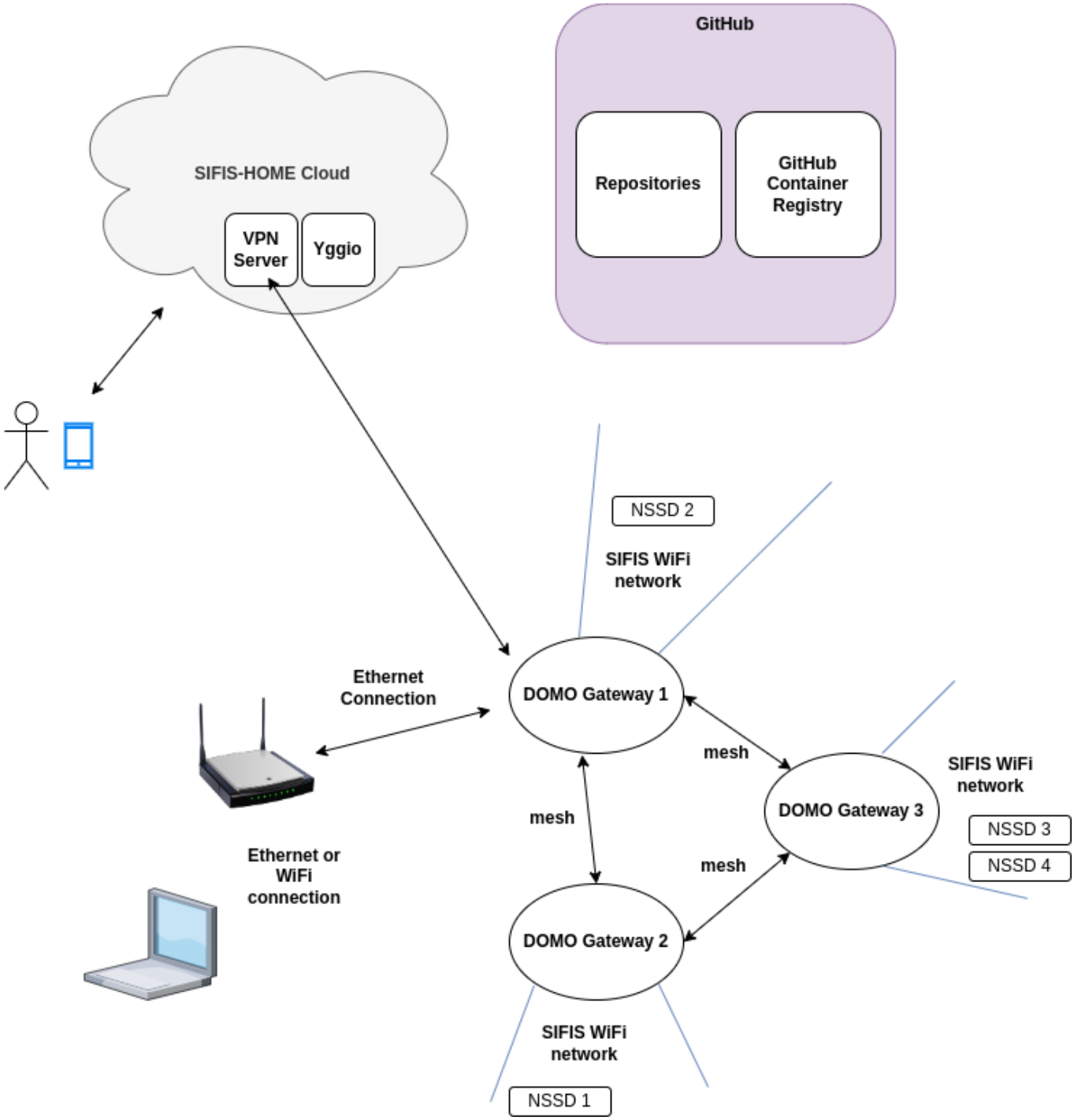


Figure 16: Pilot network architecture

Figure 16 shows the current network/system architecture of the SIFIS-Home pilot. As it can be observed a number of devices and components are present in the pilot:

Smart Home Router: it is the device used to provide Internet Connectivity to the house. It is in general provided by the Internet Service Provider (ISP), and it does not execute SIFIS-Home software components. We assume that it provides WiFi connectivity to the users of the smart home and executes a DHCP server that assigns IP addresses to the network devices deployed in the house.

DoMO gateway: a number of DoMO gateways are present in the house. At least one of them is connected to the Smart Home Router using an Ethernet connection. The DoMO gateways are connected with each other by means of a dedicated WiFi mesh network. In addition, they advertise a specific WiFi network (SIFIS WiFi network in Figure 16) to which the NSSDs used in the pilot connect to.

Laptop: our pilot also includes laptops. They are connected to the local area network created by the Smart Home Router and receive their network configuration from the DHCP server that it executes.

NSSD: there are several NSSDs inside the house. They are configured to connect to the network advertised by the different DoMO gateways present in the house. Once connected to the WiFi network of the DoMO gateway, they expose a WoT compliant API. Please note that the Bluetooth contact sensors are not directly connected to the main SIFIS-Home network, but, rather, they are connected to the Shelly 1 Plus devices, which act as Bluetooth/WiFi gateways.

User smartphones: they are the devices that can be used by the users of the smart home to control the lights and appliances connected to the NSSD deployed in the house. They run a mobile application that executes the set of components being part of the SIFIS-Home Application Framework.

SIFIS-Home Cloud: we used a physical server, named Panarea, residing at CNR facilities, to host the services being part of the SIFIS-Home Cloud framework. In detail, in the pilot implementation, we run both Yggio and a VPN server on Panarea, to allow the smart home users to access the house functionalities from a remote side.

GitHub: we use GitHub to host the source code of the different SIFIS-Home software components that we developed. Also, we make use of GitHub actions to build the different software components. Finally, we rely on the GitHub Container Registry (GHCR) service to host the Docker images of the different SIFIS-Home services.

4 Smart Devices OS distribution setup

In this section we describe the details and the setup of the specific operating systems/distributions that are used on our "operating systems/distributions used on our Laptops and DoMO gatewaysLaptop distribution
The laptops that are used in the pilot run an Ubuntu 22.04 Linux distribution where the *docker* and *docker-compose* packages have been installed using the *apt* package manager. After the installation of such packages, it is possible to execute the different SIFIS-Home components by launching their associated Docker images.

4.1 *DoMO GW OpenWRT distribution*

The DoMO Gateways run a SIFIS-Home OpenWrt Linux distribution, i.e., an OpenWrt distribution that contains all the software packages that are needed to execute the components of the SIFIS-Home Smart Device Framework. In detail, the SIFIS-Home OpenWrt distribution that we prepared contains the *docker* and *docker-compose* packages as well as the *domo-bootstrap* package (for additional details of the procedure that we used to prepare the DoMO Gateway OpenWrt image please refer to DoMO GW OpenWrt distribution). *docker* and *docker-compose* packages allow the DoMO Gateways to execute Docker containers by using the command line or the docker-compose binary (i.e., by specifying the docker containers to run in a Docker Compose yaml file). Conversely, the domo-bootstrap package provides the SIFIS-Home OpenWrt distribution with the *domo-bootstrap* application whose functionalities are described in section domo-bootstrap.

4.2 *Service configuration files*

On both the Laptop and the DoMO gateway distribution, folder /etc/domo is used to store a number of *toml* files that contain the configuration parameters of the different SIFIS-Home services. For example, Table 1 shows the toml file used to configure the DHT Manager component.

Table 1: Configuration file for the DHT manager /etc/domo/broker.toml

```
[broker]
http_port = 3000

[broker.cache]
url = "sqlite://root/domo-dht-db/sifis-db.sqlite"
table = "domo_data"
persistent = true
shared_key =
"d31c75055088a46242198d401a61e8c0db8aa580640e11be5258fc5b18e3456d"
loopback = false
```

In addition, */opt/domo/domo-compose* is the folder where we place the *docker-compose.yaml* file containing the details of all the different SIFIS-Home services to be run on a certain smart device. Figure 17 shows parts of the *docker-compose.yaml* file that is used on the DoMO gateways.

```
version: "3"

services:
  dht-analytic:
    image: ghcr.io/sifis-home/dht-analytic:latest@sha256:76e25a31f7594acf92f53f11ce98017eddeece795df848a4a359343eb685a287

    network_mode: host
  system-protection-manager:
    image: ghcr.io/sifis-home/system-protection-manager:latest@sha256:d2d1ff286fc25984fe7ea36c9fdc928296561fbf25051e4275e03cfb7e130793
    network_mode: host
  aud_manager:
    image: ghcr.io/sifis-home/aud_manager:latest@sha256:1c2baed61c7188d6a92e3b378c851a959424183cafb4c8ea33bd257f14fcf35d
    network_mode: host
  sifis-runtime:
    image: ghcr.io/sifis-home/sifis-alpine-runtime-arm64v8:latest
    environment:
      - RUST_LOG=debug
    volumes:
     - /var/run/:/var/run/
  malware:
    image: ghcr.io/sifis-home/malware:2@sha256:c0089791100ebb3b142deee78896c712579e1777d8e06c071e6dc46b79465563
    network_mode: "host"
  usage-control-engine:
    image: ghcr.io/sifis-home/usage-control-engine:latest@sha256:84a2455b045f7afaa1a04464b0c0fc0746fd311b715b5954a63029dbbb1decd4
    network_mode: "host"
    volumes:
     - /opt/domo/domo-services:/services
  domo-scheduler:
    image: ghcr.io/domo-iot/domo-scheduler-arm64v8:6
    labels:
      reload: "true"
    network_mode: "host"
    volumes:
     - /opt/domo/domo-services:/services
     - /etc/domo:/etc/domo
  sifis-dht:
    image: ghcr.io/sifis-home/sifis-alpine-dht-arm64v8:latest
    labels:
      reload: "true"
    network_mode: "host"
    volumes:
     - /opt/domo/domo-dht-db:/root/domo-dht-db
     - /etc/ssl/certs:/etc/ssl/certs
     - /etc/domo:/etc/domo
```

Figure 17: Example of docker-compose.yaml file

As it can be observed, the docker-compose file contains the list of services that compose the SIFIS-Home Smart Device Framework. We want to highlight that, since all the components of SIFIS-Home are provided as Docker images, a running Docker engine suffices for a device to execute the whole SIFIS-Home framework.

4.3 *domo-bootstrap*

The domo-bootstrap binary is an application that allows a DoMO gateway to receive all the information it needs to join a SIFIS-Home network, such as its *node id* or the *secret key* to be used by the DHT Manager application.
In detail, when a DoMO gateway is turned on, the following actions are performed by the domo-bootstrap application. If the DoMO gateway has been already configured in the past, no further action is performed. Conversely, if the device has not been configured yet, the domo-bootstrap application takes care to 1) set the 2.4 Ghz WiFi interface of the DoMO Gateway in access point mode, 2) create a dedicated WiFi bootstrap network, whose SSID is *domo-<MAC>,* where MAC is the MAC address of the device, and password is a random one, 3) start a web server listening on port 8080 and exposing a POST */config* endpoint. We assume that the SSID and password of the WiFi bootstrap network can be easily retrieved by the owner of the DoMO gateway by scanning a QR code present on the DoMO Gateway enclosure box. Once a connection to the bootstrap network is performed, the /config endpoint should be used to configure the DoMO gateway.

The */config* endpoint of the web server accepts a json payload, whose format is reported in Figure 18.

```
{
    "dht_iot": "312d9bcc6e31e990d4ee888af0dba1c11c56be4959606c8a913a3db5d93c7a2c",
    "ssid": "DoMO",
    "pass": "password",
    "iot_ssid": "DOMOTESTSIFIS",
    "iot_pass": "domotest",
    "iot_network": "10.0",
    "mesh_id": "domomesh",
    "mesh_pass": "domomesh",
    "mesh_network": "192.168.20",
    "node_id": "1"
}
```

Figure 18: Example of json payload used to configure a DoMO gateway

As it can be observed, the endpoint allows to specify both application parameters such as the shared key to be used by the DHT Manager (*dht_iot* field) and system parameters such as the SSID and password of the network to which the NSSDs will connect to (*iot_ssid* and *iot_pass* fields). Other parameters such as the id of the device (*node_id*) as well as the mesh network credentials (*mesh_id* and *mesh_pass*) can be configured too. If the json payload sent to the /config endpoint is correct, the domo-bootstrap application takes care to 1) configure the OpenWrt system using the parameters contained in the json payload, 2) generate the *toml* files needed by the different SIFIS-Home components (such as the broker.toml shown in Figure 13) and place them in the /etc/domo folder, 3) start the dockerd process and execute the domo-docker service whose initscript is the following one:

```
root@domoserver-1:~# cat /etc/init.d/domo-docker
#!/bin/sh /etc/rc.common
# Copyright (C) 2022 DoMO srls

START=99
USE_PROCD=1
PROG="/usr/bin/docker"
COMPOSE_FILE="/opt/domo/domo-compose/docker-compose.yaml"

start_service() {
        procd_open_instance
        procd_set_param command $PROG compose -f $COMPOSE_FILE up
        procd_set_param stdout 1
        procd_set_param stderr 1
        procd_close_instance
}

stop_service() {
        $PROG compose -f $COMPOSE_FILE stop
}
```

Figure 19: domo-docker init script

As it can be observed, the domo-docker initscript launches the command "*docker compose up /opt/domo/ domo-compose/domo-compose.yaml"* causing all the services specified in that files to be executed. Hence, after the configuration phase, the SIFIS-Home Smart Device framework becomes active and operational on the DoMO gateway.

5 SIFIS-Home Framework Integration in the pilot

As it can be observed by looking at Figure 16, a number of SIFIS-Home components have been integrated and tested in the SIFIS-Home pilot. Specifically, the pilot implementation comprises components being part of the SIFIS-Home Cloud, Application, Smart Device and NSSD frameworks. In the following, we report the details of all the components integrated and used in the pilot, highlighting their functionalities, and how they have been integrated with each other.

5.1 *GitHub and GitHub Container Registry*

The SIFIS-Home framework is made up of different components and services. We used a dedicated GitHub repository for every component that we developed. We used GitHub Actions to build and test the different SIFIS-Home components. Also, specific GitHub actions have been used to build Docker images for the components of the SIFIS-Home framework. We used the GitHub Container Registry to store the Docker images that we produced. In the following sections, if possible, we provide a link to the repository and Docker image of every component that is described.

5.2 *SIFIS-Home Cloud Framework*

5.2.1 Yggio

Yggio is the most important component of the SIFIS-Home Cloud Framework. It provides an interface through which users of the smart home can have a clear view of the status of their devices, control them, see system logs, and have a list of the installed SIFIS-Home third-party applications. In the following sections, we describe the different Yggio components.

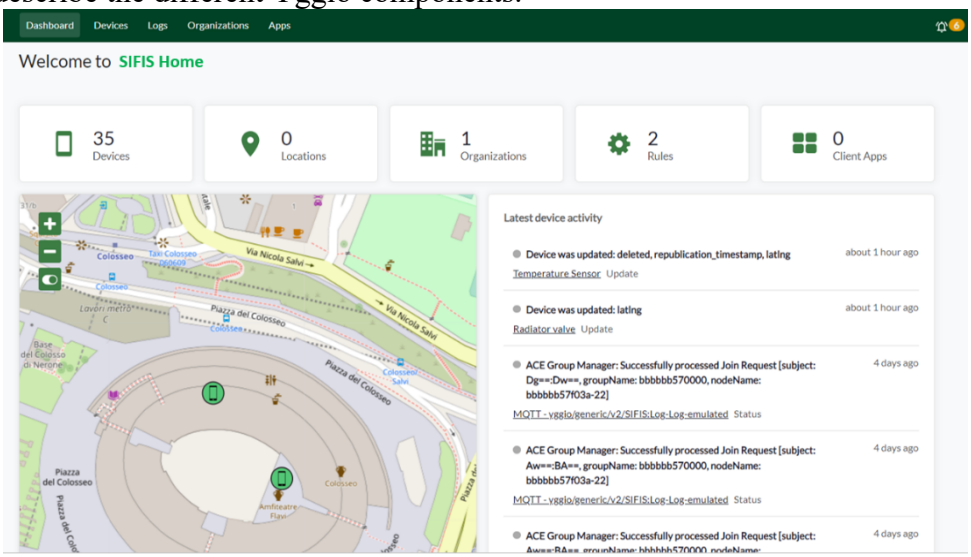


Figure 20: The start page of the Cloud Interface

Yggio / Cloud UI

The Sensative horizontal Internet of Things (IoT) integration platform Yggio is used as the backbone of the SIFIS-Home cloud interface. It provides the execution environment that makes Ratatosk FIWARE Context Broker possible to execute, and its API makes it possible to implement the SIFIS-Home overall web interface UI. Ratatosk in turn uses the open-source KeyCloak component as a security plug-in for access and authentication that provides a JWT – JSON Webtoken. The JWT is used by both the DHT and the Mobile Application to authenticate users and interact with the Ratatosk Rest API.

The cloud interface can be used to interact with the SIFIS-Home framework both within the Smart Home and outside the Smart Home via a feature-rich web UI.

FIWARE Context Broker Ratatosk: FIWARE NGSI v2 Ratatosk is a publish/subscribe Context Broker that holds the representation of a system state via FIWARE entities. The Context Broker implements [FIWARE NGSI v2 API] FIWARE NGSI v2 APIs. The FIWARE Context broker API is exclusively used to power the cloud user interface of the SIFIS-Home and is not involved in the actual secure SIFIS-Home network inside the Smart Home, which instead relies on a DHT to create a distributed and robust network without a single point of failure.

Each of the FIWARE entities is described in JSON via a data model. FIWARE defines recommended data models to simplify interoperability between systems at https://github.com/smart-data-models but, if none fits, it is also possible to define one's own data models or use a subset of an existing data model to represent the type of object one wants to describe.

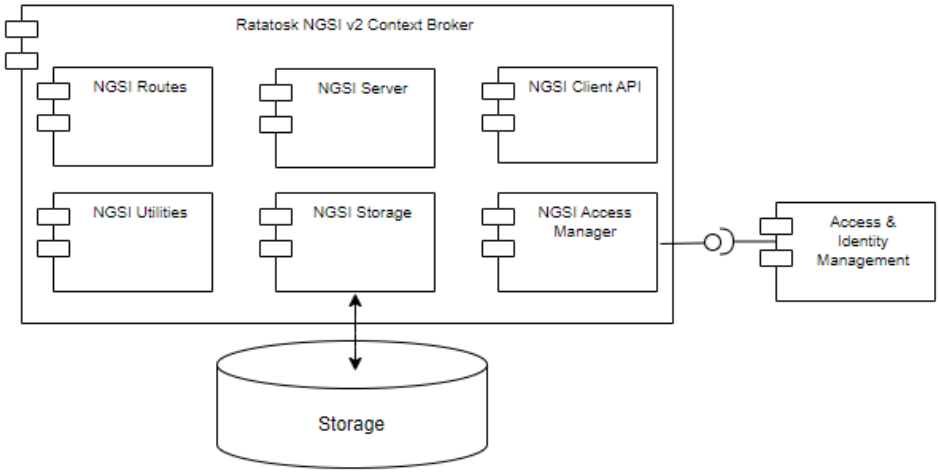


Figure 21: Ratatosk FIWARE Context Broker architecture

Ratatosk is a secure FIWARE Context Broker relying on the KeyCloak open-source component as a security plug-in, and that it always requires a valid authentication token to accept a command. This will be used to make sure that a user in the Smart Home who attempts to perform some actions has the required authorization to do so.

A design aspect of SIFIS-Home is that the cloud interface that implements the FIWARE Context Broker must be able to send a command through a firewall to the DHT-based network inside a Smart Home, and then execute some command, like turning on a lamp. The natural FIWARE solution to use NGSI subscriptions would unfortunately not work, since subscription requests are required to be IP addressable and will get blocked by the Smart Home firewall. The solution that we identified was to develop an MQTT-to-DHT bridge, i.e., the devices and analytics in the Smart Home both publish events and subscribe to the FIWARE Context Broker events not via the API and NGSI subscriptions, but rather via a standard, open-source MQTT broker that integrates with Ratatosk. The Ratatosk events will then be triggered either by the user via the UI or by other SIFIS-Home devices. There are more details about the MQTT-to-DHT bridge in Section DHT-TO-MQTT Manager.

The Ratatosk implementation is available here: https://github.com/sifis-home/yggio-ratatosk

Cloud Home

This is the main component of the User Interface and is used to launch other applications installed in the Smart Home system. The dashboard in the figure below shows some key metrics of the system, such as the number of devices and installed third-party applications (Client Apps in Figure 20).

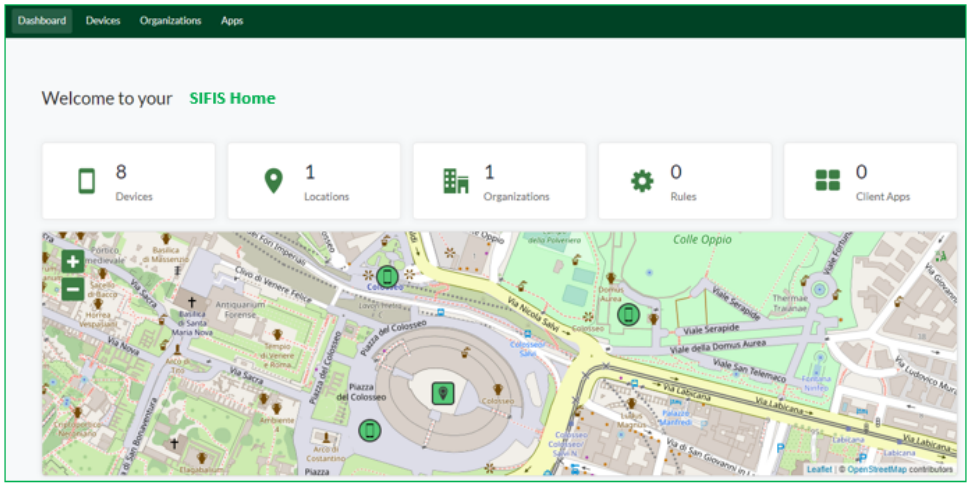


Figure 22: Home screen with a simple dashboard and a map

Cloud Device management

This component enables the configuration of the devices in the SIFIS-Home network. This is a core system component that manages onboarding, configuration, displaying of device status, and other functionalities related to the devices added to the Smart Home system. Depending on what role the logged in user has, different activities, like read or write data, are allowed.

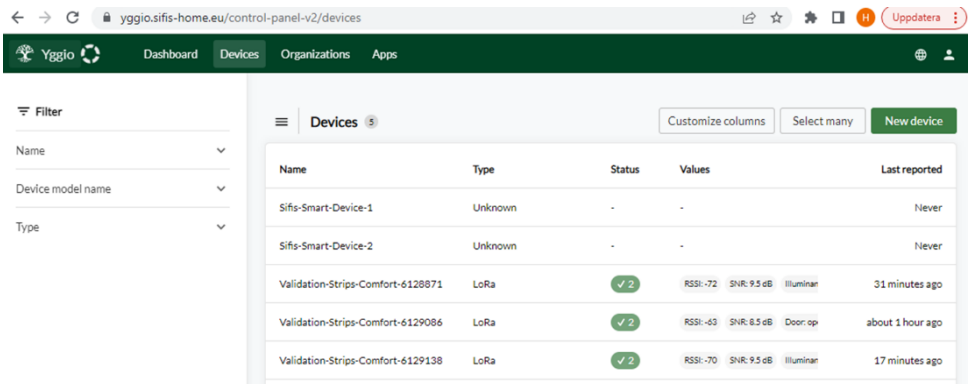


Figure 23: Device Manager with a list of devices

Cloud Settings

This component provides user interfaces for the configuration of the SIFIS-Home infrastructure and most items in the cloud interface, such as devices and analytics. Each visible item is represented by FIWARE entities in the Ratatosk Context broker. This component allows the user to view and edit the values of FIWARE entities.

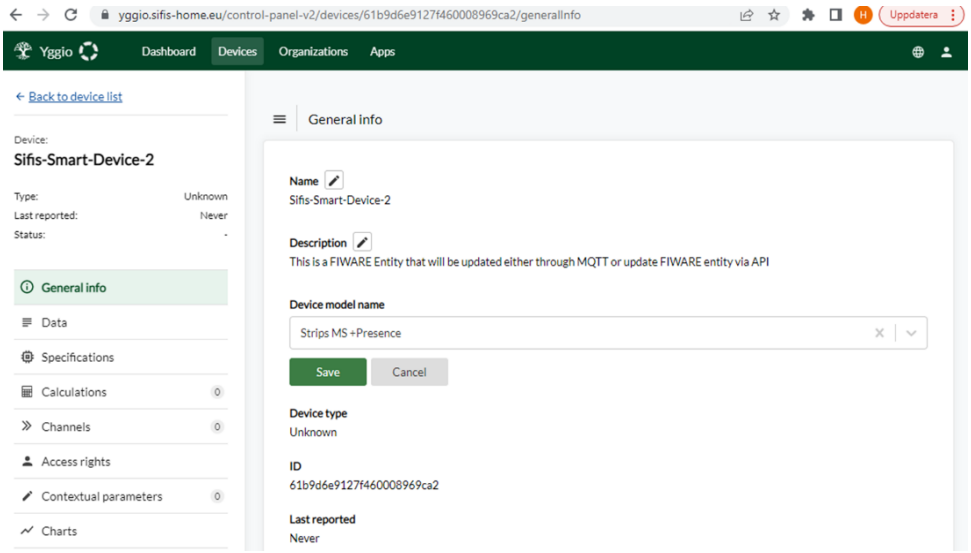


Figure 24: View status and edit settings for a SIFIS-Home Smart Device

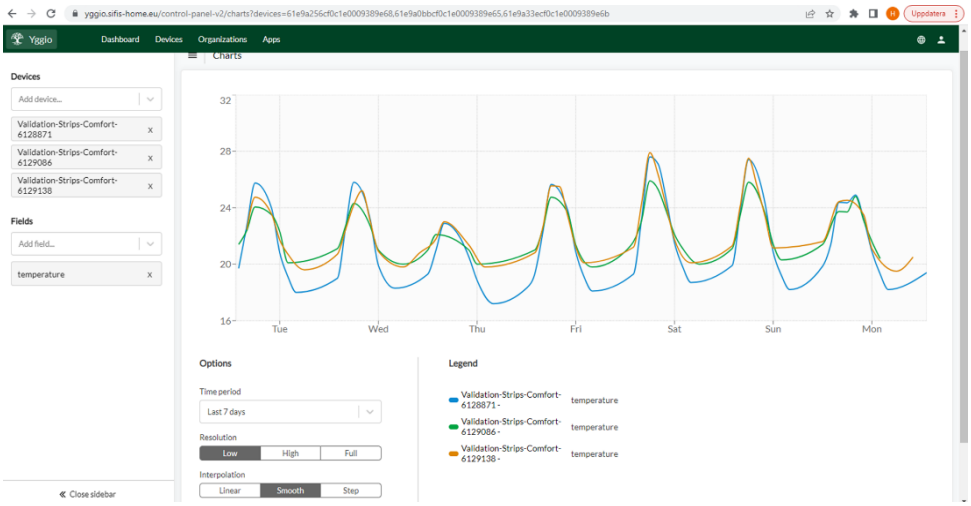


Figure 25: Compare time series data of 3 devices used for validation

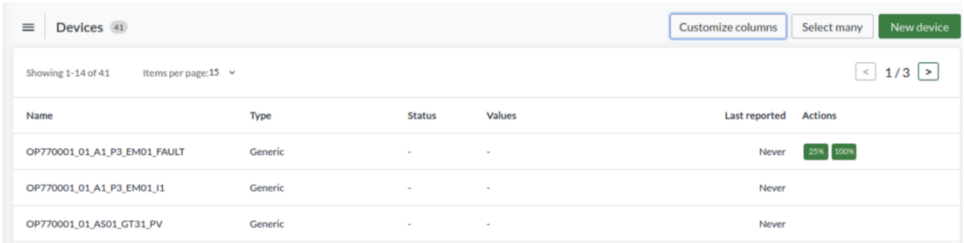


Figure 26: Control actuators via command buttons in the cloud interface

Figure 27: Create custom command buttons

The Cloud UI implementation is available here: https://github.com/sifis-home/yggio-components

Yggio Market Place

The Yggio Market Place, as depicted in Figure 17, is accessible through the cloud UI. It enables end users to see the list of available third-party applications and install them on their SIFIS-Home network.

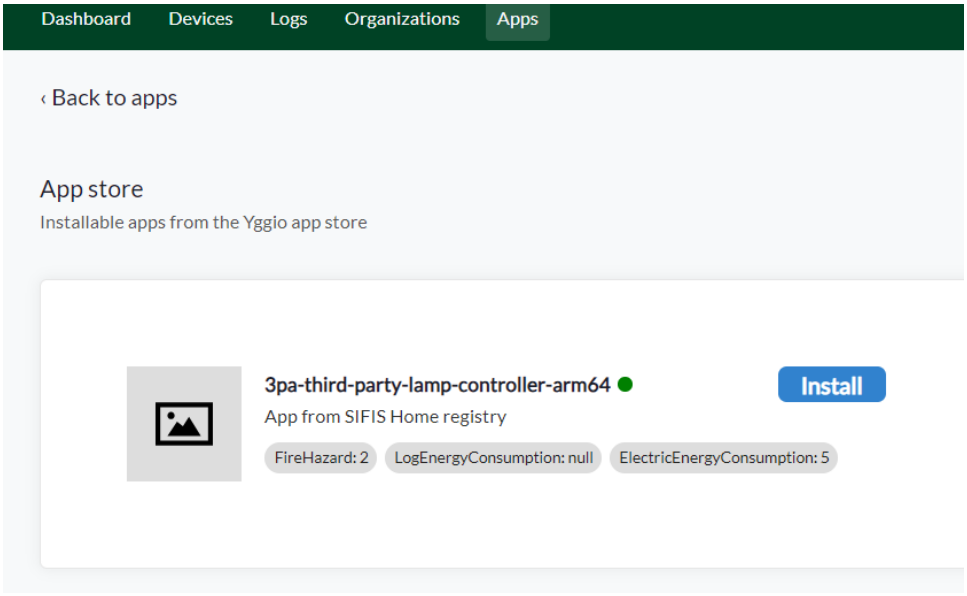


Figure 28: Example 3rd party app in the Market Place

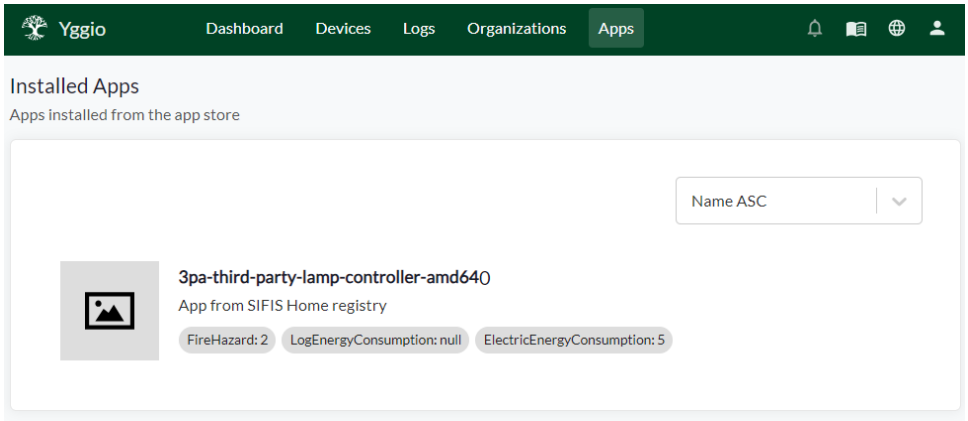


Figure 29: Example 3rd party app in the Market Place

Figure 17 Example of an installed app in the Market Place

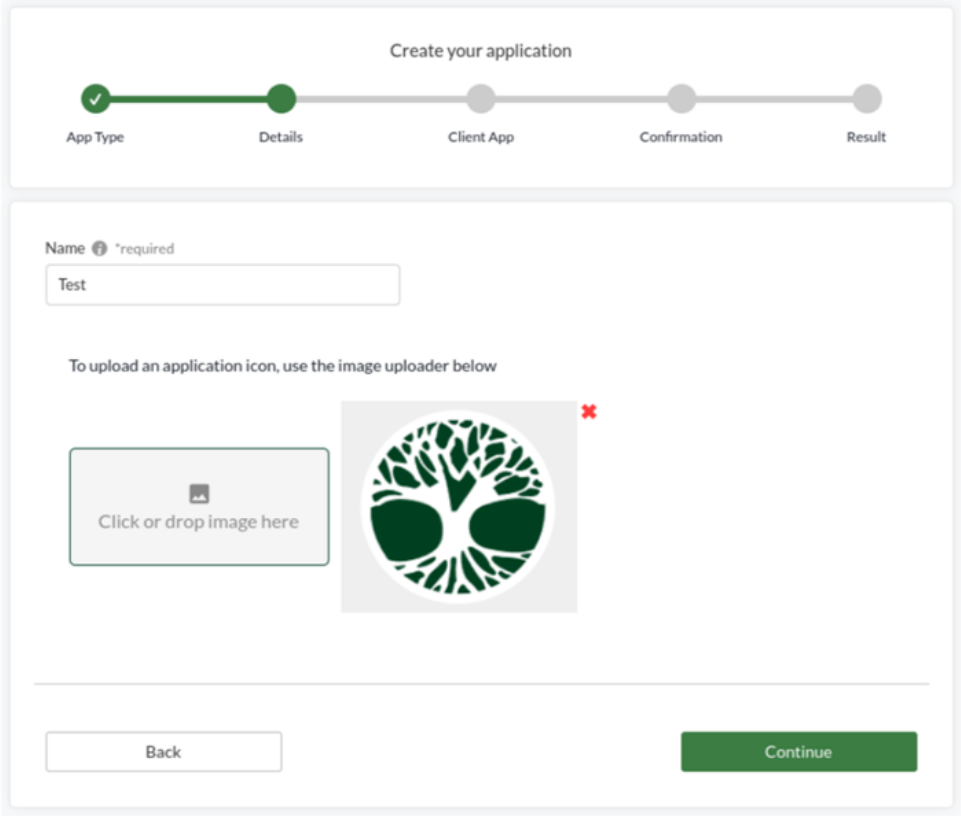


Figure 30: Edit or create an application in the Market Place

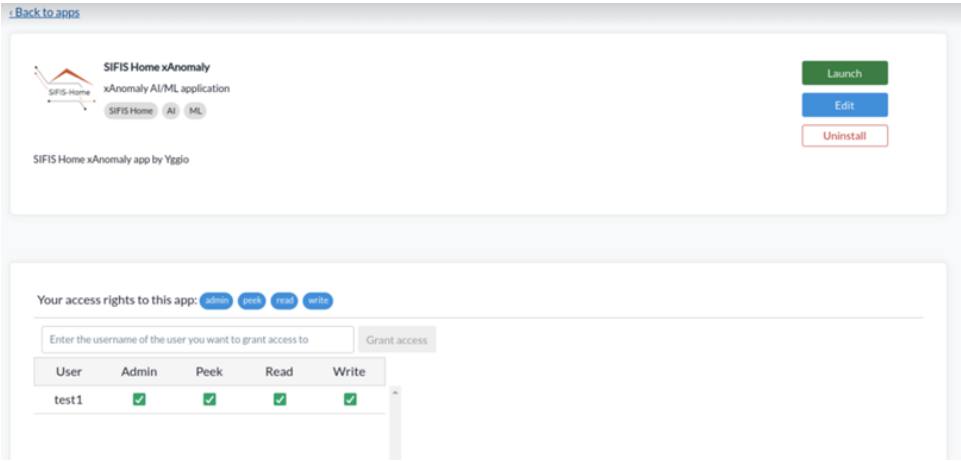


Figure 31: Share installed application with other users in the home

For seamless integration into the SIFIS-Home ecosystem, third-party applications are required to be uploaded to the SIFIS-Home GitHub Docker Container registry. Subsequently, these applications undergo an automatic security scanning process utilizing tools developed by WP2. Once the applications pass the security scanning successfully, they are appropriately labeled and are then made available in the Yggio Market Place. This meticulous procedure ensures that only trusted and secure applications are accessible to users, in the interest of preserving the integrity and safety of the SIFIS-Home environment.

The Yggio Market Place implementation is available here as an embedded application inside the control panel: https://github.com/sifis-home/yggio-components/tree/master/control-panel-v2/src/pages/apps

Alarm / Logs:

The functionality of this component encompasses two key features: displaying alarms in the cloud interface and the mobile application and gathering logs pertaining to the operation of the SIFIS-Home infrastructure. Concerning alarms, it provides a means to highlight any critical issues or events that require attention. Additionally, it facilitates the collection of logs, capturing essential information about the overall functioning of the SIFIS-Home system for analysis and monitoring purposes.

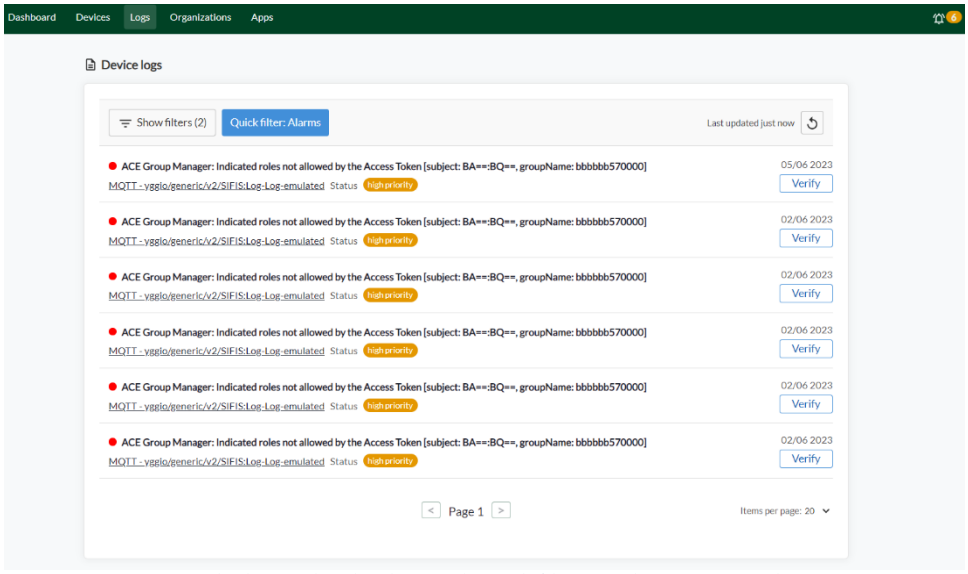


Figure 32: The log in the cloud UI with quick filter on alarms activated.

The alarm and log component receives logs from the DHT via MQTT. These logs are received under a JSON key named "log" located at the top level of each message. The "log" key contains crucial information such as type, priority, category, and message, which collectively determine how the log message should be handled and displayed in the UI.

When incoming logs have a priority level of "severe" or "high", they are categorized as alarms. As a result, the user will be notified of these alarms via both the cloud UI and the mobile application and provided with the ability to dismiss them.

{
- Some key values…
- Log : {
  - **type** The type of log. Must be 'info', 'warning', or 'error'.
  - **priority** Must be 'low', 'medium', 'high', 'severe'. Medium, high and severe priority logs can be `verified` by the user. High and severe priority logs are shown as alarms in the control panel.
  - **category** Must be 'status', 'update', 'access', 'command', 'uplink', 'rule' or 'analytics'
  - **message** The log message. Maximum 200 characters.

  }

}

Figure 33: The log JSON structure of logs

Each log within the SIFIS-Home system is equipped with access rights, ensuring that only end users who have access to the device that generated the log can view and dismiss alarms related to that device. This ensures that the responsibility for handling and acknowledging device-specific alarms lies solely with the users who have direct access to the respective device.

5.2.2 Marketplace

All the SIFIS-Home third-party applications are provided to the end user of the smart home as Docker images that are ready to be downloaded and run on the SIFIS-Home SDs. The Marketplace is the component responsible for providing access to the third party applications Docker images of the different SIFIS-Home third-party applications and make them available for download. Moreover, the Marketplace provides an API through which the metrics of a third-party application such as its software quality and its hazards list can be retrieved. We use the GitHub Container Registry to store the SIFIS-Home Docker images of the third-party applications and decided to use specific Docker Labels to store the third-party application metrics. Repository https://github.com/sifis-home/third-party-application, shows the procedure that we used to create a third-party application Docker image and add labels to it. Also, it is reported how the Docker Labels associated with a certain image can be retrieved using the GitHub Container Registry API. Such APIs are used by both Yggio Marketplace and the Mobile Application to produce the list of available third-party applications and show their metrics (see Figure 15).

Figure 34: MarketPlace mobile app view

5.2.3 VPN Server

The VPN Server is the technical solution through which we make services running on the SDs accessible from a remote side. In detail, our goal is to make the DHT Manager web service, the privacy dashboard panel, and the policy translation point panel running on a SD deployed in the house available from a remote side. Specifically, given that *yggio.sifis-home.eu* is the DNS name associated with the Panarea server, our solution allows to reach the DHT web service at the address *yggio.sifis-home.eu:3000*, the privacy dashboard panel at the address *yggio.sifis-home.eu:11000,* and the policy translation panel at the address *yggio.sifis-home.eu:9000*.

The VPN Server solution is made up of two different components, a wireguard server and a nginx proxy server. The configuration of the wireguard server that we use is reported in Figure 15. As it can be observed, the server is configured in such a way that it will create a wireguard interface, namely wg0, that will use address 10.43.89.1. Moreover, only one single client (i.e., the wireguard client running on the leader SD deployed in the house, see also section domo-scheduler) can be connected to the server and will have address 10.43.89.2 assigned to it. We highlight that UDP port 51820 has been opened on the Panarea server to allow wireguard clients to connect to the server.

Table 2: configuration of the wireguard server running on Panarea

```
########
# Wireguard server configuration for
########

[Interface]
Address = 10.43.89.1/24
ListenPort = 51820
PrivateKey = <wireguard_server_private_key>
# Deny internet traffic
PostUp = iptables -I FORWARD -i %i -o %i -j ACCEPT; iptables -i %i -A FORWARD -j REJECT;
PostDown = iptables -I FORWARD -i %i -o %i -j ACCEPT; iptables -i %i -A FORWARD -j REJECT;


[Peer]
PublicKey = <wireguard_client_public_key>
AllowedIPs = 10.43.89.2/32, 224.0.0.251/32
PersistentKeepalive = 15
```

The configuration of the nginx server that we use to redirect requests arriving to the Panarea server towards the SD connected through the VPN tunnel is reported in Figure 16. In detail, the nginx proxy forwards requests arriving to port 3000 of the Panarea server to the DHT Manager running on the connected SD and running on port 3000, request arriving to port 9000 of Panarea to the Policy translation panel running on the connected SD on port 9000 and, finally, requests on port 11000 to the Privacy dashboard panel running on port 11000 on the connected SD. Please note that 10.43.89.2 is the IP address of the leader SD.

Table 3: configuration of the nginx proxy running on the Panarea server

```
load_module                    /usr/lib/nginx/modules/ngx_stream_module.so;
worker_processes                                                          1;

events                                                                    {
 worker_connections                                                     256;
}

stream                                                                    {
 server                                                                   {
  listen                                                                3000;
  proxy_pass                                                  10.43.89.2:3000;
 }

 server                                                                   {
    listen                                                              9000;
    proxy_pass                                                  10.43.89.2:9000;
 }

 server                                                                   {
    listen                                                             11000;
    proxy_pass                                                  10.43.89.2:11000;
 }

}
```

Table 4 shows a successful ping operation from the Panarea server towards the SD VPN client. Also, Figure 35 and Figure 36 show the Privacy dashboard panel and the Policy translation point panel, respectively, that have been accessed through the VPN tunnel.

Table 4: Successful ping operation from Panarea to the connected SD using the VPN connection

```
1811aa747b95:/# ifconfig wg0
wg0       Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.43.89.1  P-t-P:10.43.89.1  Mask:255.255.255.0
          UP POINTOPOINT RUNNING NOARP  MTU:1420  Metric:1
          RX packets:41896668 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20957214 errors:7 dropped:27291 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:49216820472 (45.8 GiB)  TX bytes:2858816684 (2.6 GiB)

1811aa747b95:/# ping 10.43.89.2
PING 10.43.89.2 (10.43.89.2): 56 data bytes
64 bytes from 10.43.89.2: seq=0 ttl=64 time=0.685 ms
64 bytes from 10.43.89.2: seq=1 ttl=64 time=0.696 ms
64 bytes from 10.43.89.2: seq=2 ttl=64 time=0.663 ms
64 bytes from 10.43.89.2: seq=3 ttl=64 time=0.657 ms
```

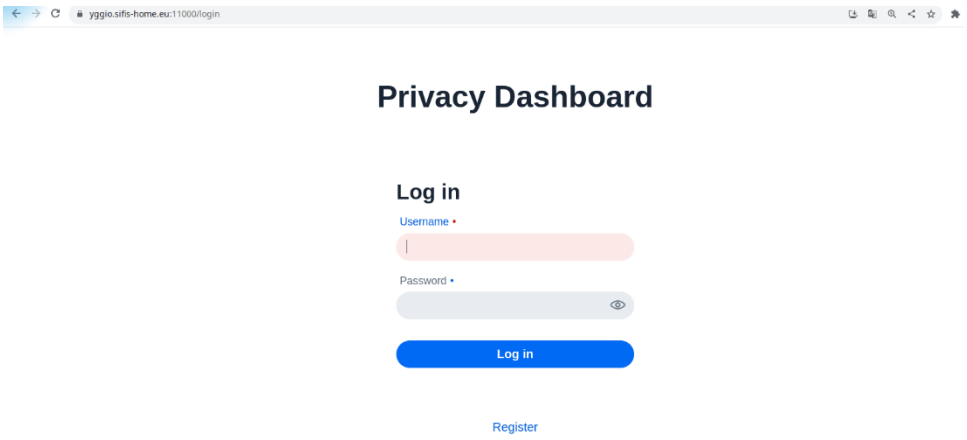


Figure 35: Privacy Dashboard login interface

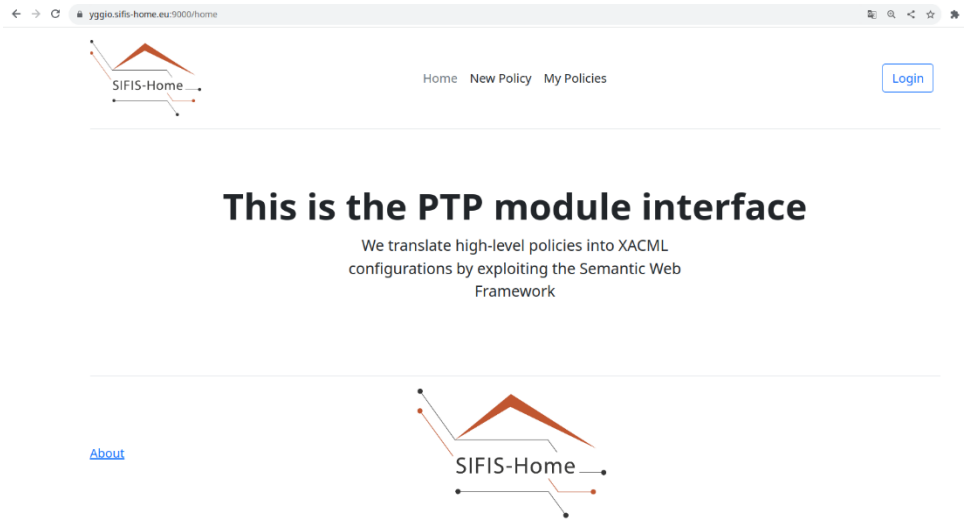


Figure 36: Policy translation point dashboard

5.3 *SIFIS-Home Application Framework*

The Mobile Application within the SIFIS-Home system serves as a user-friendly interface for end users to interact with the SIFIS-Home framework. Through the Mobile Application, users can easily manage the SIFIS-Home network basic functionalities. This includes listing the installed devices within the home, enabling users to perform various actions on these devices, such as collecting environment measure readings, controlling actuators, and turning devices on or off.

Additionally, the Mobile Application provides access to system logs for monitoring purposes, as well as the ability to install third-party applications directly into the SIFIS-Home framework, in order to extend the capabilities of the system.

The Mobile Application was written in Javascript and Vue utilizing the NativeScript framework and the Stackblitz development environment. Mobile Application can run both on Android and iOS with the NativeScript preview application available at https://preview.nativescript.org/.

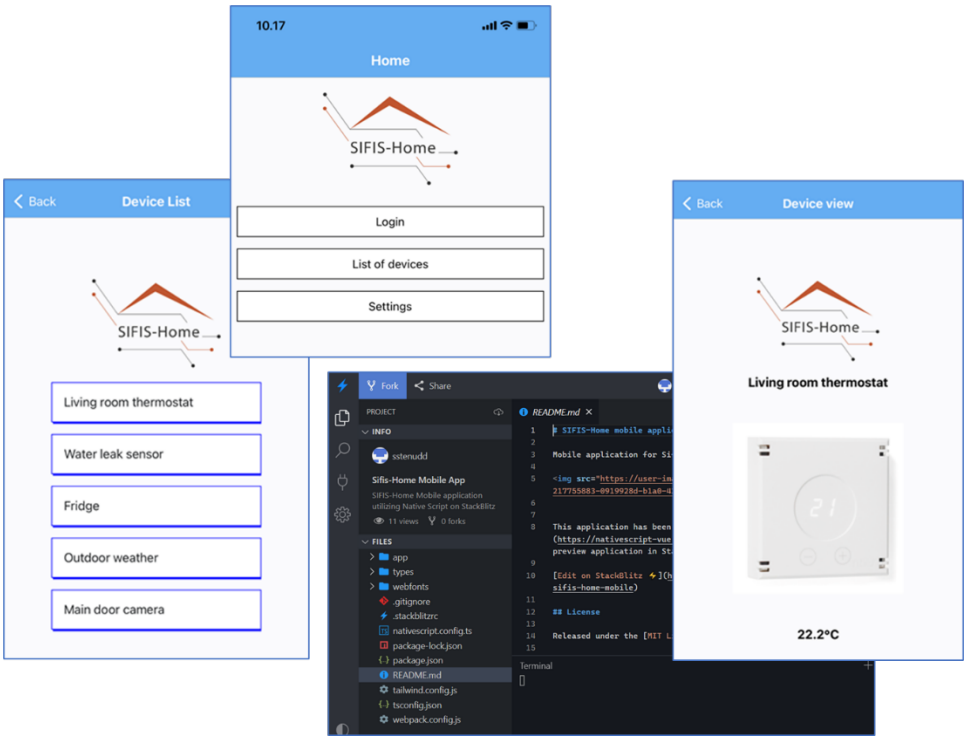


Figure 37: Mobile Application UX

Login

The Mobile Application performs the login operation using Yggio API (https://yggio.sifis-home.eu/swagger). Yggio API provides a simple method to send authentication credentials and as a POST request, and it returns a JWT token on successful login. This token is stored on the device which performed the request and is used for future communications with DHT.

Figure 38: Login view

Devices

The Mobile Application collects the topics that represent devices and visualizes data and possible actions provided by the devices.

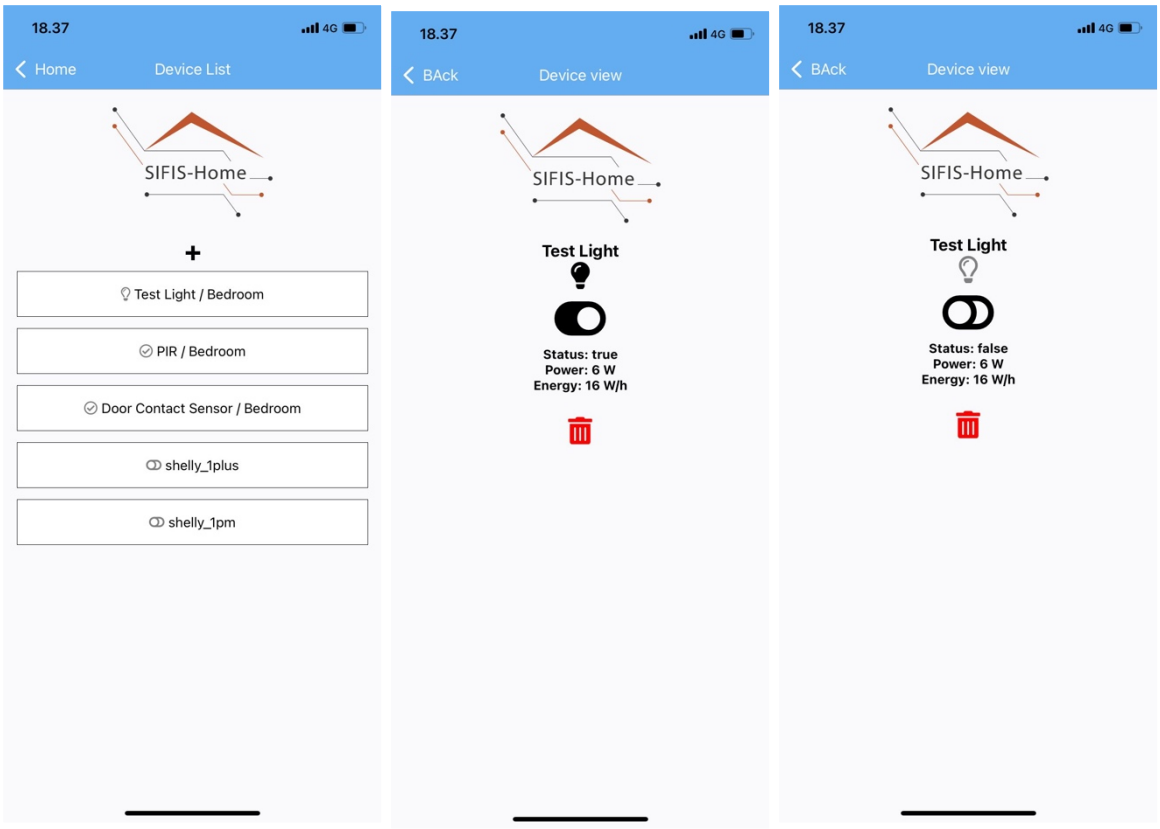


Figure 39: Devices view

Marketplace

The SIFIS-Home marketplace provides a list of the third-party applications that can be installed.
Such a list is retrieved from ghcr.io, under the SIFIS-home organization, where the docker images of the third-party applications are stored.
The name all the third-party applications starts with the prefix "3pa-", e.g., "3pa-third-party-lamp-controller-amd64", so the marketplace applies this filter to show the third-party applications only.

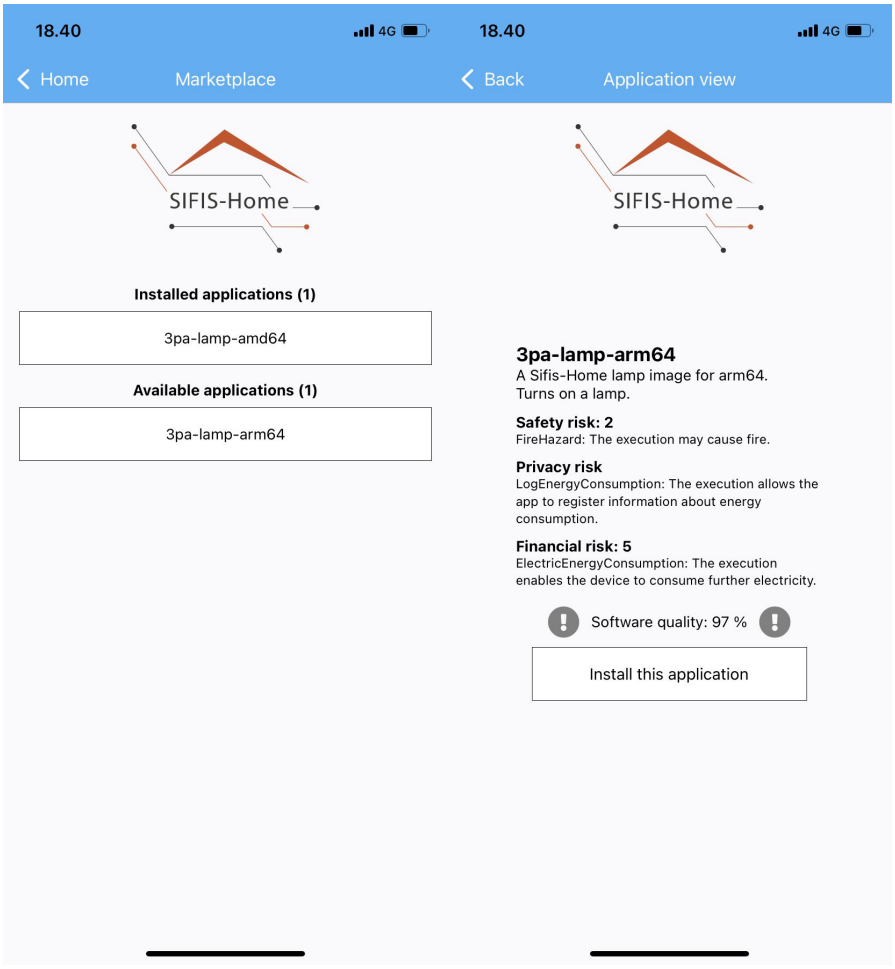


Figure 40: Marketplace view

Statistics & Analytics

The Mobile Application connects to Yggio logs API to fetch the list of system events. The logs are rendered and shown to the end user.

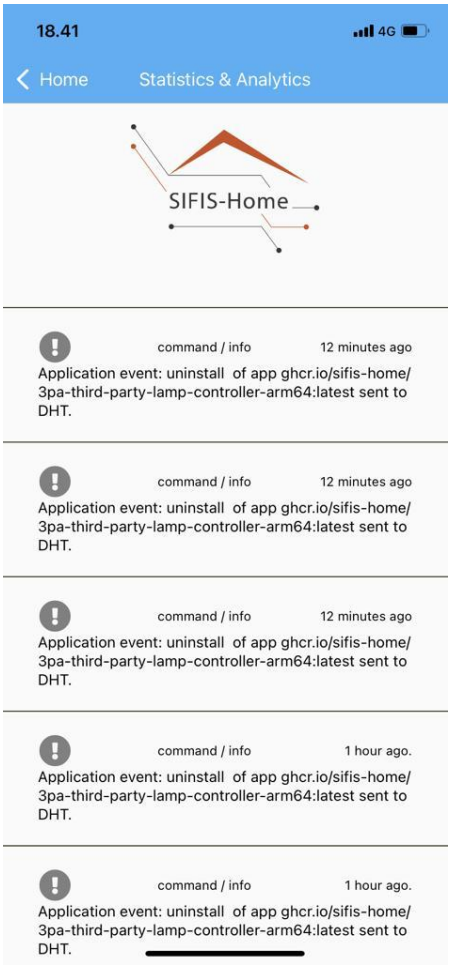


Figure 41: Logs view

Settings

Mobile Application gives option to define the DHT server address and select whether the DHT is used local or remote.

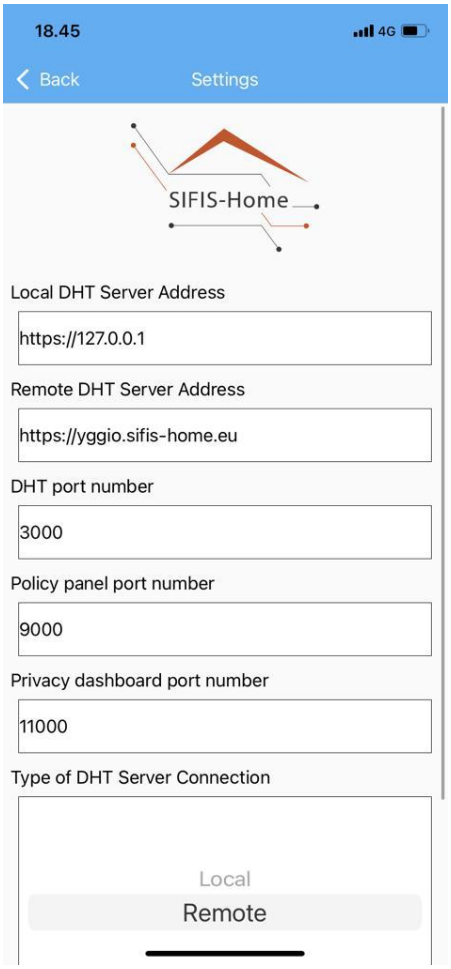


Figure 42: Settings view

Notifications

The mobile application shows notifications through a banner which appears on each view. Notifications are meant to inform about critical events, still all kind of notifications can be given using this interface. The integration of the notifications with the SIFIS-Home framework is done through a specific Notification topic, which can be used by any SIFIS-Home framework component, plus the Yggio interface which is used to relay the notification to the mobile application, even when the mobile device is outside the smart home premises.

Figure 43: Example of notification

Privacy dashboard and Policy definition Panel

The privacy dashboard and policy definition panel are deployed on in the SIFIS-Home framework as

separate services. Mobile Application provides WebView to use them. Both services can be used remotely when DHT is used via VPN connection.

5.4 *SIFIS-Home Smart Device Framework*

The SIFIS-Home Smart Device Framework is composed of the following macro components:

- Secure Lifecycle Manager
- Secure Communication Layer
- Proactive Security Management Layer
- Application Toolboxes
- Api Gateway
- DHT Manager
- VPN Manager
- NSSD Manager

In the following sections, we briefly describe all the components of the different macroblocks of the Smart Device Framework, which have been integrated and used in the pilot. However, we first need to introduce some "use-case-specific" components that have been specifically developed to optimize the performance of the SIFIS-Home Smart Device while operating in the pilot environment.

We want to highlight that, given the heterogeneity of the pilot devices, for all the components of the SIFIS-Home SD framework we built Docker images targeting both the armv8 and amd64 architecture.

5.4.1 Use-case-specific components

This set of components has been specifically developed to optimize the performance of the SIFIS-Home framework when executed in the DOMO testbed.

5.4.1.1 domo-scheduler

The domo-scheduler is an application running on all the SIFIS-Home SDs. It is a Rust application that has direct access to the DHT since it embeds the DHT as an external library. Its role is to elect one *leader* device among all the different SDs that are present in a certain SIFIS-Home network, in a dynamic way. The leader device is the only one allowed to execute specific services of the smart home, that are named *cluster* services. In detail, only one single instance of a cluster service should be running inside the SIFIS-Home network at a given point in time. This is because such services take decisions and coordinate specific smart home functionalities and, hence, do not need to be executed on multiple devices at the same time. The domo-scheduler takes as input the *id* of the node where it is executing and uses specific volatile messages published on the DHT to elect a leader node in a completely dynamic and distributed way. The following picture shows an example *toml* file used to configure the domo-scheduler application.

Table 5: domo-scheduler configuration file

```
[domo_scheduler]
node_id = 1

[domo_scheduler.cache]
```

```
url = "sqlite::memory:"
table = "domo_scheduler"
persistent = true
shared_key = "<secret_dht_key>"
loopback = false
```

The actions that are periodically executed by the domo-scheduler application are the following ones. First, a volatile message whose format is reported in Figure 17 is sent.

Table 6: domo-scheduler message format

```
{
    “name”: “domo-scheduler”,
    “message”: {
        “node_id”: <node_id>,
        “publication_timestamp”: <current_timestamp>
    }
}
```

As it can be observed, the message contains the identifier of the node where the domo-scheduler application is currently running and the time at which the message has been generated. Please note that due to the pub/sub functionality of the DHT, every domo-scheduler message that is published on the DHT is received by all the domo-scheduler applications running on the other SDs at a given point in time.

The domo-scheduler application stores the different domo-scheduler messages that it receives from the DHT. Then, it periodically runs the following simple algorithm to decide if the node where it is currently executing has the right to become the leader device. First, The domo-scheduler expunges all the domo-scheduler messages with "publication_timestamp" older than 30 seconds with regards to the current time. Second, the "lowest_external_id" is computed as the lowest "node_id" found in the set of the domo-scheduler messages. If the node id of the device where the domo-scheduler is running is lower than *lowest_external_id,* the current device becomes the leader_node. Otherwise, another node in the network should be the leader one.

The domo-scheduler uses a simple mechanism to signal to the cluster services running on the node that the device is, currently, the leader one. Specifically, it creates a file named */services/leader_file.txt* whenever the node becomes the leader one and removes the file whenever another node is selected to be the leader node.

A mechanism such as the one reported in Figure 18 should then be used to monitor, execute, and terminate a cluster service. In the pilot implementation we used simple bash scripts that we called *launch scripts*, to implement the functionality reported in Figure 18.

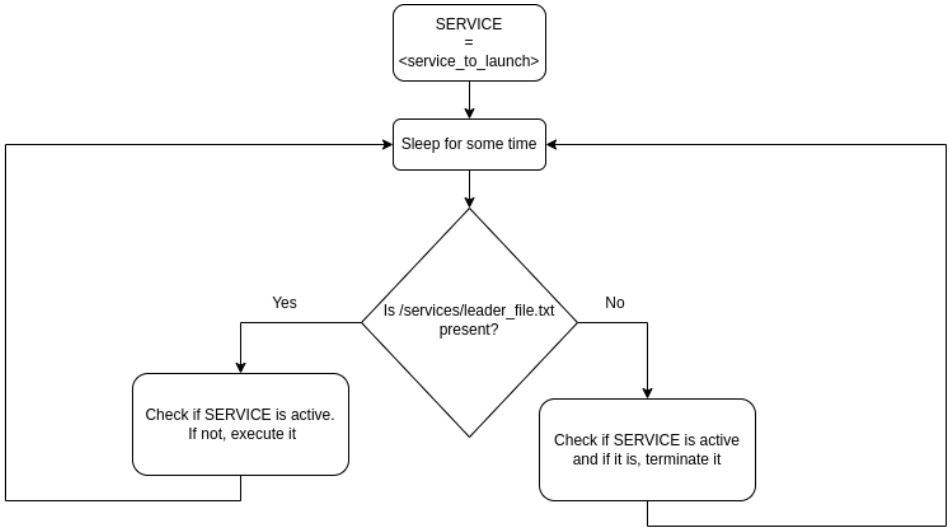


Figure 44: Executing a cluster service

In detail, the lifecycle of a cluster service, named SERVICE in the example of Figure 44, is managed by a "launch script", which periodically performs the following actions. First, file /services/leader_file.txt is checked. If it exists, this means that the node is the leader one. Hence, the launch script checks if SERVICE is running or not. If not, SERVICE is launched. Conversely, if file /services/leader_file.txt is not present, the node is not the leader one. Hence, if SERVICE is running, it is terminated.

Having a domo-scheduler application and launch scripts running on all the SDs of the system, provides an easy and robust way to launch cluster services.

5.4.1.2 SIFIS-SD-nginx-server

On all the SDs a nginx proxy server is used to protect the web services running on the SDs from external access. In detail, the nginx proxy server provides TLS termination for the DHT Manager web service, the Privacy Dashboard panel as well as the Policy Translation Point Panel (see Figure 19).

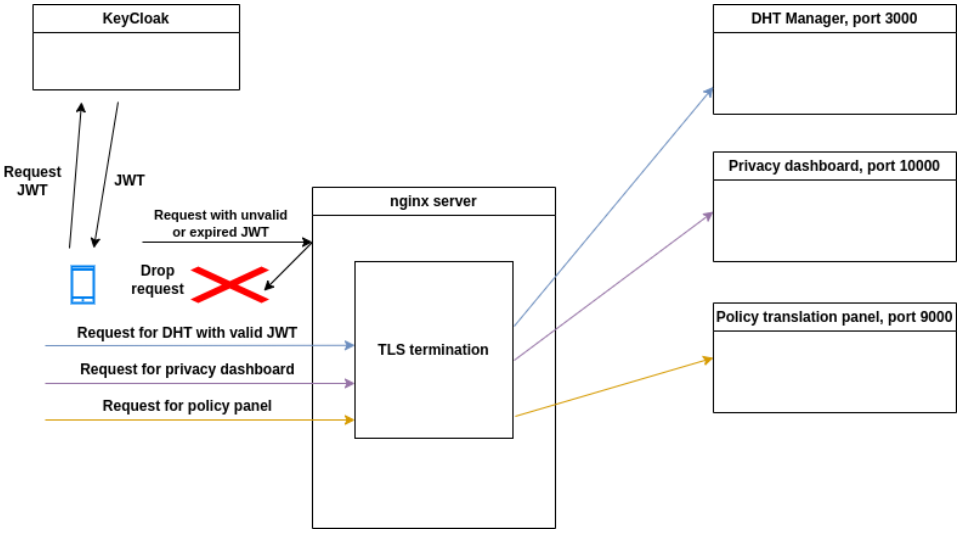


Figure 19: nginx-dht service operations

Moreover, since only authorized users should have access to the DHT Manager web service, the nginx proxy requires that all the requests towards the DHT contain a valid Json Web Token (JWT) in their HTTP Authorization header. A DHT request is considered valid if its JWT has been released by Keycloak and it has not expired yet. Figures 20 and 21 report the results of two DHT requests. The first one is valid, while the second one is not.

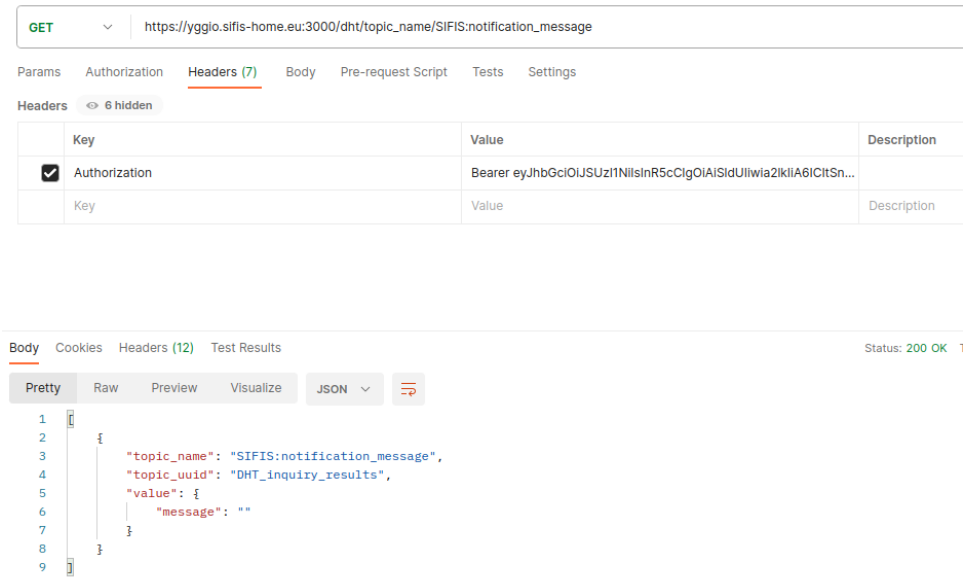


Figure 20: successful DHT request

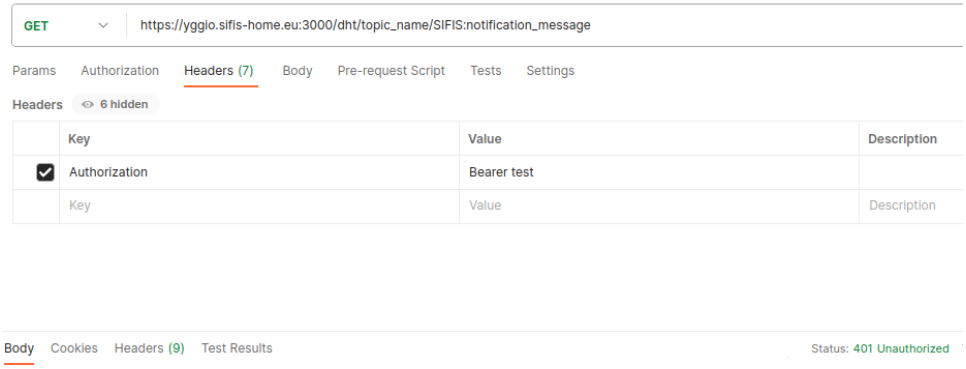


Figure 21: unsuccessful DHT request

GitHub repository	Docker Images URLs
https://github.com/sifis-home/sifis-sd-nginx/	*ghcr.io/sifis-home/domo-nginx-dht-arm64v8:latest* *ghcr.io/sifis-home/domo-nginx-dht-amd64:latest*

5.4.1.3 SIFIS Light Manager

The pilot implementation includes a light manager component also. Such component takes care to turn on and off the physical lights of a certain room when the physical buttons of the room are pressed/released by the users of the smart home.

The light manager is a cluster service that only runs on the leader SD. It has been developed using the Python language and uses the WebSocket and REST Api of the DHT to detect status changes of buttons and bistable buttons and to send the appropriate commands to the physical lights.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/uc16-light-manager	*ghcr.io/sifis-home/uc16-light-manager:latest*

5.4.2 Secure Lifecycle Manager

The components being part of the Secure Lifecycle Manager are reported in Figure 22. For the Secure Lifecycle Manager subcomponents, no noticeable differences have to be reported with respect to what has been described in D5.4. Hence, in the rest of this section, for each subcomponent, we only report a brief description and a link to their GitHub repository and Docker images. Additional details of the components can be found in D5.4.

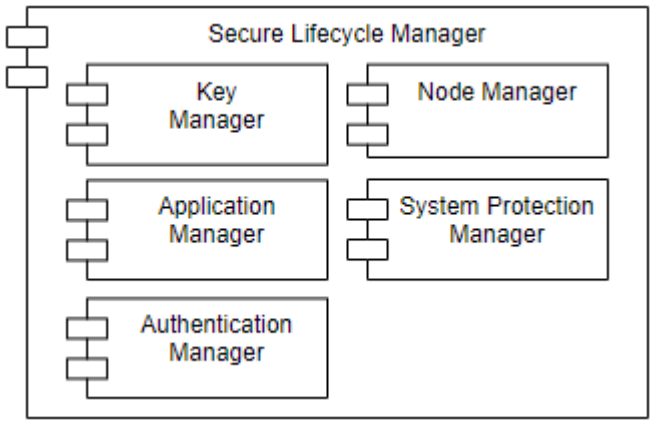


Figure 45: Secure Lifecycle Manager

5.4.2.1 *Application Manager*

The application manager is the component responsible for controlling the deployment, execution and removal of the SIFIS-Home third-party applications.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/Application-Manager	*ghcr.io/sifis-home/application-manager:latest*

5.4.2.2 *Node Manager*

The node manager component allows the dynamic joining of nodes and handles the removal of nodes in different circumstances.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/node-manager	*ghcr.io/sifis-home/node-manager-amd64:latest* *ghcr.io/sifis-home/node-manager-arm64:latest*

5.4.2.3 *System Protection manager*

It is the component that receives inputs from the various monitors and triggers actions such as node or application removal by communicating with the Application Manager and Node Manager through the

DHT.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/System-Protection-Manager	*ghcr.io/sifis-home/system-protection-manager:latest*

Authentication Manager and Key Manager
The following table reports a link to the implementation of the security solutions developed in WP3 pertaining to the "Secure Lifecycle Manager" module.

GitHub repositories
https://github.com/sifis-home/wp3-solutions *https://github.com/sifis-home/ace-ucs* *https://github.com/sifis-home/ace-entities* *https://bitbucket.org/marco-tiloca-sics/ace-java* *https://bitbucket.org/marco-tiloca-sics/ace-java* *https://github.com/rikard-sics/californium/tree/edhoc*

5.4.3 Secure Communication Layer

The components being part of the Secure Communication Manager are reported in Figure 24. No noticeable differences have to be reported with respect to what has been described in D5.4. Hence, in the rest of this section, for each subcomponent, we only report a brief description and a link to their GitHub repository and Docker images. Additional details of the components can be found in D5.4.

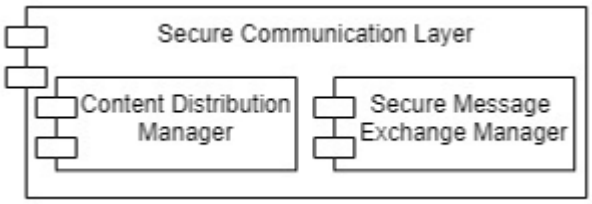


Figure 46: Secure Communication Layer

5.4.3.1 *Secure Message Exchange Manager and Content Distribution Manager*

The following table reports a link to the implementation of the security solutions developed in WP3 pertaining to the "Secure Communication Manager" module.

GitHub repositories
https://github.com/sifis-home/wp3-solutions *https://github.com/sifis-home/ace-ucs* *https://github.com/sifis-home/ace-entities* *https://bitbucket.org/marco-tiloca-sics/ace-java* *https://bitbucket.org/marco-tiloca-sics/ace-java* *https://github.com/rikard-sics/californium/tree/edhoc* https://github.com/rikard-sics/californium/tree/group_oscore

5.4.4 Proactive Security Management Layer

The components being part of the Proactive Security Management Layer are reported in Figure 25. No noticeable differences have to be reported with respect to what has been described in D5.4. Hence, in the rest of this section, for each subcomponent, we only report a brief description and a link to their GitHub repository and Docker images. Additional details of the components can be found in D5.4.

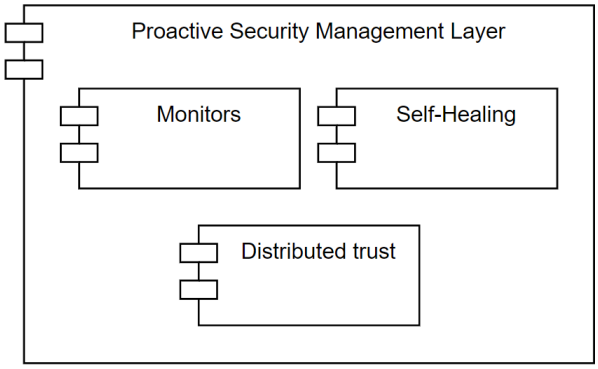


Figure 47: Proactive Security Management Layer

5.4.4.1 *Monitors*

This component is a collection of services that log specific events at different levels:

DHT Monitor

Implemented through the libP2P library, it logs the number and type of operations performed on the DHT.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/dht-monitor	ghcr.io/sifis-home/dht-monitor:latest

Application Monitor

This service in-lines security critical APIs to log and control their behaviour and rights to be executed when they are invoked by third-party applications.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/sifis-api *https://github.com/sifis-home/sifis-message*	*ghcr.io/sifis-home/sifis-alpine-runtime-arm64v8:latest*

Network Monitor

It acquires traffic by capturing packets via iptables. It is intended to run on central networking devices, such as routers through which other devices communicate both inside the SIFIS-Home network and to the outer Internet.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/wp4-aud_manager	*ghcr.io/sifis-home/aud_manager:latest*

SysCall Monitor

It collects system call events through a REST API and conveys them for further assessment to the responsible analytic in the Data Analytic Toolbox component of the Application Toolboxes module.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/Kernel-Monitor	*ghcr.io/sifis-home/kernel-monitor:latest*

5.4.4.2 *Distributed Trust*

The distributed trust component continuously assigns to each smart device a trust score and manages distributed decisions under biased voting. The distributed trust management has been integrated as a functional module of the Node Manager.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/node-manager	*ghcr.io/sifis-home/node-manager-amd64:latest* *ghcr.io/sifis-home/node-manager-arm64:latest*

5.4.4.3 *Self Healing*

The self-healing component has been integrated as a functional module of the Node Manager.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/node-manager	*ghcr.io/sifis-home/node-manager-amd64:latest* *ghcr.io/sifis-home/node-manager-arm64:latest*

5.4.5 Application Toolboxes

The components being part of the Application Toolboxes are reported in Figure 27. No noticeable differences have to be reported with respect to what has been described in D5.4. Hence, in the rest of this section, for each subcomponent, we only report a brief description and a link to their GitHub repository and Docker images. Additional details of the components can be found in D5.4.

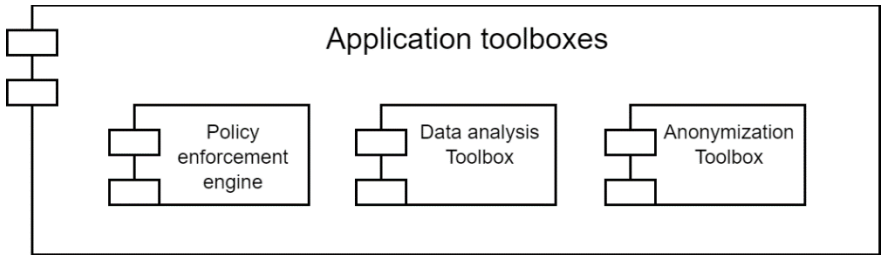


Figure 48: The application toolboxes

5.4.5.1 *Data Analysis Toolbox*

This component of the SIFIS-Home framework is devoted to the execution of the analytics on the data collected from the sensors and Smart Devices in the Smart Home.

GitHub repositories	Docker Images URLs
Privacy-Aware Speech Recognition (https://github.com/sifis-home/flask-whisper-speech-recognition)	*ghcr.io/sifis-home/flask-whisper-speech-recognition:latest*
Privacy-Aware Face Recognition (https://github.com/sifis-home/flask-private-deepface).	*ghcr.io/sifis-home/flask-private-deepface:latest*

Privacy-Aware Device Anomaly Detection (https://github.com/sifis-home/flask-device-anomaly-detection).	*ghcr.io/sifis-home/flask-device-anomaly-detection:latest*
Privacy-Aware Parental Control (https://github.com/sifis-home/flask-parental-control).	*ghcr.io/sifis-home/flask-parental-control:latest*
Privacy-Aware Object Recognition (https://github.com/sifis-home/flask_object_recognition)	*ghcr.io/sifis-home/flask_object_recognition:latest*
Privacy-Aware Speaker Verification (https://github.com/sifis-home/flask_speaker_verification)	*ghcr.io/sifis-home/flask_speaker_verification:latest*
Privacy-Aware Audio Signal Classifier (https://github.com/sifis-home/flask_audio_signal_classifier)	*ghcr.io/sifis-home/max-model:latest*
Analytics API (https://github.com/sifis-home/analytics_api)	*ghcr.io/sifis-home/analytics_api:latest*

5.4.5.2 *Anonymization Toolbox*

The anonymization toolbox contains software tools that preserve privacy of data before, during, and after the analysis of such data. The tools have been integrated as functional modules of the analytics modules.

5.4.5.3 Policy Enforcement Engine

There are three different modules being part of the Policy Enforcement Engine, namely the Usage Control Engine, the Policy Translation Point and the Privacy dashboard. They are briefly described below and link to their source code and Docker images is also provided.

Usage Control Engine

The Usage Control Engine is implemented following the Usage Control (UCON) model. The UCON model allows dynamic evaluation of access policies through mutable attributes.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/usage-control	*ghcr.io/sifis-home/usage-control-engine:latest*

Policy Translation Point

The Policy Translation Point (PTP) translates high-level security policies into low-level policies in a XACML formalism. Furthermore, PTP detects potential conflicts like redundancies and inconsistencies between high-level policies.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/policy-translation-point	*ghcr.io/sifis-home/policy-translation-point:latest*

Privacy dashboard
The privacy dashboard is a centralized web interface for data controllers and data subjects that provides an organized overview of privacy notices, a list of data subjects, data retention periods, tracking of data subject rights requests, and facilitates communication between data subjects and data controllers for GDPR compliance. For additional details please refer to D2.6 and D2.7.

GitHub repository	Docker Images URLs
https://github.com/sifis-home/privacydashboard	*ghcr.io/sifis-home/privacydashboard:latest*

5.4.6 Api Gateway

The components being part of the SIFIS-Home Api Gateway are reported in Figure 34. No noticeable differences have to be reported with respect to what has been described in D5.4. Hence, in the rest of this section, for each subcomponent, we only report a brief description and a link to their GitHub repository and Docker images. Additional details of the components can be found in D5.4.

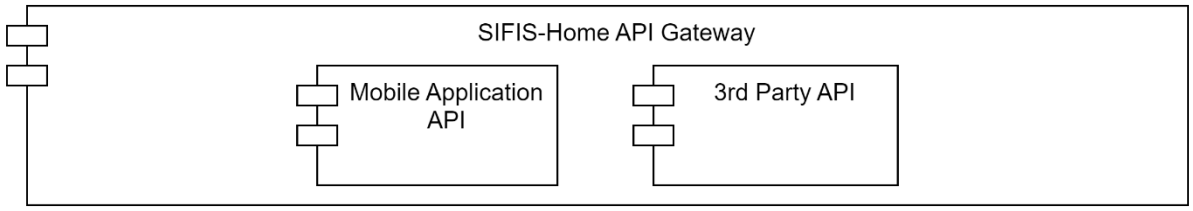


Figure 49: API Gateway with the mobile and 3rd party API

5.4.6.1 Mobile Application API

No changes to report with respect to D5.4.

5.4.6.2 3rd party API

This component provides the API that allows downloaded third-party applications to interact with the Smart Home system.
It is implemented as a rust crate based on “tarpc” to implement both applications (that act as untrusted clients) and runtimes (that act as trusted endpoint and interface to the DHT).
The application binaries are expected to run in a segregated environment such as “ujail” or docker and interact only with the runtime via a Unix socket.

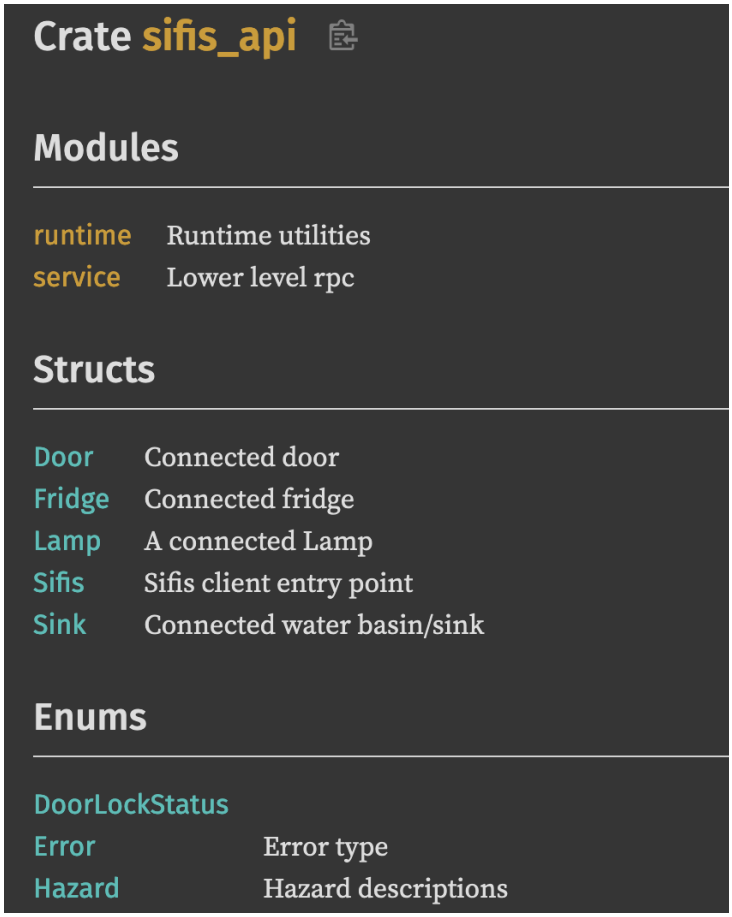


Figure 50: The manifest generator/validator

GitHub repository	Docker Images URLs
https://github.com/sifis-home/sifis-api *https://github.com/sifis-home/sifis-message*	*ghcr.io/sifis-home/sifis-alpine-runtime-arm64v8:latest*

5.4.7 VPN Manager

The VPN Manager is a cluster service that we use to make services running on the SDs accessible from a remote side. The VPN Manager runs only on the leader node and uses a wireguard-client application to connect to a wireguard server running on the Panarea server. As it can be observed from Figure 19-20, the wireguard client is configured in such a way that it connects to the wireguard server running on the Panarea server. Then, the DoMO gateway where it executes takes address 10.43.89.2. Figure 21 shows a successful ping operation from the gateway to the Panarea server. Please note that since i) the VPN Manager is the process opening the VPN connection and ii) Panarea has a public IP address, the solution overcomes NAT or port forwarding issues.

Table 7: VPN manager toml configuration file

```
[domo_vpn]

[domo_vpn.wireguard]

# Wireguard client private key
interface_private_key      = "<secret_client_key>"
```

```
# Address and subnet mask in CIDR notation of the wireguard interface
interface_address        = "10.43.89.2/24"

# Wireguard remote peer (server) public key
peer_public_key          = "<server_public_key>"

# Wireguard remote peer (server) allowed IPs
peer_allowed_ips         = "10.43.89.1/32, 224.0.0.251/32"

# Wireguard remote peer keepalive
peer_keepalive           = 15
```

Table 8: VPN manager wireguard configuration

```
[Interface]
Address    = 10.43.89.2/24
MTU        = 1280
PrivateKey = <client_private_key>

[Peer]
PublicKey          =  <wireguard_server_public_key>
Endpoint           =  panarea.sifis-home.eu:51820
AllowedIPs         =  10.43.89.1/32, 224.0.0.251/32
PersistentKeepalive =  15
```

```
root@domoserver-1:~# ifconfig wg0
wg0       Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.43.89.2  P-t-P:10.43.89.2  Mask:255.255.255.0
          UP POINTOPOINT RUNNING NOARP  MTU:1280  Metric:1
          RX packets:605025 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3267430 errors:2 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:77847324 (74.2 MiB)  TX bytes:4161436772 (3.8 GiB)

root@domoserver-1:~# ping 10.43.89.1
PING 10.43.89.1 (10.43.89.1): 56 data bytes
64 bytes from 10.43.89.1: seq=0 ttl=64 time=0.647 ms
64 bytes from 10.43.89.1: seq=1 ttl=64 time=0.685 ms
64 bytes from 10.43.89.1: seq=2 ttl=64 time=0.723 ms
64 bytes from 10.43.89.1: seq=3 ttl=64 time=0.688 ms
64 bytes from 10.43.89.1: seq=4 ttl=64 time=1.101 ms
```

Figure 51: domo-vpn client successful ping towards Panarea

5.4.8 DHT Manager

5.4.8.1 *DHT*

The SIFIS-Home DHT is a component that offers a completely distributed publish/subscribe mechanism through which SIFIS-Home applications can exchange messages. The SIFIS-Home DHT allows to publish both "persistent" and "volatile" messages. Persistent messages are messages that need to be stored in a persistent way, so that they are available even after a node reboot operation. In detail, persistent messages are stored on an Sqlite database. Volatile messages are instead messages that need to be delivered to all the available applications but that do not need to be persisted on disk.

The SIFIS-Home DHT has a built-in mechanism to solve possible data conflicts that can arise

when a network partition occurs. In detail, every time a message is published on the DHT, the DHT also stores its publication timestamp. Then, the publication timestamp is used to assure that only the most recently published messages will be stored and made available to the applications.

The SIFIS-Home DHT has been developed using the Rust language. Rust applications can include the DHT by embedding it as a library. Non-Rust applications can access the DHT by means of a REST + WebSocket API provided by the DHT Manager. Please note that Rust applications can also use the REST + WebSocket API provided by the DHT Manager to access the DHT. In the next section, we report the details of the REST and WebSocket API provided by the DHT Manager.

SIFIS-Home DHT REST API

The DHT provides a REST API through which it is possible for an external application to access the DHT. Here we report the main REST API endpoints. In the following, <DHT_ADDRESS> indicates the IP address of the node where the DHT executes while <DHT_PORT> is the HTTP port used by the DHT.

HTTP Method	**Endpoint**	**Parameters**	**Description**
GET	http://<DHT_ADDRESS>:<DHT_PORT>/get_all	-	Returns all the published persistent messages
GET	http://<DHT_ADDRESS>:<DHT_PORT>/topic_name/<topic_name>	<topic_name>	Returns all the persistent messages whose topic_name is <topic_name>
GET	http://<DHT_ADDRESS>:<DHT_PORT>/topic_name/<topic_name>/topic_uuid/<topic_uuid>	<topic_name> <topic_uuid>	Returns the message whose topic_name is <topic_name> and topic_uuid is <topic_uuid>
POST	http://<DHT_ADDRESS>:<DHT_PORT>/pub	The content of the message to be published is specified in the request body (type application/json)	Publishes a volatile message whose content is specified in the payload of the request
POST	http://<DHT_ADDRESS>:<DHT_PORT>/topic_name/<topic_name>/topic_uuid/<topic_uuid>	<topic_name>, <topic_uuid>, The content of the message to be published is specified in the request body (type application/json)	Publishes a persistent message whose topic_name is <topic_name> and whose topic_uuid is <topic_uuid>

SIFIS-Home DHT WebSocket API

The DHT provides also a WebSocket API to access the DHT. In the following, we report the WebSocket messages that can be sent to the DHT Manager to request operations on the DHT. The websocket API URL is ws://<DHT_ADDRESS>:<DHT_PORT>/ws.

Message	Parameters	Description
RequestGetAll	-	Returns all the published persistent messages
{"RequestGetTopicName": {"topic_name": "<topic_name>"}}	<topic_name>	Returns all the persistent messages whose topic_name is <topic_name>
{"RequestPubMessage": <payload>}	<payload>: payload of the message to be published	Publishes a volatile message
{“RequestPostTopicUUID”: { “topic_name”: <topic_name>, “topic_uuid:”: <topic_uuid>, “value”: <payload> }}	<topic_name>, <topic_uuid>, <payload>	Publishes a persistent message whose topic_name is <topic_name> and whose topic_uuid is <topic_uuid>

SIFIS-Home DHT code and deployment
The SIFIS-Home DHT source code and Docker images are available at:

GitHub repository	Docker Images URLs
https://github.com/sifis-home/libp2p-rust-dht	*ghcr.io/sifis-home/sifis-alpine-dht-arm64v8:latest* *ghcr.io/sifis-home/sifis-alpine-dht-amd64:latest*

The command line parameters that are available are:

***SQLITE_FILE*:** absolute path of the sqlite file where persistent messages published on the DHT are stored.

***PRIVATE_KEY_FILE*:** path to the file containing the private key of the node in PEM format. A 2048 bits long private RSA key file in PEM format can be generated using command "openssl genrsa -out private.pem 2048". If *private_key_file* does not exist, the key pair is automatically generated by sifis-dht and stored inside file *private_key_file* .

***IS_PERSISTENT_CACHE*:** if set to true indicates that sifis-dht is authorized to write messages to the provided sqlite file. If set to false, the *SQLITE_FILE* content will only be used to initialize the cache.

***SHARED_KEY*:** 32 bytes long shared symmetric key in hex format (command "openssl rand -hex 32" can be used to generate a random key)

***HTTP_PORT*:** port to be used for the HTTP interface

***LOOPBACK_ONLY*:** if set to true, only the loopback interface will be used, meaning that only other local instances of sifis-dht are discovered. If set to false, all the available network interfaces of the device will be used. Hence, two sifis-dht instances running on the same local network should discover each other.

5.4.8.2 *FIWARE API*

The FIWARE Api component is a cluster service that only runs on the leader device of a SIFIS-Home network. This component forwards the persistent messages published through the DHT to the Ratatosk FIWARE Context broker that is part of the Yggio instance residing on the SIFIS-Home cloud. Also, it forwards commands entered by the user in the Yggio user interface to the DHT (see Figure 22). In detail, from the Yggio UI and thanks to the FIWARE API component it is possible to control devices, e.g., turn on and off lamps, as well as require the installation of third-party applications. The Yggio UI can be accessed from the user both when it is at home or when it is on a remote side.

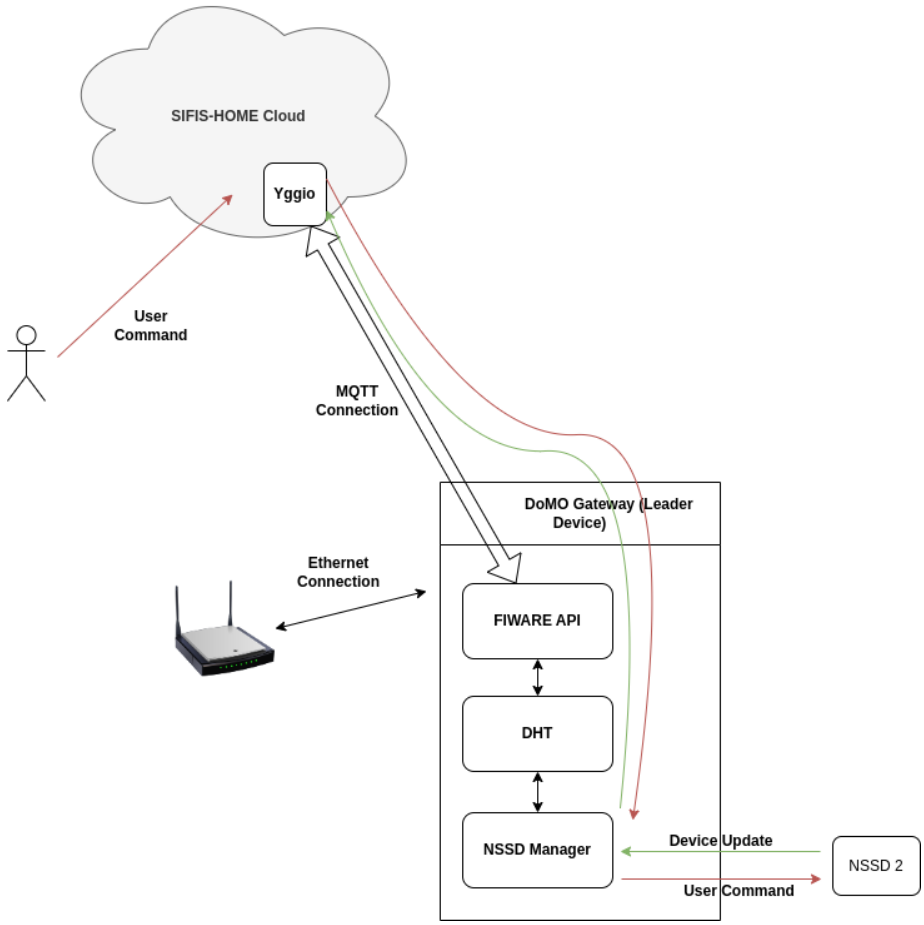


Figure 52: Unsuccessful DHT request

The Fiware API component uses both REST API and the MQTT protocol to set up and receive/publish messages from/to Yggio. More in detail, the FIWARE API component is provided with a set of dedicated credentials that allow it to access the Yggio REST API and MQTT broker. It then uses the REST API to associate and reserve a dedicated MQTT topic for each DHT topic. Please note that the solution allows to overcome NAT/firewall issues since the Yggio instance is provided with a public IP address and the connection is initiated by the FIWARE API connection.

The FIWARE API component has been developed using the Rust language. Its source code and Docker image can be found at:

GitHub repository	Docker Images URLs
https://github.com/sifis-home/dht-to-mqtt	*ghcr.io/sifis-home/sifis-dht-to-mqtt-amd64:latest* *ghcr.io/sifis-home/sifis-dht-to-mqtt-arm64v8:latest*

5.4.9 NSSD Manager

The SIFIS-Home NSSD Manager is the SIFIS-Home component responsible for interacting with the NSSDs present in the house. It is composed of two different modules namely the CoAP Manager and WoT Manager.

5.4.9.1 *CoAP Manager*

On a device acting as CoAP client, the CoAP Manager receives commands and retrieves information from the DHT Manager, and then takes care to execute the requested operations, by interacting with the targeted CoAP server device(s).

GitHub repository	Docker Images URLs
https://github.com/sifis-home/wp3-solutions	*ghcr.io/sifis-home/phase0-client:latest* *ghcr.io/sifis-home/phase1-client:latest* *ghcr.io/sifis-home/phase2-client:latest* *ghcr.io/sifis-home/phase3-client:latest* *ghcr.io/sifis-home/phase4-client:latest* *ghcr.io/sifis-home/group-client1:latest* *ghcr.io/sifis-home/group-client2:latest*

5.4.9.2 WoT Manager

The WoT Manager has been developed using the Rust language and is composed of three main modules: the DHT module, the M-DNS Module and the Web of Things (WoT) Module.

- **DHT Module:** the DHT Module is the responsible for communicating with the DHT Manager. Being a native Rust application, the WoT Manager embeds the DHT as an external library. The WoT Manager communicates with the DHT Manager in order to receive commands from the user (e.g., "turn on a certain light") and to update the status of the managed devices (e.g., to signal that an actuator is connected to the system).
- **M-DNS Module:** the M-DNS Module uses the m-DNS protocol to detect the presence of WiFi actuators in the network advertised by the DoMO gateway where it is in execution. In detail, the M-DNS module periodically performs an m-DNS discovery operation that produces as a result the list of WiFi actuators that are connected to the DoMO gateway advertised network.
- **WoT Module:** the Web of Things module manages the communication of the WoT Manager with the NSSD. It uses a WoT API to interact with the NSSD.

The interaction between the WoT Manager and both the DHT Manager and NSSDs is shown in Figure 53 while the operations continuously performed by the WoT Manager are reported in Figure 54.

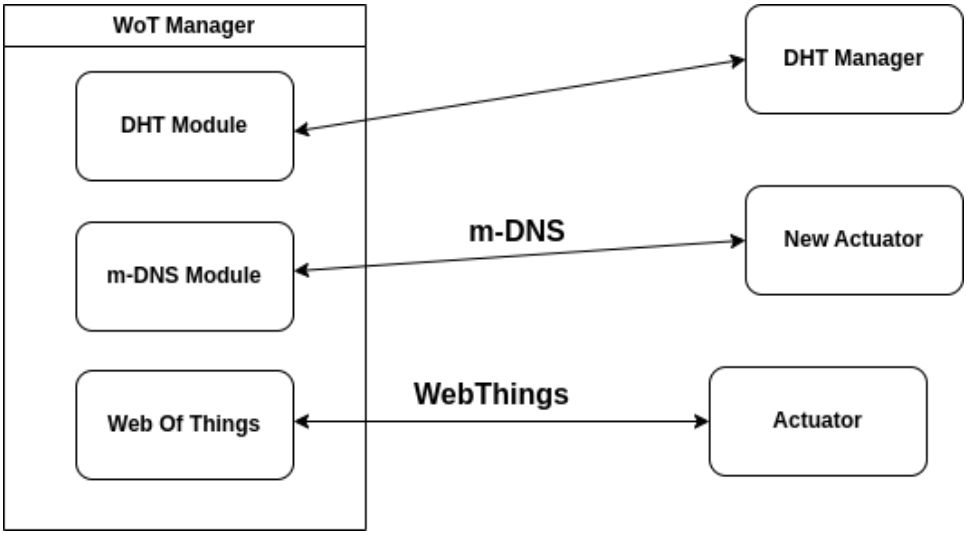


Figure 53: WoT Manager interaction with the DHT Manager and WiFi actuators (NSSD)

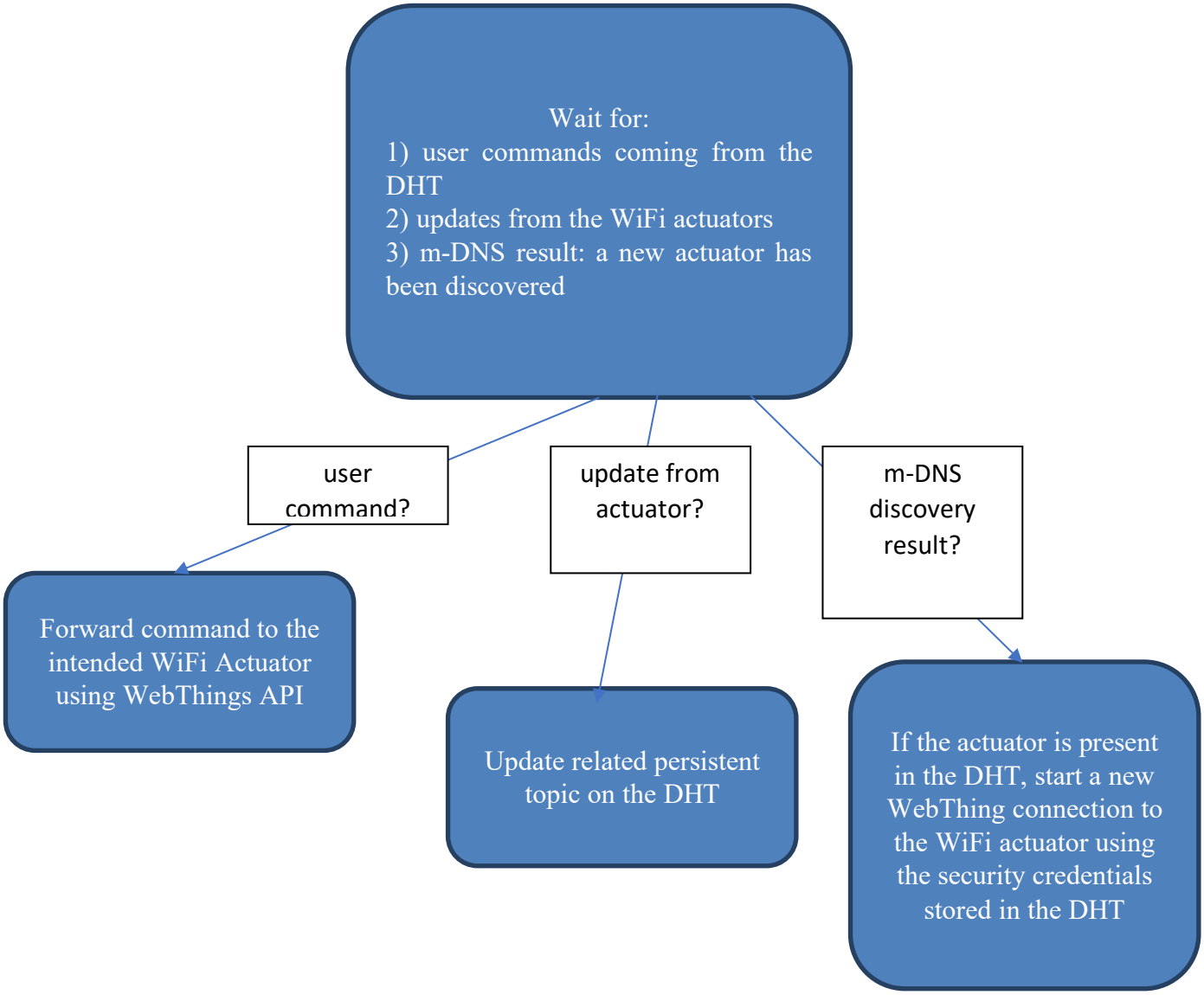


Figure 54: WoT Manager operations

As it can be observed, the WoT Manager continuously waits for an event to occur. The events can be of three different types: i) a user command is received from the DHT, ii) an update from one of the WiFi actuators is received, iii) a new WiFi actuator connected to the network advertised by the DoMO gateway. In case a user command is received from the DHT, it is forwarded to the intended WiFi actuator using the WoT API offered by the WoT firmware installed on the WiFi actuators. Conversely, if a state update is received from one of the actuators, the related persistent topic on the DHT is updated. Finally, if a new WiFi actuator has been discovered by the M-DNS module and the actuator has been registered on the DHT, a new WoT connection towards it is started using the security credentials stored in the DHT.

The WoT Manager uses both persistent and volatile topics. In detail, we use persistent topics to store the status of the various actuators inside the DHT, so that it is accessible by all the SIFIS-Home applications. Conversely, we make use of volatile messages to send/receive user commands.

topic_name	**topic_uuid**	**Description**
shelly_1	Random UUID	Topic used to store the status of a Shelly 1 Device.
shelly_1pm	Random UUID	Topic used to store the status of a Shelly 1pm Device.
shelly_25	Random UUID	Topic used to store the status of a Shelly 2.5 Device
shelly_dimmer	Random UUID	Topic used to store the status of a Shelly Dimmer Device
shelly_em	Random UUID	Topic used to store the status of a Shelly EM Device
shelly_rgbw	Random UUID	Topic used to store the status of a Shelly RGBW Device
shelly_1plus	Random UUID	Topic used to store the status of a Shelly 1 Plus Device
domo_light	Random UUID	Topic used to represent a physical light
domo_bistable_button	Random UUID	Topic used to represent a physical bistable button
domo_button	Random UUID	Topic used to represent a physical button
domo_window_sensor	Random UUID	Topic used to represent a window contact sensor
domo_door_sensor	Random UUID	Topic used to represent a door contact sensor
domo_pir_sensor	Random UUID	Topic used to represent a motion sensor
domo_ble_thermometer	Random UUID	Topic used to store temperature and humidity measurements produced by the domo temperature/humidity sensors.

Below we also report the structure of the different volatile messages used by the WoT Manager to

receive user commands for the actuators and the physical lights.

Volatile message format used to control the actuators

```
{
  "command": {
    "command_type": "shelly_actuator_command",
    "value": {
      "mac_address": <actuator_mac_address>,
      "action_name": <action_name>,
      "action_payload": <action_payload>
    }
  }
}
```

As it can be observed, the actuator to which send the command can be specified using parameter *mac_address*. The specific action to perform is identified by parameters *action_name* and *action_payload*.

Volatile message format used to control lights

```
{
  "command": {
    "command_type": "turn_command",
    "value": {
      "topic_name": “domo_light”,
      "topic_uuid": <topic_uuid_of_the_light_to_control>,
      "desired_state": True/False
    }
  }
}
```

As it can be observed, *command type* indicates that we want to send a turn command. The light to control is specified by proving its uuid in the *topic_uuid* field. Finally, the desired state for the light is indicated by parameter *desired_state*.

SIFIS-Home WoT Manager code and deployment
The WoT Device Manager source code and Docker images are available at:

GitHub repository	Docker Images URLs
https://github.com/sifis-home/domo-wot-bridge	*ghcr.io/sifis-home/domo-wot-bridge-amd64:latest* *ghcr.io/sifis-home/domo-wot-bridge-arm64v8:latest*

Please note that an instance of the WoT Manager is present on every DoMO Gateway.

5.4.10 Riots WoT integration

The integration of the Riots devices to Sifis-Home architecture was implemented as part of the project. Riots devices are third party devices that use proprietary wireless communication protocol, and the implementation is done in smart device. Figure below shows the modules that are involved in the implementation.

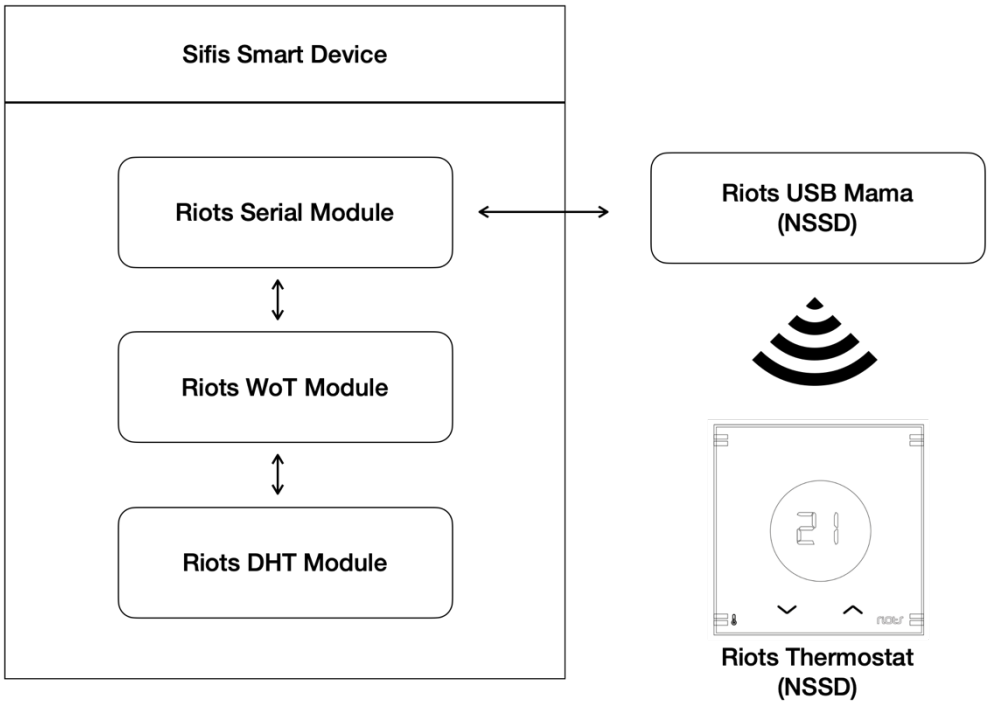


Figure 55: RIOTS WoT implementation

Riots Serial Module is a software component that acts as an integration link between Riots devices and the SIFIS-Home architecture.
Riots WoT Module collects the data from Riots Serial module and provides the abstraction of Riots devices as Web of Things compliant devices.
Riots DHT Module publishes the device data collected from Riots WoT Module to DHT.

The implementation uses following persistent DHT topic to store the model and data of Riots Thermostat so it is accessible by all the SIFIS-Home applications.

topic_name	topic_uuid	Description
SIFIS::RiotsThermostat	Random UUID	Topic used to store the status of a Riots Thermostat Device.

Example of a persistent message format used to provide thermostat data:

```
{
        "topic_name": "SIFIS::RiotsThermostat",
        "topic_uuid": "FirstRiotsThermostat",
        "value": {
                "set_temperature": 20,
                "temperature": 24.7,
                "humidity": 49,
                "status": 0,
```

```
            "name": "Riots Thermostat 1"
        }
},
```

Riots WoT integation code and deployment
The source code and Docker images are available at:

GitHub repository	Docker Images URLs
https://github.com/sifis-home/riots-webthings	*ghcr.io/sifis-home/sifis-home/riots-usb:latest* *ghcr.io/sifis-home/sifis-home/riots-webthings:latest* *ghcr.io/sifis-home/sifis-home/riots-dht:latest*

Integration details
For demo purposes a piece of hardware was prepared, Riots Thermostat was installed to a piece of plywood in a project box side by side with a indication light that is used to show the status of heating as seen in the figure below.

Figure 56: Riots WoT pilot device

The collected thermostat data was delivered to DHT as described in previous chapters and visualized in the Sifis-Home mobile app as shown in figure below.

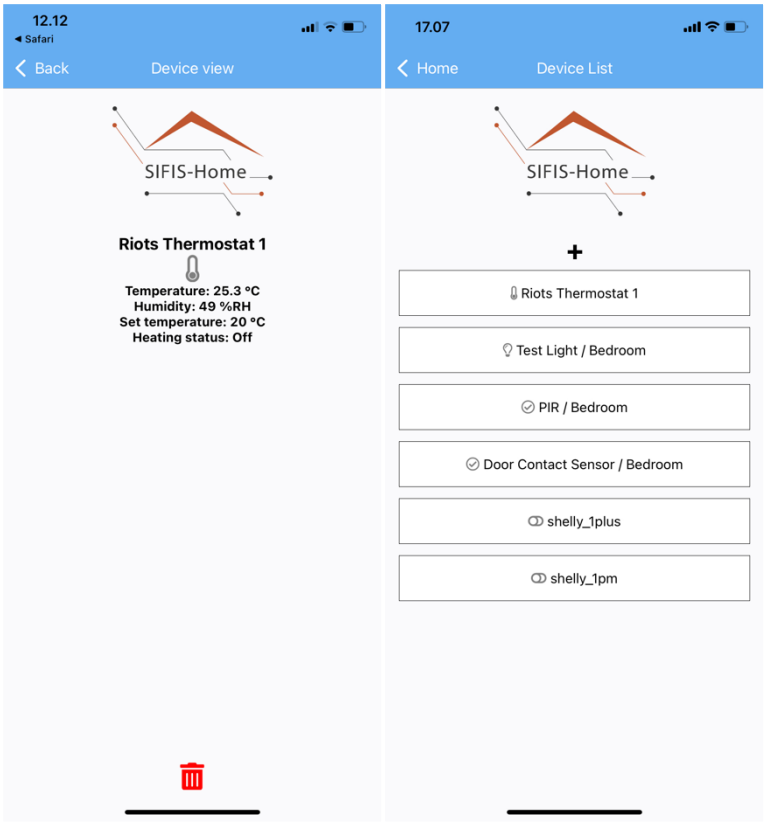


Figure 57: Riots Thermostat device shown in Sifis-Home mobile app

5.5 *NSSD Framework*

5.5.1 DoMO WiFi actuators firmware

We decided that the NSSDs, being part of the pilot, need to expose a Web of Things compliant API. In detail, in a WoT-based architecture every NSSD is a server that exposes a set of functionalities to possible clients. Web of Things does not mandate the use of a specific protocol to make the functionalities of a Web of Thing Object (WebThing) accessible to an external application. In our implementation, we decided that our NSSDs are HTTPS servers exposing their functionalities through a WebSocket API. In a WoT server, *properties* are used to expose settings and characteristics of a WebThing. For example, we can have the property *description* that is a textual description of a certain WebThing (e.g., "kitchen light"). In addition, *actions* are used to request the execution of a certain operation to a WebThing. A possible action to allow a user to turn on and off a certain light can be, for example, *turn.* Web of Things also provides *events* to allow a WebThing to signal anomalous conditions. Our implementation uses WoT properties and actions only.

5.5.1.1 *Firmware implementation and structure*

The WoT firmware for the NSSD devices has been developed using the C++ language and the Arduino ESP8266 Framework. Its code is available on GitHub (https://github.com/sifis-home/domo-wot-actuator). We also used PlatformIO (https://platformio.org/) to simplify the firmware development and building processes. All the different WiFi actuators that we use in the pilot share the same code base. This allows to reduce code repetition and speeds up testing operations.

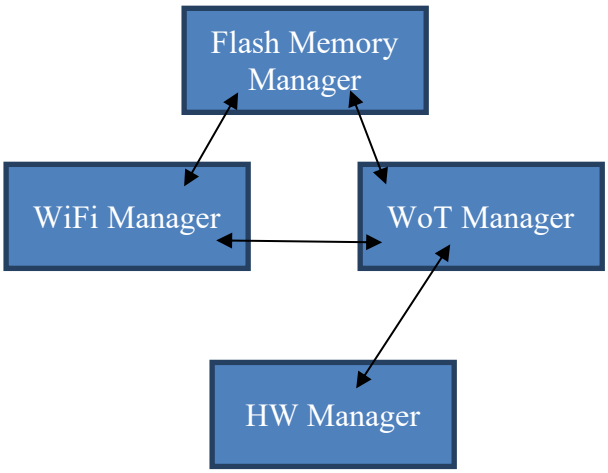


Figure 58: WoT firmware modules

In detail, our WoT firmware is composed of 4 different modules (Figure 17):

- **WiFi Manager:** it is the module responsible for managing the WiFi connection of the actuator. It communicates with the Flash Memory Manager to get the WiFi SSID and Password of the network to which the WiFi actuators should connect to. Also, it signals to the WoT Manager events of connection to/disconnection from the WiFi network. The default WiFi network to be used is specified in the firmware code.
- **Flash Memory Manager:** it is the module that is responsible for reading/writing data from/to the persistent memory of the actuator. It provides WiFi credentials to the WiFi Manager. Also, it provides the WoT Manager with the server certificate and credentials to be used by the HTTPS WebSocket server needed to expose the WoT API (see below for additional details).
- **HW Manager:** it is the module responsible for managing the physical peripherals/devices of the actuators. It uses the ESP8266 GPIO pins to activate/deactivate the actuator relays and to get the current status of attached input devices such as bistable buttons. Also, it communicates with the energy monitoring chips to provide power/energy readings to the user.
- **WoT Manager:** it is the module responsible for creating a WoT compliant API for the WiFi actuators. In particular, the WoT Manager main task is to start and monitor an HTTPS WebServer with WebSocket support. Also, the WoT Manager is responsible for starting up an m-DNS resolver that allows the discovery of the actuator by part of the NSSD Manager. In our implementation, every actuator is identified by the m-DNS name *<shelly_model>-<mac_address>.local*, where shelly_model identifies the particular actuator type (i.e., shelly1, shelly_1pm, etc.) and mac_address is the MAC address of the actuator. The WoT manager is informed about network connection/disconnection events from the WiFi Manager. Also, it receives the security material needed to correctly start up the HTTPS Web Server from the Flash Memory Manager (see section Security for additional details). Finally, it communicates with the HW Manager module to activate/deactivate the physical relays, get energy/power readings, and update on the input channels states.

5.5.1.2 WoT API: properties and actions

As mentioned before, our WoT implementation uses both WoT *properties* and *actions.* They are detailed below.

Properties

We use a single property named *status,* of type String, to represent the current state of the actuator. In detail, the property *status* is the serialization of a JSON Object that contains a number of different fields. We report in Figure 18 a possible value for the *status* property for a shelly1 actuator and a description of the various fields.

Table 9: Example of WoT status property


```
{
"ap_mac_address":"9483c413a0d4",
"fw_version":"v1",
"gateway":"192.168.1.1",
"input1":false,
"ip_address":"192.168.1.26",
"mac_address":"98:cd:ac:2d:4c:35",
"mcu_temperature":94.01399994,
"mode":0,
"output1":false,
"rssi":-61,
"topic_name":"shelly_1",
"wifi_ssid":"****"
}
```


Field name	Description
ap_mac_address	MAC address of the WiFi Access Point to which the actuator is currently connected to
fw_version	Firmware version
gateway	IP address of the WiFi actuator gateway
input1	State of input channel 1
ip_address	IP address of the WiFi actuator
mac_address	MAC address of the actuator
mcu_temperature	temperature of the MCU
mode	Current operation mode
output1	State of output channel 1
rssi	RSSI signal level
topic_name	topic_name of the persistent message used to store the status of the actuator inside the DHT
wifi_ssid	SSID of the WiFi network to which the actuator is connected to

Please note that the various fields of the status property are updated over time. For example, in case the relay number 1 of the actuator is activated, the output1 field value changes from false to true. A WebSocket client connected to the actuator receives a *PropertyStatusUpdate* message whenever the status property value changes.

Action

We use an action named *shelly_action,* of type Object, to allow an external application to request the execution of specific operations to the actuators. The shelly_action contains two mandatory fields: *action_name* and *action_payload*. The former is used to identify the specific type of action that must be executed by the actuator. The latter contains parameters for the action execution. Our implementation

currently provides the actions reported in the table below.

action_name	action_payload	Description
set_output	output_number: number of the relay to be activated/deactivated desired_state: desired state of the relay	Action that allows to activate/deactivate output relays.
set_dimmer	dim_value: desired dimming level	Action that allows to request a dimming operation.
pulse_action	output_number: relay to use for the pulse operation duration: duration of the pulse signal in ms	Action that allows to request a pulse operation using an output relay.
set_shutter	desired_state: OPEN, CLOSED, STOPPED	Action that allows to open/close/stop a roller shutter.
set_rgbw	rgbw_value: desired rgbw value	Action that allows to set RGBW values.
set_led_dimmer	output_number: output channel to be used, dim_value: desired dimming level	Action that allows changing the dimming values of LED lights.
change_wifi	wifi_ssid, wifi_password	Action that allows changing the WiFi network to which the actuator should connect to.
change_mode	mode	Action that allows changing the actuator operation mode (i.e., RELAY mode or SHUTTER mode).
update_action	fw_url	Action that allows updating the firmware of the actuator.

A WebSocket client can request the execution of a specific action by sending a specific *ActionRequest* message.

5.5.1.3 Security

As mentioned above, the WoT Manager module takes care to expose a Web of Things compliant API that can be used by external applications to access the functionalities of the WiFi actuators. Communication between WebSocket clients and the WoT-enabled actuator are encrypted and protected. In particular, only allowed users/applications are able to communicate with the actuator and request the execution of specific actions. To this end, the WoT manager uses an HTTPS server with WebSocket support. The server certificate and server key to be used are generated during the actuator flashing phase and stored on the flash memory of the actuator (see section below). In addition, every WebSocket client should provide a user/password pair in order to access the WebSocket server functionalities. In detail, every WebSocket client should use the HTTP basic access authentication to send its username and password when making a request to the actuator. In basic HTTP authentication, the request contains a header field in the form of Authorization: Basic <credentials>, where credentials is the Base64 encoding of the username and password joined by a single colon. Our implementation uses a dedicated user/password pair for every actuator. They are generated and stored on the flash memory of the actuator

during the actuator flashing phase. In this way, we provide encrypted communication and can guarantee that only allowed applications have access to the WiFi actuators functionalities.

5.5.1.4 Flashing procedure

The flashing procedure is the operation through which we install the WoT firmware on our WiFi actuators and provide them with the needed security material. We developed a flashing tool to ease the actuator flashing operation. Before starting the flashing procedure, the actuators should be put in programming mode and connected to a PC where the flashing tool is executed.

The flashing tool follows a number of steps that are detailed in the following:

1) The user selects the model (*actuator_model*) of the actuator to be flashed (i.e., shelly1, shelly1pm, etc.). The corresponding firmware is selected.
2) The MAC address (*mac_address*) of the actuator to be flashed is retrieved.
3) The security material to be used by the actuator is generated. In detail, a random user/password pair is generated. Also, a server private/public key pair and a server certificate with CN field equal to "*actuator_model-mac_address.*local" is created. The server certificate is signed using the SIFIS-Home Certification Authority key. Please note that the SIFIS-Home Certification Authority certificate is contained in the SIFIS-Home OpenWrt distribution used by the DoMO Gateways. It is stored in the trust store of the DoMO gateways by installing a dedicated OpenWrt package that we created. Please note that we are currently assuming that the device/PC used to flash the WiFi actuators is trusted. In detail, we are assuming that that the CA private key used to sign the actuator certificates has been provided and saved on the flashing device/PC using a secure channel. In a production environment, the use of an intermediate CA that is only used to sign the actuator certificates is recommended.
4) The flash memory of the actuators is completely erased.
5) The security material (serverKey, serverCert, user/password) is stored on the flash memory of the actuator (SPIFFS partition).
6) The WoT firmware is installed.

The user/password pair and the MAC address of the flashed actuator are saved in a local text file. Please note that the NSSD Manager needs to know the user/password pair used by a specific actuator for being able to connect to it. Hence, they need to be inserted in the DHT of the SIFIS-Home house where the actuator will be installed. Currently, the credentials are stored inside the DHT using a Web-based control panel that we created (see below).

5.5.1.5 WoT firmware operations

The following figure reports the operations that are performed by our WoT-enabled actuators. When the actuator is turned on, the Flash Memory and the HW peripherals are initialized. Then, the WiFi Manager obtains the WiFi credentials from the Flash Memory Manager. Then, the WiFi connection is activated and the HTTPS WebSocket WoT server is started by getting the security material from the flash memory. Also, the m-DNS resolver is activated. At this point, Websocket clients, such as the NSSD Manager, can connect to the actuator to get property updates and send action requests.
The actuator continuously waits for i) updates from the physical peripherals that produce an update of the WoT *status* property, ii) requests to execute a specific *shelly_action* by part of a WebSocket client that, in general, cause the WoT Manager to communicate with the Hardware Manager to start operations involving the physical devices.

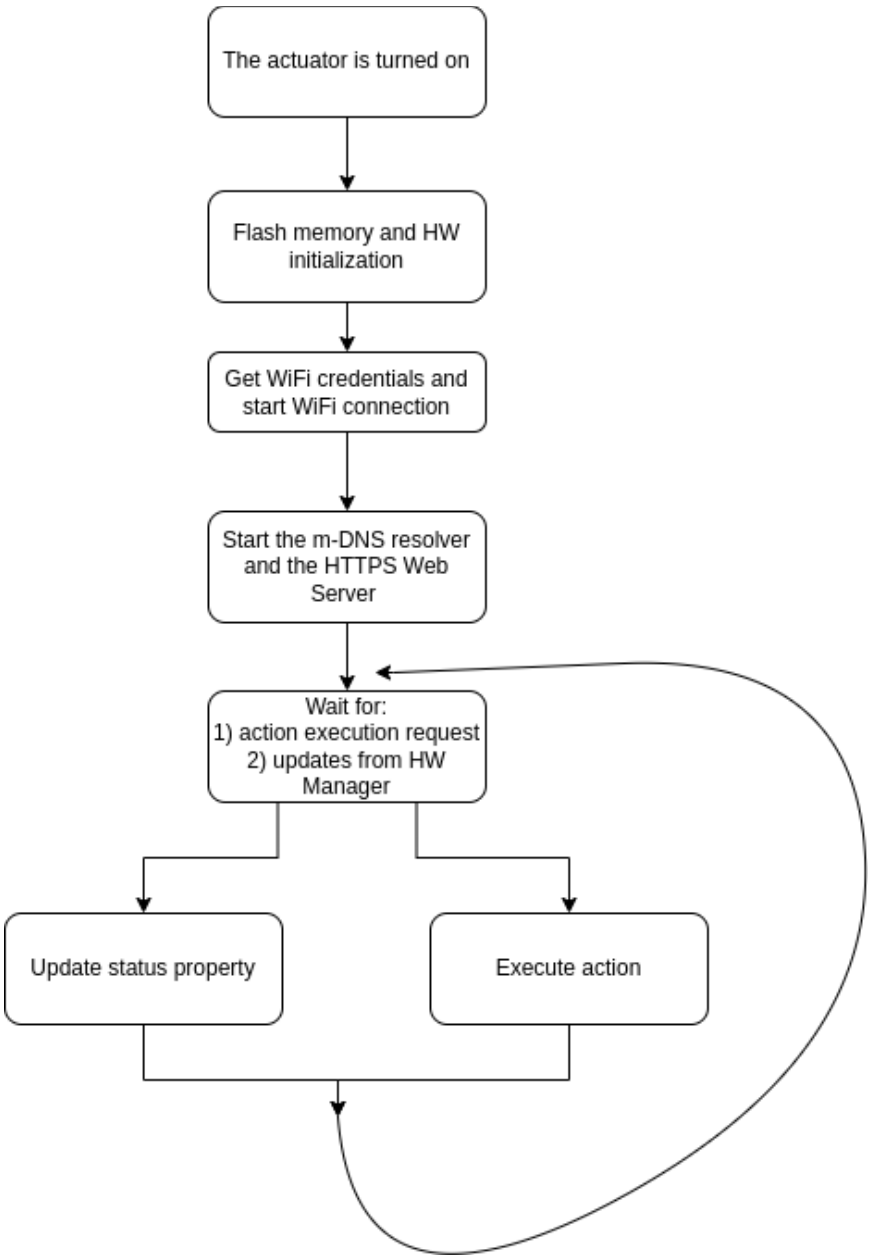


Figure 59: WoT firmware operations

6 Use Cases and Functional Validation

This section describes in detail the setup of the testbed, actors and operations performed on the DOMO infrastructure, to validate the use cases defined in D1.2.

6.1 *Use Case 01 – Login Through Biometrics*

This use case maps the requirement of being able to recognize one of the registered users (tenant) of a SIFIS-Home instance by using biometrics. The identification of the user is necessary to decide on allowed functionalities or to provide customized services. In the pilot validation, we use as biometric the face identification, exploiting the *Face Recognition* analytic from the Data Analysis Toolbox.

To validate the use case, a classifier has been trained with a set of identities from persons working in SIFIS-Home. Other identities have been used for a correct training of the classifier. To demonstrate the

full integration in the DOMO pilot, we have set up a laptop webcam connected to the WiFi network of one of the three DOMO Gateways, which are present in the DOMO testbed. The correct or incorrect identification of a user is notified to the administrator through a notification on the Mobile App component. More in details, the evaluation of the use case involves four actors: two Tenants whose identity is known, one external user whose identity is unknown, and the Administrator who will receive the notification.

The use case is considered as correctly performed if the Administrator receives two notifications with the correct identities of the Tenants and one notification for the non-recognized external user.

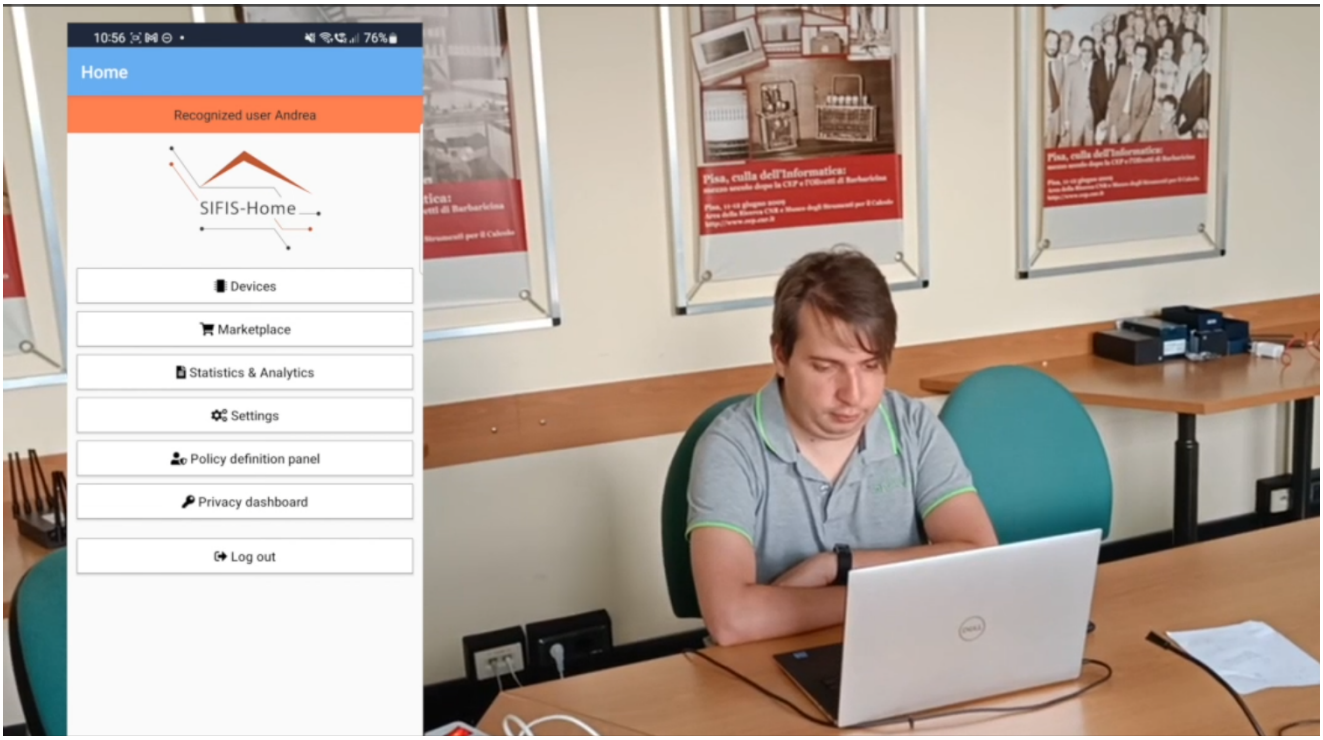


Figure 60: User identification through biometrics

6.2 *Use Case 02 - Operate Through Voice Commands*

In this use case we verify that it is possible to control the SIFIS-Home functionalities by using voice commands. The SIFIS-Home framework has to be able to recognize specific keywords to automatically provide services, filtering all the other speeches that normally happen inside a house. The issued commands will correspond to an actionable action, and they issue a complex integrated workflow, involving the Input Manager, the DHT Manager, the Data Analysis Toolbox, the Policy Enforcement Engine and the NSSD Manager.

To validate the use case, we are using the typical DOMO testbed setup, with three DOMO Gateways and three NSSDs, with commands issued through a microphone connected to a laptop. The actor involved is one smart home Tenant, who will insert a voice command to turn on a light, in the middle of other words. The SIFIS-Home framework will react to the command by turning on the light, while will ignore all other words that do not consist in a command. The performed demo will show the actual functionality with a person working on the project as Tenant, showing at the same time which messages are received by the main components of this workflow.

Figure 61: Voice command recognition

6.3 *Use Case 03 – Person Movement or Presence Notification*

This use case aims at verifying the capability of the SIFIS-Home framework to detect presence of people in a dark environment. The final goal of this functionality could be multi-purpose as it can be used for physical intrusion detection, or to have no need for the tenants looking for the light if waking up at night. The use case exploits again the full DOMO testbed with three DOMO gateways, one PIR (passive infrared) sensor as first NSSD, the mobile device for notification and an additional NSSD to control a lamp.

To validate the use case, we include two actors: a smart home Tenant and the Administrator. The Tenant will move inside a dark room in relative proximity to the PIR sensor. If the presence is recognized, the administrator will receive a notification on the mobile application (intrusion detection), whilst the SIFIS-Home framework will react by turning on a light.

The use case is considered as correctly validated if these two actions are triggered when the person is moving inside the room, without generating false positives or negatives, triggering the notification and the light. The demo will be performed by showing the physical interaction with the PIR sensor.

6.4 *Use Case 04 – Notification About Software Intrusion*

This use case aims at verifying the capability of the SIFIS-Home framework to detect possible malicious applications running on a SD and take appropriate countermeasures. The use case exploits the DOMO testbed using three DOMO gateways and the mobile device for notification.

To validate the use case, we intentionally start up a malicious application that publishes a considerable amount of messages on the DHT on one of the DOMO gateways. The anomalous application behaviour should be detected by the DHT monitor analytic and reported to the System Protection Manager. The

System Protection Manager then generates a notification that is received by the Administrator of the smart home on his mobile device. The use case is considered successfully validated if the malicious application behaviour is detected by the SIFIS-Home framework and a notification is successfully received on the mobile application.

6.5 *Use Case 05 – Register Device*

This use case aims at verifying that the SIFIS-Home framework allows the smart home Administrator to register a device at any time. The use case exploits the three DOMO gateways, a DOMO Shelly 1PM WiFi actuator, that is connected to a physical light, and the mobile device.

To validate the use case, we perform the following actions. The smart home Administrator uses the Devices panel provided by the Mobile Application to insert a new Shelly 1PM actuator device into the system. The Administrator should be asked to provide all the details of the device such as its ID, MAC Address, and the connection credentials to be used to connect to it. The Mobile Application then publishes the information of the device on the DHT, making it available to all the NSSD Managers running on the different SDs. As soon as an NSSD Manager detects the presence of the registered device in its local WiFi Network, a connection with the device is established, to start controlling it.

The use case is considered successfully validated if the registration procedure is performed with no issues and if it is possible to control the device using the Mobile Application some time after it has been included into the system.

6.6 *Use Case 06 – Unregister Device*

This use case aims at verifying that the SIFIS-Home framework allows the smart home Administrator to unregister a device from the system at any time. The use case exploits the three DOMO gateways, a DOMO Shelly 1PM WiFi actuator, that is connected to a physical light, and the mobile device.

To validate the use case, we perform the following actions. The smart home Administrator uses the Devices panel provided by the Mobile Application to get the list of devices that have been installed and registered into the system. The Administrator should be able to select the Shelly 1PM actuator from the list of available devices and the Mobile Application should show a page with detailed information about the selected device. From such a page, it should be possible to unregister the device from the system. After the device has been unregistered, it should not be possible to control it anymore.

The use case is considered as successfully validated if the procedure to unregister the device is performed with no issues and if it is not possible to control anymore the device after it has been unregistered.

6.7 *Use Case 07 –Configure Device*

This use case aims at verifying that the SIFIS-Home framework allows the smart home Administrator to change the configuration details of a certain device at any time. The use case exploits the three DOMO gateways, a DOMO Shelly 1PM WiFi actuator, that is connected to a physical light, and the mobile device.

To validate the use case, we perform the following actions. The smart home Administrator uses the Devices panel provided by the Mobile Application to get the list of devices that have been installed and registered into the system. The Administrator should be able to select the Shelly 1PM actuator from the list of the available devices and the Mobile Application should show a pagewith detailed information about the selected device. From such a page, it should be possible to change the configuration

parameters of the device. The new configuration parameters should be successfully saved on the DHT and made available to all the SIFIS-Home services.

The use case is considered as successfully validated if the procedure to change the device parameters is performed with no issues and if the new configuration is correctly stored on the DHT.

6.8 *UC 08 – Install third-party applications*

This use case aims at verifying the capability of the SIFIS-Home framework to allow the smart home Administrator to install a third-party application on its SDs at any time. The use case employs the three DOMO Gateways and the mobile device.

To validate the use case, we perform the following actions. The smart home Administrator accesses the Marketplace panel of the Mobile Application. The panel should report the list of third-party applications that can be potentially installed on a SD as well as the applications that have beeninstalled already. For every available third-party application, relevant metrics such as its software quality and the list of hazards should be reported. Then, the Mobile Application UI should allow the user to request the installation of the application. Finally, a notification reporting the result of the installation operation should be sent by the system.

The use case is considered successfully validated if the installation procedure can be performed with no issues, i.e., the third-party application is downloaded on the SD and an installation notification is reported on the mobile device.

6.9 *UC 09 – Parental control*

This use case aims at showing that the SIFIS-Home framework allows the smart home Administrator to create new policies at any time. The use case employs the three DOMO gateways and the mobile device.

To validate the use case, we perform the following actions. The Administrator opens the Policy Panel on the Mobile Application. The Policy Panel should report the list of Policies that have been defined and inserted into the system in the past. Then, a clear UI should allow the Administrator to easily define new policies specifying the affected users, the involved devices and, possibly, the days/hours when the policy should be considered active. In detail, we are going to define a policy that does not allow turning on the appliances of a certain room when there are children inside. This use case affects the Policy Translation Point, the DHT Manager and the Usage Control Engine components.

The use case is considered successfully validated if the Administrator successfully creates the new policy and if the policy, translated into XACML format, is correctly received by the Usage Control Engine.

6.10 *UC 10 – Configure User Settings*

This use case extends and specifies the former use case by demonstrating how the SIFIS-Home framework enables the user (tenant or administrator) to remotely configure policies for specific Smart Home users. While the SIFIS-Home framework is able to work also in lack of an Internet connection, the presence of the cloud components allows for remote control and configuration of the smart home. In this specific use case, we consider as actor the Administrator or the Maintainer, operating on an instance of SIFIS-Home from outside of the smart home premises.
The demonstration is performed by exploiting the SIFIS-Home Mobile Application, on a device with disabled Wi-Fi, which is thus not connected to the other Smart Devices and without direct access to the

DHT. The Administrator uses the mobile app to set up a policy, using the Mobile Application, which provides the possibility to select the policy subject, i.e., the entity the policy refers to. The demonstration also shows, as for the previous use case, the message exchange with the policy enforcement engine, to demonstrate the live interaction with the DHT from outside of the smart home premises.

6.11 *UC 11 – Control statistics and analytics*

This use case shows the capability of the SIFIS-Home framework to provide the users with the list of commands and configurations that have been sent to the devices of the smart home over time. Such use case employs one of the DOMO gateways present in the testbed, a Shelly 1 PM WiFi actuator and a physical light. Also, we use a Laptop to access the Yggio web interface and the mobile device to access the logs panel.

To validate the use case, the following actions are performed. First, the Tenant of the smart home opens the Devices Panel of the Yggio Cloud UI and selects the testbed light from the list. The Yggio UI should show a control interface that allows sending on/off commands to the light. Then, the Statistics & Analytics panel of the mobile application should be open. The panel should report the list of logs that have been generated by the smart home devices and the SIFIS-Home components. The Tenant turns on and off the testbed light several times. Immediately after the commands are executed, a log should appear in the Statistics & Analytics Panel.

The use case is considered successfully validated if logs of the commands sent during the use case testing session are correctly stored and shown in the logs panel of the mobile application.

6.12 *UC 12 – Remote configuration of device*

This use case extends Use Case 07, demonstrating how the SIFIS-Home framework allows reconfiguring a device at any time and even from a remote side. The involved devices are the three DOMO gateways and the mobile device. Moreover, a Laptop connected to the local network where the three DOMO gateways are placed is used.

To demonstrate the use case, the following actions are performed. First, the content of the DHT topic associated with a certain Shelly 1PM actuator is retrieved from a local instance of the DHT (i.e., by using the well-known Postman application we perform an HTTP GET request pointing to the DHT instance running on one of the three DOMO gateways). We then open the mobile application and turn off the WiFi connection, forcing the mobile application to use the VPN tunnel that the house establishes with the Panarea server. We change the parameters of the Shelly 1PM actuator and show that the changes are propagated to the local DHT instances.

The use case is considered successfully validated if the parameters of the actuator are successfully changed using the mobile application despite it is not connected to the local WiFi network.

6.13 *UC 13 – Remote configuration of policies*

This use case extends and generalizes Use Case 08, demonstrating as done with Use Case 10, the possibility to define and edit policies also from outside the smart home premises. The involved actors are the Administrator or the Maintainer, who accesses the SIFIS-Home framework either through the mobile application or through the cloud interface (Yggio), outside of the smart home premises. As for Use Case 10, the validation is performed by disconnecting the device from which the access is performed from the WiFi network, to ensure that it cannot directly communicate with the DOMO gateways. The Administrator will then define a policy through the Mobile Application, which will be translated in XACML and stored in the Policy Enforcement Engine.

6.14 *UC 14 - Remote handling of emergency situations*

This use case demonstrates the capability of the SIFIS-Home framework to timely communicate emergency situations to the Administrator or to the Maintainer. In this specific demonstrator, we are considering an intrusion detection scenario, where one NSSD (door/window contact sensor) is used to notify the unauthorized opening of a physical door. The set up includes three DOMO gateways running the SIFIS-Home framework, one mobile device used by the administrator to receive notification about the status change of the NSSD.
The demonstration is performed by having one person physically operating the door, controlled by the NSSD.

The use case is considered as validated if the Administrator is notified in a short time of the intrusion attempt.

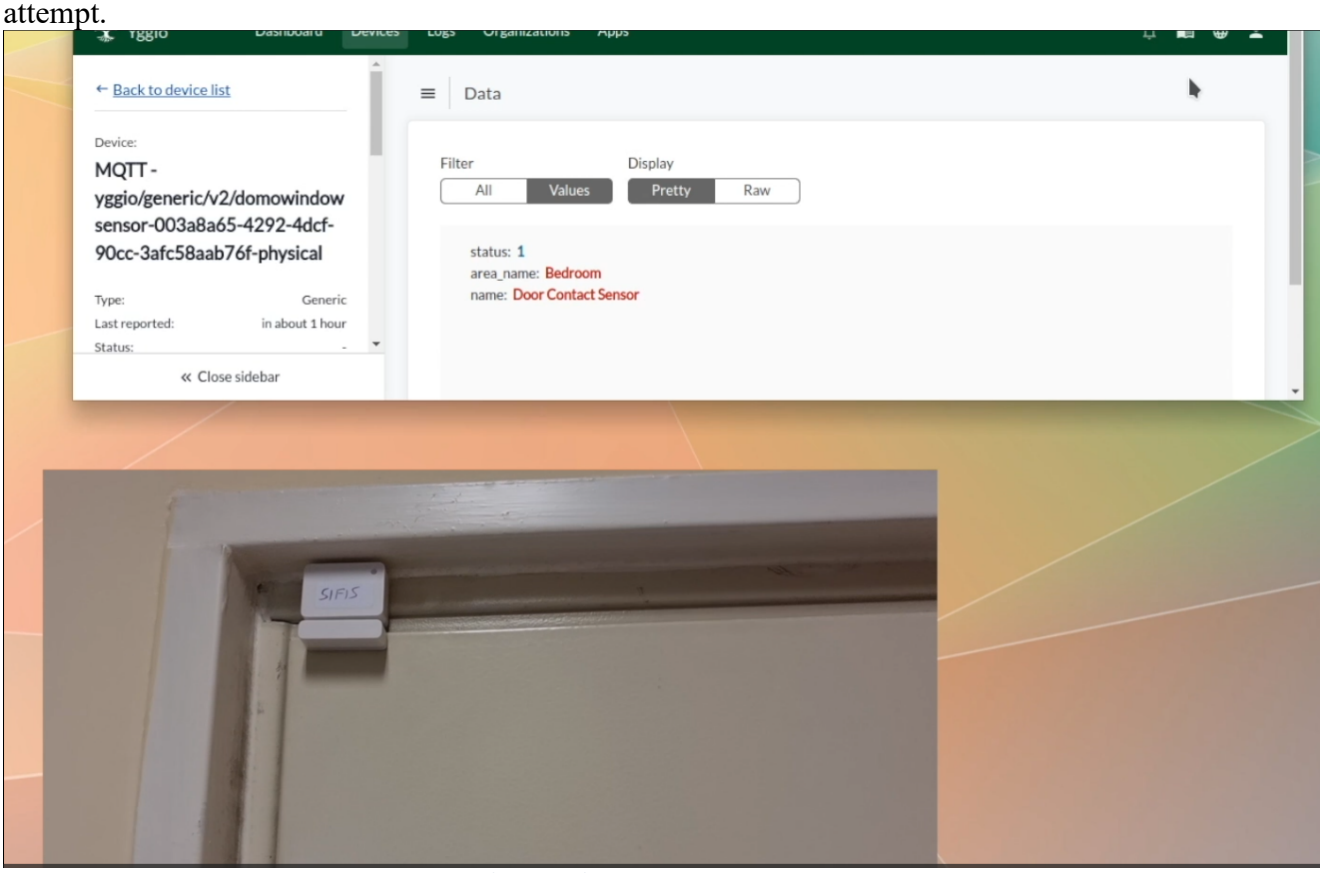


Figure 62: Notification of emergency situation - Intrusion Detection

6.15 *UC 15 – Turn on/off lights using the control panel*

This use case is satisfied through the Mobile Application or the Yggio cloud interface. This use case is related to a key functionality of a smart home, as it demonstrates that it is possible to remotely control actuators in the smart home such as lights, without using physical buttons/switches. Through mobile applications and Yggio, these operations can be done either when connected to the SIFIS-Home network, or from outside the SIFIS-Home premises.

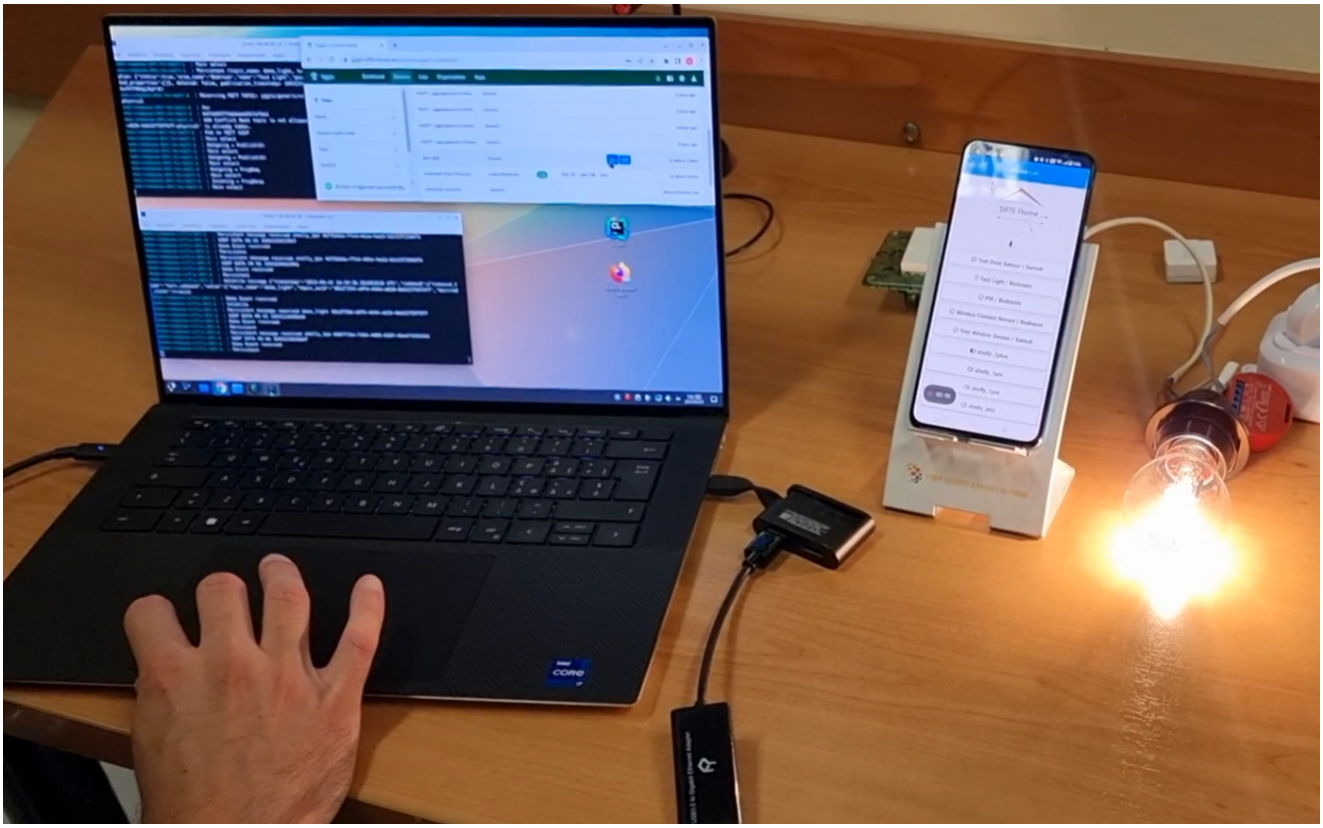


Figure 63: Light control use case

The considered actor for the validation of this use case is the home Tenant or Administrator, who will access the Mobile Application or Yggio to turn on and off a light bulb. The testbed for this demonstrator is made of three DOMO Gateways and one Shelly 1PM controlling a light bulb. The mobile phone is used to access the mobile application.

The use case is considered demonstrated if the user is able to effectively control the light, in a timely manner.

6.16 *UC 16 – Turn on/off lights pressing/releasing buttons*

This use case demonstrates how, despite the complexity introduced by the SIFIS-Home framework, it is possible to control actuators with physical switches or buttons, without introducing noticeable delays. This use case introduces also the concept of SIFIS-Home services, such as the Light Manager, which is an application implementing the logic for controlling lights through physical buttons. In fact, in the considered demonstrator, when using a Shelly actuator, we actually decouple the physical connection between a switch/button and the controlled light. This enables a much higher flexibility as it is possible to configure different functionalities for a single switch/button. Services are similar to third-party applications, but they do not use the SIFIS-Home Developer –APIs, and they are considered trusted as they come already integrated as “native applications”.

The demonstrator is very similar to the previous use case, introducing in addition the physical buttons. The use case is considered validated if the light is controlled in a timely manner, non-distinguishable from controlling a light without having the SIFIS-Home overhead.

6.17 *UC 17 – Being able to interact with the devices only if authorized*

This use case demonstrate that the SIFIS-Home framework prevents unauthorized users to access the functionalities of the smart home. The use case employs the DOMO gateways and the mobile device.

We demonstrate the use case in the following way. The user opens the Mobile Application and the Login panel is presented to it. First, a set of valid credentials is used, i.e., we test the situation where a valid User of the smart home is trying to perform the login operation. The mobile application sends the user credentials to the SIFIS-Home KeyCloak service and receives an access token back. Then, the mobile application uses the received token to access the DHT. Since the token is successfully validated by the system, the information stored in the DHT can be retrieved and shown to the user. We then logout from the mobile application and try to login again by using a set of invalid credentials. This time, the mobile application does not receive an access token from KeyCloak. Hence, the login cannot proceed further.

We consider the use case successfully validated if the valid login operation succeeds and the invalid one fails.

6.18 *UC 18 – Being able to control the house in case of failures*

This use case demonstrates one of the key features of the SIFIS-Home framework: the resilience to failures or compromission of smart devices. The use case is demonstrated in the DOMO testbed, by using the full set up of three DOMO gateways, which are used to control a Shelly actuator and the attached lightbulb. The use case aims at demonstrating the capability of self-healing of the SIFIS-Home framework. The involved actor is an external user who will physically disconnect one smart device, which is directly connected to and controls the NSSD attached to the lightbulb. Also, a Tenant will be involved to control the light through the mobile application. To demonstrate the functionality of the provided use case, we first show the DHT status to demonstrate that the NSSD is actually controlled by a specific DOMO gateway. Then, we unplug the power cord from that device, and we show that, after a short time, the NSSD can be controlled again since it automatically reconnects to another gateway.

The use case is validated if the reconfiguration happens successfully and it is still possible to turn on/off the light from the mobile application, after the smart device has been disconnected.

Figure 64: Disconnection of DOMO gateway to test self-healing

6.19 *UC 19 – Being alerted if a device is generating anomalous traffic*

This use case aims at verifying the ability of the SIFIS-Home system to detect anomalous traffic conditions and take appropriate countermeasures. To validate this use case, we employ the DOMO gateways and the mobile application. The use case involves the DHT Manager, the AUD Analytics, the System Protection Manager and the Mobile Application.

We validate the use case in the following way. First, we open the mobile application and verify that no notifications are currently present in the main UI. We intentionally activate a script that generates anomalous traffic. In detail, we use command *"nmap -sn external_network_address"* to generate a set of outgoing discovery packets. The AUD Analytics detects the anomalous conditions and reports the event to the System Protection Manager. The System Protection Manager then sends a notification that is received by the mobile application and shown to the user.

We consider the use case successfully validated if the anomalous traffic condition is detected and if the notification of the anomaly is correctly received on the mobile device.

6.20 *Summary of use cases implementation and validation*

The following table summarizes the previously described use cases and tests for a more practical presentation in view of the formal validation through GQM.

Use Case	**Implementation in the pilot**	**E2E Test technique**
UC 01 - Login through biometrics	For the pilot implementation of this use case, we are going to use a Laptop (SD) provided with a camera. The identity of the user will be recognized through a Face Recognition operation.	A Resident User for which a face model has been registered into the System will enter inside a certain room. If the Resident is identified by the System, a notification of successful recognition of the User will be sent. Otherwise, a notification event reporting a failure in identifying the User will be generated. The test will be repeated with multiple different Resident users.
UC 02 - Operate through voice commands	For the pilot implementation of this use case, we are going to use a Laptop (SD) provided with a microphone, or a smartphone. Also, the DoMO Wi-Fi actuators and physical lights will be used Then, voice commands will be sent to the System.	A Resident User will issue a specific voice command (e.g., "turn on the light"). If the command is successfully recognized by the System a light will be turned on. Otherwise, a notification will be sent reporting that the command was not intelligible. The test will be repeated with multiple different Resident users.
UC 03 - Being alerted if motion sensors detect people presence	For the pilot implementation of this use case, we are going to use a DoMO gateway and a Wi-Fi actuator to which a motion sensor is connected to.	A person will enter inside a certain room and motion will be detected. The motion event should be reported in the Notification panel of the mobile application and the web control panel.
UC 04 – Get notification about software intrusion	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	We are going to install and execute a malware on the DoMO gateway. The software intrusion should be detected and reported to the user through Mobile

		application notification panel.
UC 05 – Register device	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a DoMO Wi-Fi actuator.	An authorized User will open the mobile application and use a specific control panel to register a DoMO Wi-Fi actuator (NSSD) into the system. The mobile application should ask the user to provide the details of the NSSD to be inserted into the system (e.g., MAC address, user/password pair to use to communicate with the device). After the device is inserted into the system, it should be possible to control it from the control panel.
UC 06 – Unregister device	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a DoMO Wi-Fi actuator.	An authorized User will open the mobile application and use a specific control panel to un-register the DoMO Wi-Fi actuator (NSSD). After the device is unregistered is should not be reported anymore in the device list. Also, it should not be possible to control it.
UC 07 – Configure device	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a DoMO Wi-Fi actuator.	An authorized User will open the mobile application and use a specific control panel to configure a specific DoMO Wi-Fi actuator (NSSD). From the panel it should be possible to change the device configuration, e.g., specify the specific physical objects (lights, sockets, etc) to which the actuator is connected to.
UC 08 – Install third party applications	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	An authorized User will open the mobile application and use a specific control panel to obtain the list of the third-party applications that can be potentially installed into the System. The user will select a specific third-party application for installation. At the end of the procedure, the application should be installed on the Smart Device.
UC 09 – Parental control	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	The User will open the mobile application. The System should allow the User to define new policies or change existing ones. In detail, we are going to define a policy that does not allow turning on the appliances of a certain room when there are children inside.
UC 10 –	For the pilot implementation of this use	The Administrator User will open the

Configure User Settings	case, we are going to use a DoMO gateway and a smartphone.	mobile application and use the User control panel. The User control panel should allow the Administrator to define policies for the different users.
UC 11 – Control statistics and analytics	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	The User will open the mobile application and will select a specific device. The mobile application should report a log and an information panel that reports the main events related to the device.
UC 12 – Remote configuration of device	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	The User will open the mobile application when he/she is on a remote side. The System should allow the User to change the configuration of a device.
UC 13 – Remote configuration of policies	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	The User will open the mobile application when he/she is on a remote side. The System should allow the User to define new policies or edit existing ones.
UC 14 - Remote handling of emergency situations	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a DoMO Wi-Fi actuator connected to an alarm contact sensor.	We open the alarm contact sensor. The event should be reported in the Notification panel of the mobile application and on the web-based control panel. The User should be provided with system logs and should have the possibility to control the house to, for example, turn off the alarm.
UC 15 – Turn on/off lights using the control panel	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a DoMO Wi-Fi actuator connected to a physical Light.	The User will open the mobile application and will use the control panel to get the list of the devices that are installed inside his/her house. The mobile application should show the possible commands for the different devices. The User will then use the GUI controls to turn on a certain Light. The Light should be turned on and the new Light state should be updated on the UI.
UC 16 – Turn on/off lights pressing/releasing buttons	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a DoMO Wi-Fi actuator connected to both a physical Light and a physical button.	The User will press the physical button. The System should detect the button press event and turn on/off the Light.
UC 17 – Being able to interact with the devices only if authorized	For the pilot implementation of this use case, we are going to use a DoMO gateway and a smartphone.	An Authorized User will perform a login operation. The operation should succeed. An Unauthorized User will perform a login operation. The operation should fail.
UC 18 – Being	For the pilot implementation of this use	We start the test with the 2 DoMO

able to control the house in case of failures	case, we are going to use 2 DoMO gateways, a smartphone and a DoMO Wi-Fi actuator connected to a physical light.	gateways active and operational. We verify that it is possible to turn on/off the physical light. We turn off the DoMO gateway to which the DoMO Wi-Fi actuator is connected to. After some time from the deactivation of the DoMO gateway, we issue the turn on/off command again. It should still be possible to control the physical light.
UC 19 – Being alerted if a device is generating anomalous traffic	For the pilot implementation of this use case, we are going to use a DoMO gateway, a smartphone and a NSSD.	We simulate an anomalous traffic condition by changing the program/firmware executed by the NSSD. The System should detect the anomalous traffic condition and report the event in the Notification panel of the mobile application.

Table 3. Pilot implementation and E2E test case for every use case

7 GQM Validation

The used GQM template is reported in the following table.

Object of study	SIFIS-Home framework
Purpose	Validation of the main use cases
Focus	Functional requirements
Perspective	End user (E2E testing)
Context	Pilot implementation

The GQM questions and metrics for every E2E test case are reported below.

Use Case	GQM questions – Answers	GQM metrics – Value
UC 01 - Login through biometrics	Q1.1: Is a Resident User recognized by the System? - Yes Q1.2: Is a Non-Resident User not recognized by the System? - Yes Q1.3: Is a notification sent when the Resident is recognized? - Yes Q1.4: Is a notification sent when the Non-Resident is not recognized? – Yes	M1.1: Percentage of resident users that are correctly recognized by the system. – 87% M1.2: Percentage of non-resident users that are recognized as resident users by the system. – 0% M1.3: Percentage of notifications correctly sent upon resident notification. – 100% M1.4: Percentage of notifications correctly sent upon non-resident notifications. – 100%
UC 02 - Operate through voice commands	Q2.1: Is the voice command executed successfully by the System? - Yes Q2.2: Is a notification generated if the voice command is not intelligible? – No	M2.1: Percentage of voice commands that are correctly executed by the system. - TBA M2.2: Percentage of notifications correctly generated upon not intelligible voice commands. - 0%

UC 03 - Being alerted if motion sensors detect people presence	Q3.1: Is the people presence detected by the System? - Yes Q3.2: Is the motion event reported by the System? - Yes	M3.1: Percentage of motion events that have been successfully reported during testing. – 100% M3.2: Number of false motion events that have been reported during testing. – 0%
UC 04 – Get notification about software intrusion	Q4.1: Is the malware execution detected? - Yes Q4.2: Is the software intrusion reported to the user? - Yes	M4.1: Number of times the software intrusion event has been successfully detected. – 94% M4.2: Percentage of false software intrusions that have been detected. – 0,2%
UC 05 – Register device	Q5.1: Is the device correctly registered into the system? - Yes Q5.2: Is it possible to control the device after it has been registered into the system? - Yes	M5.1: Percentage of devices that are correctly registered after a testing session. – 100% M5.2: Percentage of controllable devices of those registered. – 100%
UC 06 – Unregister device	Q6.1: Is the device correctly unregistered from the System? - Yes Q6.2: Is it possible to control the device after it has been deregistered? - No	M6.1: Percentage of correctly unregistered devices after a testing session. – 100% M6.2: Percentage of controllable devices of those that are deregistered. - 0%
UC 07 – Configure device	Q7.1: Is the current configuration of the device available to the smart home users? - Yes Q7.2: Is it possible to change the configuration of the device? - Yes	M7.1: Percentage of users to which the current configuration is available. – 100% M7.2: Percentage of users that are able to change the configuration of the device. -100%
UC 08 – Install third party applications	Q8.1: Is it possible to get the list of 3rd party applications? - Yes Q8.2: Is the installation of a 3rd party application performed successfully? - Yes	M8.1: percentage of 3rd party applications retrieved. – 100% M8.2: percentage of 3rd party applications that are correctly installed. – 100%
UC 09 – Parental control	Q9.1: Is it possible to create the policy required for parental control? - Yes	M9.1: existence of a policy for parental control. - Yes
UC 10 – Configure User Settings	Q10.1: Is it possible to retrieve the list of the configured users? - Yes Q10.2: Is it possible to define different policies for the different users? - Yes	M10.1: number of configured users. - 5 M10.2: number of policies for the users. – 2
UC 11 – Control statistics and analytics	Q11.1: Are statistics and analytics reported for every device? - Yes	M11.1: percentage of devices for which statistics and analytics are correctly reported. -100%
UC 12 – Remote configuration of device	Q12.1: Is it possible to configure a device from a remote side? -Yes	M12.1: percentage of devices that can be configured from remote. - 100%
UC 13 – Remote configuration of policies	Q13.1: Is it possible to configure and edit policies from a remote side? -Yes	M13.1: percentage of policies that can be configured from remote. - Yes

UC 14 - Remote handling of emergency situations	Q14.1: Is the emergency reported in the Notification panel of the mobile application? - Yes Q14.2: Is it possible for the users to interact with the House after the emergency notification is received? -Yes	M14.1: Number of times a false contact sensor activation is reported. - 0 M14.2: Percentage of contact sensor activations that have been successfully reported. -100%
UC 15 – Turn on/off lights using the control panel	Q15.1: Is it possible to get the list of installed devices? -Yes Q15.2: Is it possible to get the list of commands for a certain device? -Yes Q15.3: Are commands executed successfully? -Yes	M15.1: number of installed devices retrieved. - 5 M15.2: number of commands retrieved for each device. - 2 M15.3: percentage of commands executed successfully. - 100%
UC 16 – Turn on/off lights pressing/releasing buttons	Q16.1: Is the button press event detected and reported? - Yes Q16.2: Is the light turned on/off after a button press event? - Yes	M16.1: Number of times the system failed in detecting a button press event. - 0 M16.2: Percentage of successful detections of button press events. - 100%
UC 17 – Being able to interact with the devices only if authorized	Q17.1: Is an Authorized User able to perform a login operation? - Yes Q17.2: Is the login request of an Unauthorized user rejected? - Yes	M17.1: Percentage of successful logins by authorized users. -100% M17.2: Percentage of successful logins by unauthorized users. -0%
UC 18 – Being able to control the house in case of failures	Q18.1: Is it possible to control the devices of the house before the Smart Device failure occurs? - Yes Q18.2: Is it possible to control the devices of the house after the Smart Device failure? - Yes	M18.1: percentage of devices that can be controlled before the smart device failure occurs. – 100% M18.2: percentage of devices that can be controlled after the smart device failure occurs. -100%*
UC 19 – Being alerted if a device is generating anomalous traffic	Q19.1: Is the anomalous traffic condition detected? -Yes Q19.2: Is the user informed that a device is generating anomalous traffic? -Yes	M19.1: Number of times the anomalous traffic event has been successfully detected. – 100% M19.2: Percentage of false anomalous traffic events that have been detected. – 0%

Table 4: GQM questions

For the purpose of evaluating the overall outcome of testing purposes for functional requirements, we also consider two additional aggregate metrics, described below:

1) **N_uc**: Number of use cases that the pilot is able to demonstrate. Such metric is computed as the raw count of the use cases for which a test case is run successfully;
2) **N_fr**: Number of functional requirements that are covered by the pilot. Such metric is computed as the sum of the functional requirements that are covered by the use cases that were successfully run in the acceptance testing phase.

We successfully validated all the different use cases. Hence N_uc = 19. In addition, we verified 41 out of 57 functional requirements having N_fr=41.

8 Usability Requirements Validation

Reprising from D5.4, the validation of the usability requirements has been postponed to this deliverable as we deemed it as more appropriate to perform such validation directly on a real testbed instead of on a simulated one.
The following table reports the list of usability non-functional requirements and the results of the validation, performed on the DOMO testbed.
For verifying the usability, we have involved two persons that were not participating in the project

NFR ID	NFR Description	Validation
US-01	The system shall be easy to use for users with no technical background.	Tenant functionalities have been operated by 2 non-trained users.
US-02	The SIFIS-Home system shall be autonomous and learn based on the users' habits, still according to defined privacy policies.	This requirement has not been validated as it requires long-term usage.
US-03	The SIFIS-Home system shall consider special cases in its design, such as color blindness.	This requirement has not been validated.
US-04	The SIFIS-Home system shall preserve consistency among all devices, related database and constraints.	This requirement is validated by design
US-05	The SIFIS-Home hardware components should be easy to use for the elderly and users with no engineering background.	This requirement has not been validated as it was not possible to involve elderly users
US-06	The SIFIS-Home system shall have an explorable interface.	This requirement has been validated by design
US-07	Proper and easy hardware installation should be considered.	This requirement has been validated by design through the design of the DOMO pilot
US-08	The image-based identification through biometrics in a room (interior) or in an open space (exterior), without obstacles or face covering elements, it should be performed by the system in a radius of at least 10 meters from the device.	This requirement has been validated in a controlled environment.
US-09	An untrained user should be able to understand that an attack is ongoing in less than a minute from reading the SIFIS-Home alert or notification.	An attack has been simulated during the testing of tenant functionalities by untrained users. The users have been asked to interpret the received notification and the answer was correct.
US-10	An untrained user should be able to recognise a software intrusion in less than one minute.	See validation of previous requirement
US-11	An untrained user should be able to perform the device registration procedure in less than 5 minutes.	An untrained user has been explained the rationale of the device registration operation and has been able to perform the operation through the mobile app in 3 minutes.
US-12	An untrained user should be able to perform the device de-registration procedure in less than 5 minutes.	As for previous requirement

US-13	An untrained user should be able to perform the configuration of devices in less than 5 minutes.	As for previous requirement
US-14	An untrained user should be able to perform the installation of an application in less than 5 minutes.	The untrained user has been able to install a third-party app in less than a minute.
US-15	An untrained user should be able to complete the configuration of policies for groups of users in less than 5 minutes.	As for requirement US 11
US-16	An untrained user should be able to complete the configuration of policies for groups of devices in less than 5 minutes.	As for previous requirement
US-17	An untrained user should be able to complete the configuration of profiles in less than 5 minutes.	This requirement has not been validated
US-18	An untrained user should be able to perform a profile change in less than 30 seconds.	This requirement has not been validated
US-19	An untrained user should be able to access the statistics for visualizing and interpreting them in less than 5 minutes.	As for requirement US 11
US-20	The Multi-Level Anomaly Detection system (MLADS) must monitor network traffic provided by several input sources and several locations.	Validated through UC 19 demo
US-21	The workload of the devices should be available to the MLADS.	Validated by design
US-22	The list of applications running on each device should be available to MLADS.	Validated by design
US-23	Raw sensor data must be available to be analysed by MLADS.	Validated by design
US-24	Features from different devices should be aggregable directly or by means of pre-processing through specific analysis tools.	Validated by design
US-25	When possible, a dataset should not be present on a single device for analysis.	Validated by design
US-26	The presence of a GPU is needed to perform DL-based analysis.	Validated by design

Appendix A: List of Code Components

Code Component	GitHub Public Link
DHT Manager	*https://github.com/sifis-home/libp2p-rust-dht*
NSSD Manager	*https://github.com/sifis-home/domo-wot-bridge*
WoT firmware for DoMO WiFi actuators	*https://github.com/sifis-home/domo-wot-actuator*
Smart Device Mobile API	*https://github.com/sifis-home/wp6_mobile_application_api*

Appendix B: List of Acronyms

Acronym	Meaning
RGBW	Red Green Blue White Light
EMMC	Embedded MultiMediaCard
WoT	Web of Things
DHT	Distributed Hash Table
SD	Smart Device
NSSD	Not So Smart Device

Appendix C: DoMO GW OpenWrt Distribution

OpenWRT is a Linux distribution geared towards building firmware for routers. It is inherently focused on cross-compilation and is based on a GNUMake-based toolchain closely resembling the Linux Kernel build system.

Its source distribution is split in multiple git repositories, the `openwrt` one contains the core components and the toolchain machinery, additional `packages` repositories provide optional components.

Its main configuration files are `.config` and `feeds.conf`. The former matches the file with the same name used in the Linux Kernel, the latter resembles Debian's `/etc/apt/sources.list` in format and purpose.

The package layout is a directory containing a single Makefile implementing pre-named `defines` and variables that are then sourced by the main Makefile if its source tree is `installed` using the feeds management script. For each package present in the `installed` tree a metadata index is produced and used by the build system .config machinery.

The packages are built as installable packages (opkg) or built and installed in the base image depending on the `.config`.

In order to produce an OpenWrt image for the DoMO Gateways the following are required:

- The OpenWRT core git tree
- A custom packages repository with packages for the custom software produced and the toolchain required to build them, in our case the Rust compiler is the only component missing.
- A package containing our custom configurations regarding the network and the boot process
- A `feeds.conf` pointing to our package repository.
- A `.config` targeting the hardware and adding to the base image the software

Additional care had been taken to not diverge from the distribution philosophy to reduce the odds of having clashing changes as the upstream distribution evolves:

- Every component providing a daemon has a procd-compliant initscript
- All the non-standard configuration is packaged as a uci-defaults script

DoMO GW OpenWRT distribution setup scripts

The whole SIFIS-Home OpenWrt image creation is automated via a simple bash script. In detail, the bash script performs the following steps:

- It clones the openwrt github mirror
- It generates the feeds.conf to include the SIFIS-specific packages
- It populates the .config file using the settings extracted using the diffconfig script
- It builds the full image and the upgrade image

DoMO GW flashing procedure

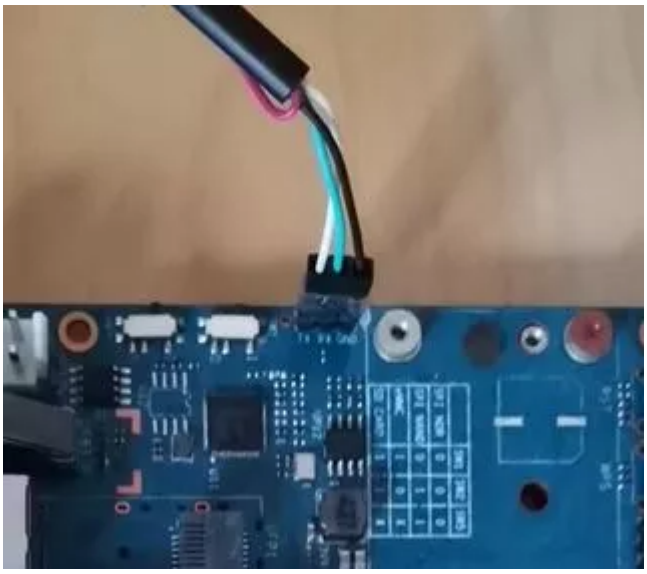
The Banana-pi R3 SoC sports a 128MB NAND and a 32MB NOR memory in mutually exclusive access and a SD port sharing the I/O pins with the internal 8GB eMMC storage.

The OpenWRT build system generates images that can target all the possible I/O. The SD image can flash the NAND and the NOR with an image, directly from u-boot. The NAND image can flash the eMMC from u-boot; the NOR image cannot, due to the constrained memory available.

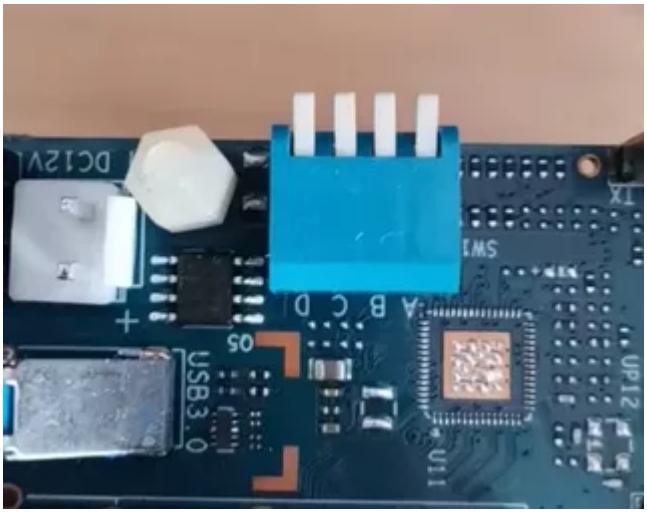
The manufacturer's suggested procedure to flash the system is to boot from the SD with a modern image, flash the NAND storage, reboot from the NAND and flash the eMMC.

In detail, the following actions need to be executed to flash a Banana Pi R3.

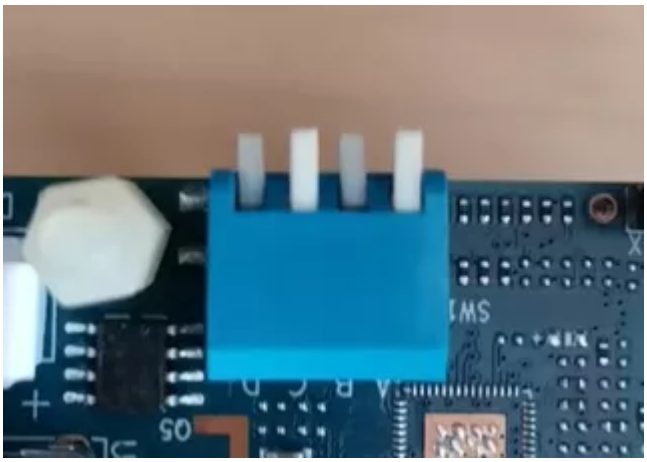
First, a USB-Serial converter should be connected to the debug console of the BPI R3 as shown in the picture below.



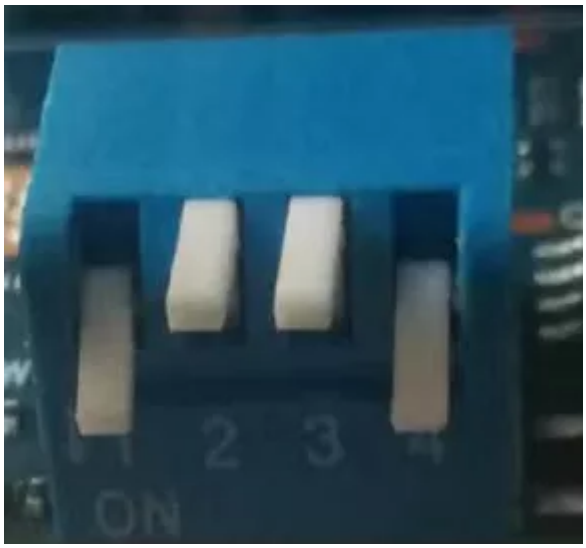
Also, an SD card flashed with the DoMO gateway image should be inserted in the SD card slot. Then, all the BPI R3 jumpers should be set to the high position so as to allow the card booting using the SD card.



After powering up the board, the “Flash to NAND” operation should be selected on the Uboot menu. After the operation is terminated, the board should be powered off and the jumpers should be set as reported in the following figure.



The board should be powered up again and the “Flash to EMMC” operation should be selected from the Uboot menu. After the operation is terminated, the board should be powered off again and the jumpers should be set as shown in the following picture:



Some time after the board is powered up, it can be noticed that a domo-<MAC> network is created. This means that the domo-bootstrap application is up and running and the DoMO gateway is ready to receive its configuration.