



D5.3

Final version of SIFIS-Home testbed

WP5 – Integration, Testing and Demonstration

SIFIS-Home

Secure Interoperable Full-Stack Internet of Things for Smart Home

Due date of deliverable: 31/03/2023

Actual submission date: 31/03/2023

Responsible partner: SEN

Editor: SEN

E-mail address:

hakan.lundstrom@sensative.com

31/03/2023

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: Håkan Lundström (SEN), Marco Tiloca (RISE), Marco Rasori (CNR), Domenico De Guglielmo (DOMO), Arpad Mueller (CNR), Marco Simoni (CNR), Samuli Stennud (RIO).

Approved by: Andrea Saracino (CNR), Joni Jämsä (CEN), Giles Brandon (IC)

Revision History

Version	Date	Name	Partner	Section Affected Comments
0.1	15/11/2022	Defined ToC	SEN	All
0.2	20/11/2022	Deprecated Deliverable	SEN	All
0.3	30/01/2023	Defined Testbeds	SEN, CNR, DOMO	All
0.4	10/03/2023	Ready for Review	SEN	All
1.0	31/03/2023	Final version	SEN	All

Executive Summary

This deliverable, which deprecates D5.1, describes the design, development, integration and the deployment of the final version of the SIFIS-Home emulated and simulated testbed that has been deployed to allow the partners to develop, deploy and test their applications for the different use cases. The emulated devices are virtual devices based on x86 hardware architecture meanwhile the simulated devices are Raspberry PIs on ARM hardware architecture. The two testbeds also integrate the cloud interface to enable access to the SIFIS-Home network from outside the home. To speed up development and validation of the cloud interface, we also included three live NSSD (Not so smart devices) based on a standard IoT protocol.

The state-of-the-art integration and deployment tools are based on Docker [DOCKER] with GitHub [GITHUB] as code, build and integration repository and Docker Hub in combination with Docker Watchtower [WATCHTOWER] as deployment repository which enabled us, together with the static analysis tools developed by WP2, to establish a full CI (Continuous integration) and CD (Continuous deployment) process. Docker Hub [DOCKER HUB] is also used as repository for 3rd party applications by the SIFIS-Home Marketplace and thus development of 3rd party applications will follow the same processes and the development of the core SIFIS-Home framework component and modules.

The testbed was designed based on the requirements and architectural design from WP1 and has gone through several iterations since the SIFIS-Home architecture evolved during the project execution. That several iterations would be required were expected since the original architecture was done early in the project before actual implementation was started by the partners in WP3, WP4, WP5 and WP6.

Further challenges were overcome in relation to the Cyber-Perimeter and how to manage the communication flow between the smart devices behind the firewall in the home and the cloud interface located on a server on the Internet. We concluded that the best way was to let devices inside the home set up the sessions to the cloud interface and subscribe to possible events, including user initiated, from the cloud interface. By doing so, the traffic will pass through firewalls with standard configurations without problems and the cloud interface is able, when required, to communicate with the devices inside the Cyber-Perimeter in a secure way.

While the final version of the testbed is designed and, to a large extent, up and running, the SIFIS-Home component integration and deployment activities are still ongoing. The focus has been on designing the testbed so that it can be used in a collaborative way by the partners to efficiently implement, deploy, verify and validate the SIFIS-Home technologies according to the requirements.

Table of contents

Executive Summary	3
1 Introduction.....	6
2 Actors, devices and the Cyber-Perimeter.....	6
2.1 Analysis of Components and Actors of the SIFIS-Home architecture	6
2.2 Analysis of The Smart Home Cyber-Perimeter	8
3 Testbed design	8
3.1 Main requirements on the testbed	8
3.2 Designing the testbed.....	10
3.3 Panarea server	11
4 Implementing the testbed	12
4.1 Architecture.....	12
4.2 Allocation of components	13
4.3 Definitions of testbeds	17
4.4 Component implementation and integration highlights.....	20
4.5 Authorization and Access management integration	24
5 Interfaces of the testbed	24
5.1 Mobile Application API.....	24
5.2 WebOfThings Smart Devices	25
5.3 Ratatosk, the FIWARE Context Broker.....	25
5.4 User Interface.....	27
5.4.1 Mobile Phone Application Interface	28
5.4.2 Cloud interface.....	29
5.5 GitHub.....	32
6 Continuous integration and deployment	34
7 Validation of implementation and of the testbed	36
8 Conclusion	42
9 References.....	43
Glossary	44

1 Introduction

In this deliverable, we report on the design and implementation of the final version of the SIFIS-Home testbed based on the requirements, architecture and design from WP1. The report goes through the different key steps that were performed with a focus on the testbed design aspect and the implementation implications of the requirements.

It starts with an identification of the user roles and the different device types that the testbed needs to support and discusses if we will have access to live devices or must rely on emulated and simulated devices. Then it moves on to cover the Cyber-Perimeter concept and how the expected firewalls at every home put some constraints on the communication flow between the home side of the SIFIS-Home network and the external side.

The final testbed design is based on the final architecture and the requirements defined in WP1 D1.4 as well as the consortium partners' competence and access to assets that can be used in setting up the testbed. Once agreement was reached with all partners about who does what and when, the implementation aspects of the different main components were analysed one by one, also looking for any specific design considerations that were overlooked during the original architecture design phase. In chapter 6 the state-of-the-art tools used for CI (Continuous integration) and CD (Continuous deployment) are discussed and the process of how they work is described.

The next steps walk through the interfaces for some of the key components, like the FIWARE Context Broker Ratatosk [RATATOSK], which holds the state of the system in the cloud UI for users that access the SIFIS-Home network from outside the home. A further focus is put on the mobile application. The mobile application API as well as WebOfThings [WoT, 2021] which is the base for how SIFIS-Home smart devices interface NSSD (Not so smart devices). There is also a description with screen shots from some of the components in the web UX while the concluding section contains the validation strategy for each of the requirements from D1.2.

2 Actors, devices and the Cyber-Perimeter

2.1 Analysis of Components and Actors of the SIFIS-Home architecture

The main components of the SIFIS-Home testbed architecture are the following:

- *Smart Devices*: These are devices that implement a SIFIS-Home distributed hash table (DHT) that enables a client to set up a Peer-to-Peer (P2P) logical model. General examples of Smart Devices are Smart TVs, Smart Refrigerators, Laptops/Desktops, Family Hubs. From a testbed point of view, we are planning to both simulate them and use physical devices based on Raspberry Pis.
 - *Internet Connected Smart Devices*: This is a subset of the Smart Devices without a SIFIS-Home client APP, like smart phones and tablets, but located inside the Smart Home Cyber-Perimeter and equipped with a network interface which enables connectivity outside of the Smart Home. From a testbed point of view, in SIFIS-Home we have analytics capabilities to analyse the traffic from those devices via a gateway network interface layer with a SIFIS-Home client software.
- *Not So Smart Devices (NSSD)*: These are constrained devices that cannot be customized by installing third party software or applications. In the SIFIS-Home testbed, we are using several of these, such as smart sensors, smart cameras, smart lights, smart speakers and smart locks. NSSDs are either using standard protocols like Z-wave, LoRaWAN or are TCP/IP based with

Internet connectivity options and are set to connect through their responsible Smart Devices. In SIFIS-Home we use WebThings on top of TCP/IP as NSSD devices.

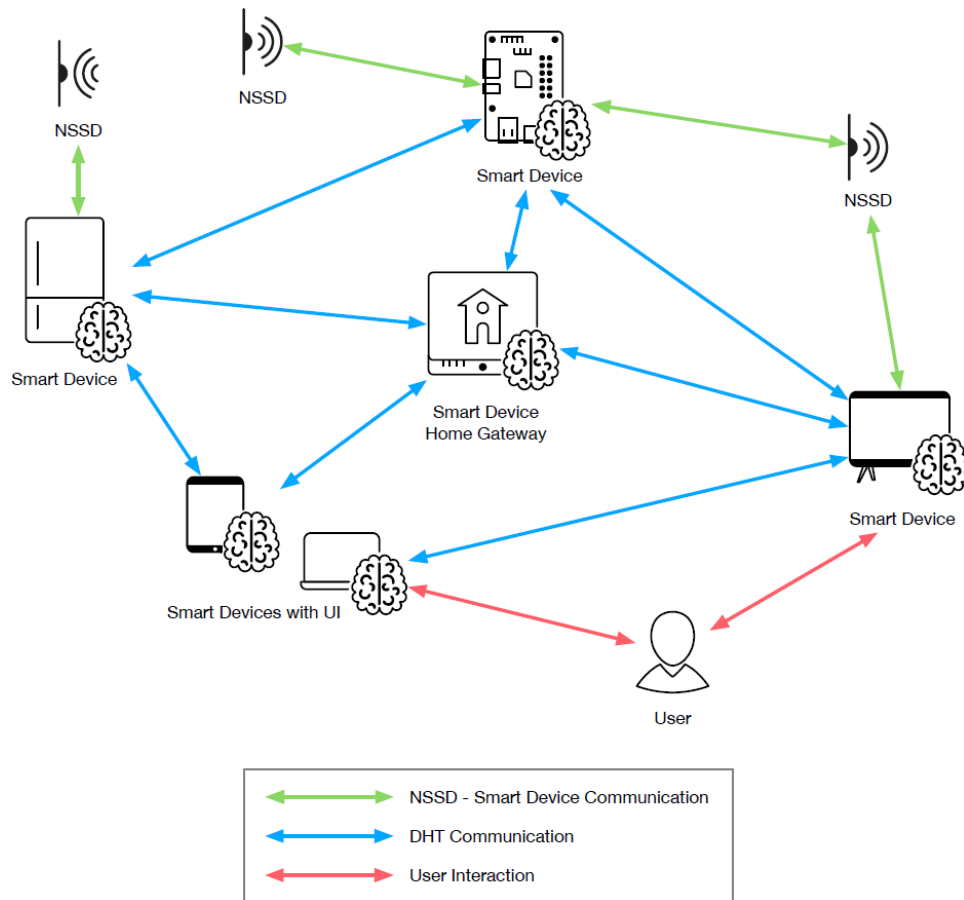


Figure 1: Communication and interaction between components

The actors we have defined for the SIFIS-Home architecture are the following:

- *SIFIS-Home Administrator*: The administrator is a human user who is the owner of an instance of the SIFIS-Home architecture.
- *SIFIS-Home Tenant*: The SIFIS-Home tenant is the standard user of the Smart Home system.
- *SIFIS-Home Maintainer*: The maintainer is an entity external to the Smart Home and trusted by the administrator to correctly configure the Smart Home security, privacy and safety policies.
- *SIFIS-Home Tenant with restrictions*: This user is a Smart Home tenant with restrictions on the functionalities they can access.
- *Guest*: A guest is a Smart Home user who is not resident in the Smart Home but is allowed to access and use the premises and some functionalities for a limited amount of time, upon authorization from the administrator or another tenant.
- *External Operator*: The external operator could be a technician, a plumber, gardener, or house cleaner, allowed to access the Smart Home premises for a limited amount of time, with the authorization from a tenant.

In the testbed design, all these roles need to be represented and have their distinct usage.

2.2 Analysis of The Smart Home Cyber-Perimeter

In order to protect the Smart Home and its users from unintended disclosure of sensitive information, in WP1 we defined a logical distinction between the outside and inside of the Smart Home, based on what we called the Smart Home *Cyber-Perimeter*.

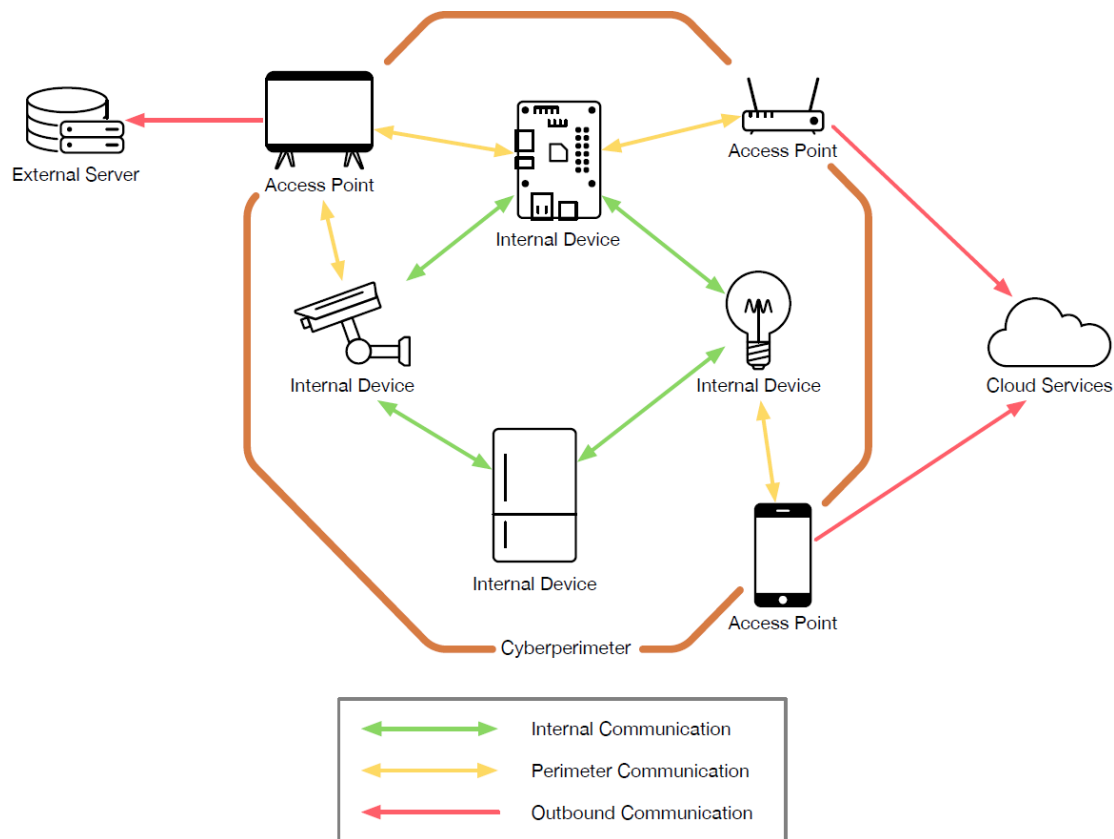


Figure 2: Concept of Smart Home Cyber-Perimeter

From a testbed point of view, the Smart Home Cyber-Perimeter is secured by a firewall that isolates the SIFIS-Home network from the internet on the outside. Inside the home the SIFIS-Home network analytics are constantly monitoring the activities of the network to be sure no malicious actor has been able to penetrate the firewall.

Since a user should also be able to access the SIFIS-Home network from outside the home, a cloud service is required as well as part of the SIFIS-Home architecture. The SIFIS-Home network subscribes to a publish / subscribe broker in the cloud service to be able to share the status of the network with the cloud service and to be able to receive commands from users outside of the home via the cloud service. The fact that a publish / subscribe method is used ensures all communication sessions start inside the house and therefore do not compromise security.

3 Testbed design

3.1 Main requirements on the testbed

When designing the testbed, we considered how to setup, verify and validate the testbed itself and to

ensure the fundamental part of the SIFIS-Home system worked as intended. The conclusion was that it should be possible to verify and validate both the Functional requirements as well the NFR (Non-functional requirements), as defined in D1.2 and D2.1, when using the testbed. The principal areas that needed to be covered were as follows.

- Integration feasibility as a fundamental requirement of the testbed. It must be feasible to integrate all SIFIS-Home software components and analytics in an efficient CI (Continuous integration) flow into the testbed for both emulated and simulated devices.
- Deployment flow is another fundamental requirement of the testbed. After delivery of new software to the GitHub repositories, it must be feasible to automatically build and deploy the new updated code to all emulated and simulated devices connected to the testbed, a so-called CD (Continuous deployment) flow.
- Communication flow and interfaces between all integrated components, such as the cloud interface to the SIFIS-Home Smart Devices, and between SIFIS-Home Smart Devices to NSSD (Not So Smart Devices).
 - This holds both for simulated physical smart devices executing on Raspberry PIs as well as for emulated devices running in virtual machines in a server that is part of the testbed.
- Hardware architecture, the testbed, the SIFIS-Home framework and the continuous integration and continuous deployment tool chain must support the deployment of both x86 and ARM hardware architecture. There is both x86 and ARM based hardware included in the testbed design.
- It is also important to check that every interface behaves as expected and that data structures stored in the DHT (Distributed Hash Tables) as well as databases in the cloud interface look and behave as expected.
- General UI interaction, verification and validation. If things work well when performing different UI interactions, it is a good indication that the system works fundamentally correctly and as expected.
- Notification and event management verification and validation. These are events that could arise from any device or analytics executing in the network. It must be possible to display these events and present their outcomes to the user, thus enabling decision-making on how to handle the events.
- Performance related verification and validations. The system as such must be reliable and display good performance when different types of user interactions occur.
- Stability related verification and validations. The system must be stable in the long term and be able to run for consecutive months without any execution problems.
- Security related verification and validations including policy enforcement depending on logged in user and intrusion detections and the other analytics defined in WP3 and WP4.
- Privacy related verification and validations are collected to ensure that no data violate GDPR regulations.
- The Cyber-Perimeter is one of the most difficult requirements to fulfil, since the SIFIS-Home network and its devices must be reachable for end users who may be inside or outside the home.

In summary, once all the above items have been successfully verified and validated in a SIFIS-Home system in the testbeds, it should also be possible to assess the key metrics, such as code coverage, unhandled exceptions, security metrics and privacy metrics defined in deliverable D1.2.

3.2 Designing the testbed

The testbeds are built around a server, called “Panarea”, which is located at the CNR facilities and can be reached by the partners via SSH - Secure Shell access – in order to upload and configure code. In the Panarea server, the SIFIS-Home cloud interface based on Sensative Yggio will run, as well as several virtual devices that each will serve as a SIFIS-Home Smart Device based on x86 hardware architecture. These emulated devices are expected to include the full SIFIS-Home software stack including applicable analytics and network security solutions. The testbeds should include physical SIFIS-Home Smart Devices based on Raspberry PI’s. A Raspberry PI is based on ARM hardware architecture. Furthermore, some standard validation NSSD - IoT products based on LoRa WAN protocol were connected early to the testbed to collected data for analytics as well as secure basic software quality control of the cloud interface.

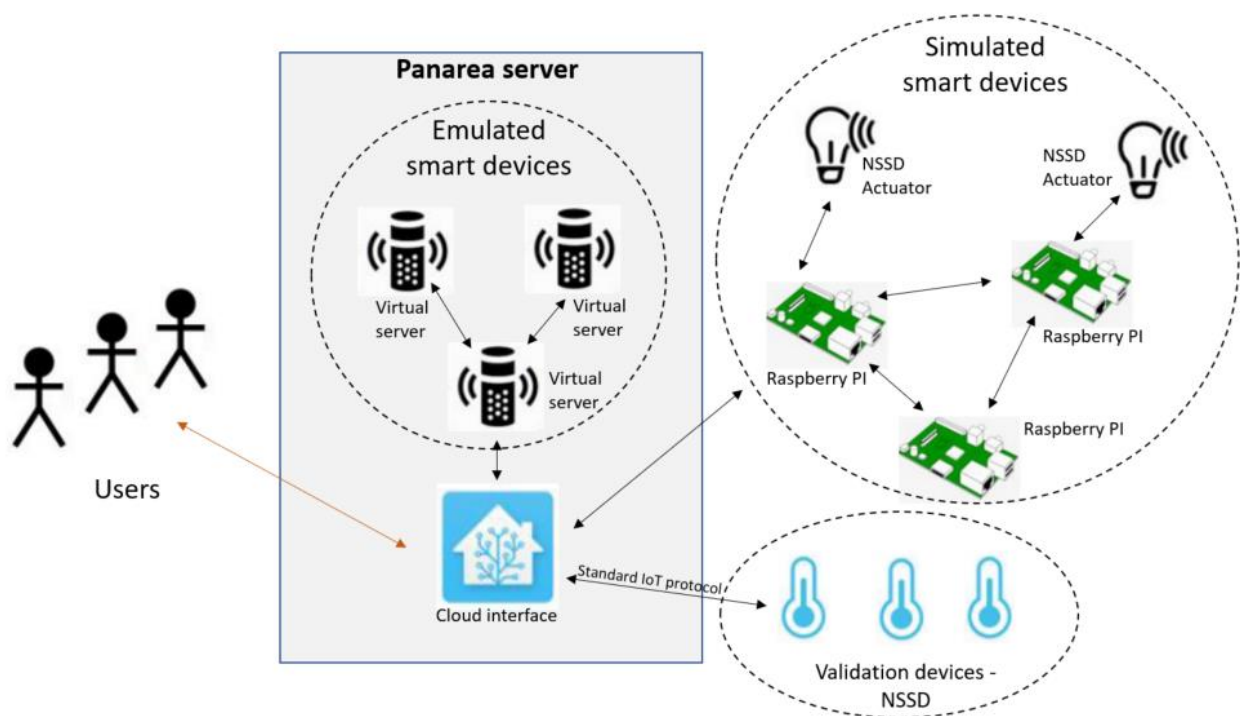


Figure 3: The SIFIS-Home testbed setup

The hardware support requirement - that both x86 and ARM are required - as well as the knowledge that partners used different types of programming languages and internal tool chains led to the conclusion that a software container-based approach was the only feasible way forward. The container solution selected was the industry standard and open-source Docker, because it fulfils the requirements on both the hardware support as well as allows partners to use different types of tools to develop their code. There are also state-of-the-art continuous integration and continuous deployment tools readily available for Docker which can be leveraged. The fact that partners’ own tool chains are also supported is vitally important for the partners to be able to exploit the SIFIS-Home results in their respective products.

In summary, the testbed design includes Docker based emulated SIFIS-Home Smart Devices, the physical simulated SIFIS-Home Smart devices built upon Raspberry PIs with connected WebThings based NSSD (Not so smart devices), and some standard IoT LoRaWAN devices for validation purposes. All will be connected via the core technology in SIFIS-Home, the DHT – Distributed Hash Table with a publish/subscribe mechanism. The SIFIS-Home devices will authenticate with and registrate with the cloud interface thus creating the SIFIS-Home network. A mobile application installed in an end user's mobile phone can be used to directly interface the SIFIS-Home network from inside the home, for example, to turn on or turn off items without going via the cloud interface. End users outside of the home can access the SIFIS-Home network via the cloud interface, thus enabling end users to check the system status of the Smart Home from anywhere.

3.3 *Panarea server*

The SIFIS-Home emulated testbed is being built around the Panarea server, hosted at the CNR premises, in a dedicated virtual machine (VM). The VM exploits Docker for deploying a lightweight testbed. In particular, through each Docker container, we instantiate both the cloud interface as well as emulated SIFIS-Home smart devices. Thus, each Docker can be viewed as a Linux-based device, able to install the software of the SIFIS-Home framework. To complete the testbed, both physical simulated Smart Devices and Not So Smart Devices will get connected to the cloud interface and the emulated Smart Devices on the Panarea server. In Figure 4 we depict the logical structure of the emulated testbed. As shown in the figure, the testbed exploits Docker as a virtualization environment, on top of an ESXi-based VM. Each Docker container represents a Smart Device, thus being able to run the SIFIS-Home framework as a service on top of it. A further container, in the current testbed, runs Ratatosk as an external component, whose functionalities can be queried via API.

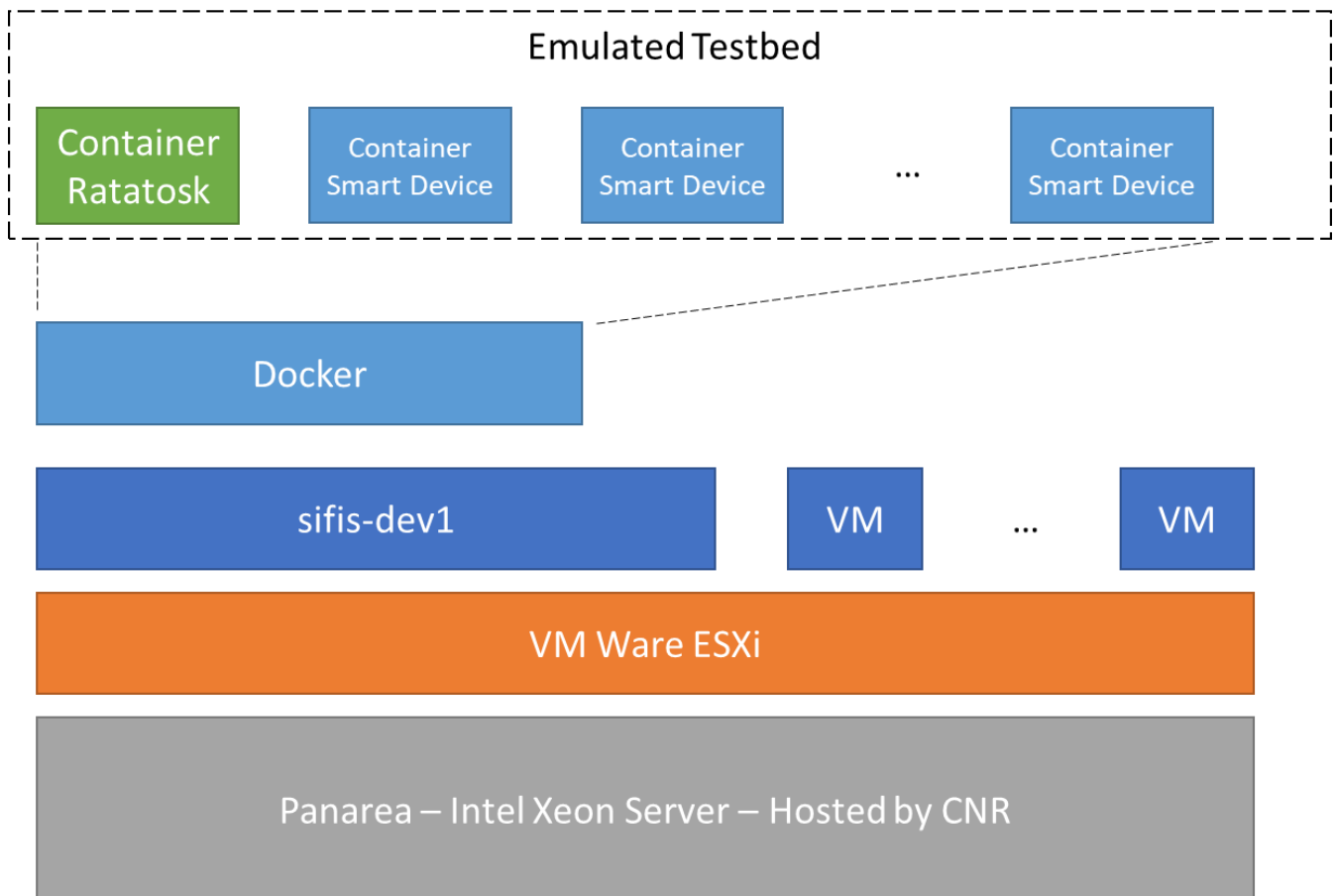


Figure 4: Architecture of the SIFIS-Home emulated testbed

4 Implementing the testbed

4.1 Architecture

The design of the SIFIS-Home framework in WP1 is based on Docker containers with *microservice* design patterns. Each microservice defines an API for interfacing with it. This design pattern applies to the testbed design as well but, to be able to make everything work well, a few extra integration components were created to enable us to verify and validate the system.

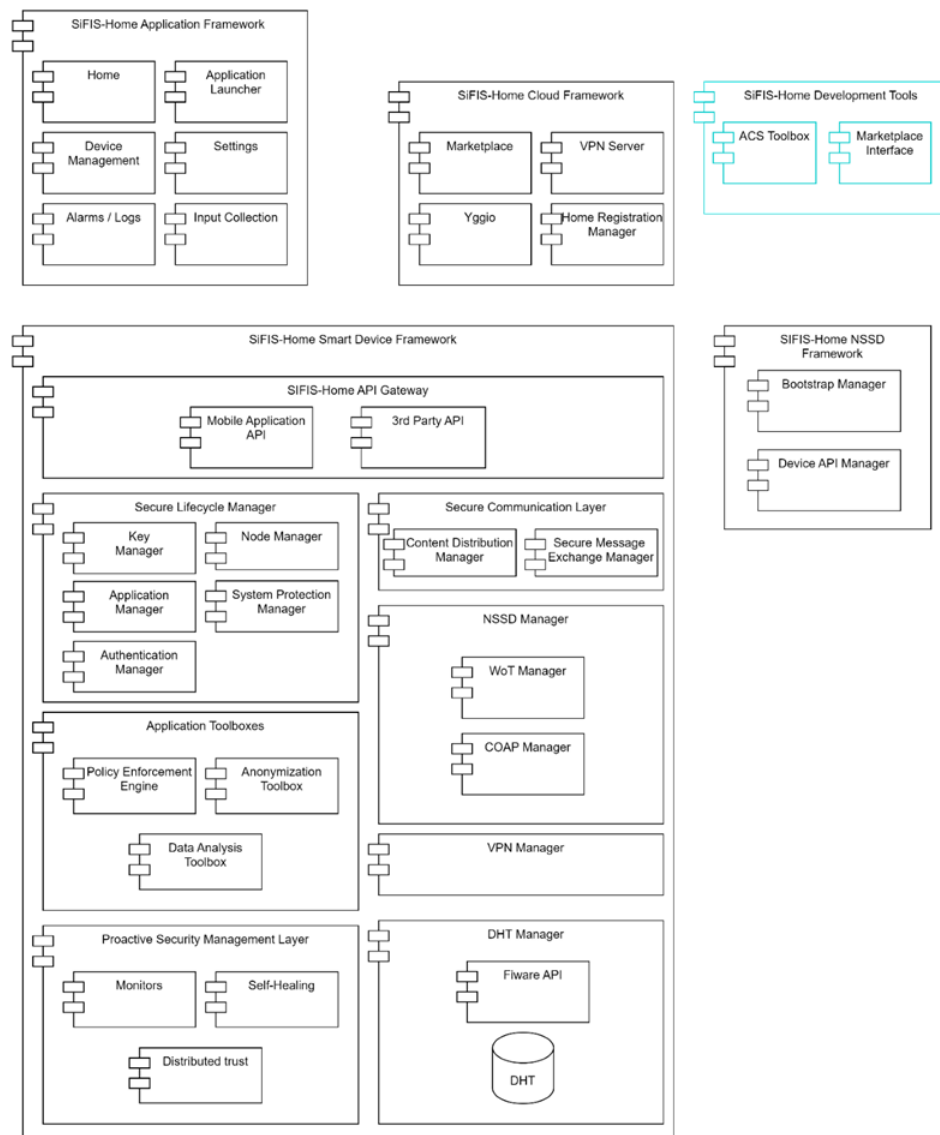


Figure 5: Final high-level architecture of the SIFIS-Home framework as defined in D1.4

The high-level architecture of the SIFIS-Home framework defined in deliverable D1.4 is the basis for the final version of the testbed design and is shown in Figure 5 for reference. The SIFIS-Home API gateway hosts the Mobile Application API and the API for 3rd party applications, and it is only used from the SIFIS-Home mobile application and 3rd party applications from inside the house. An item not obviously clear in the architecture is that the cloud interface interacts to a large extent with the DHT via the standard protocol MQTT with TLS 1.3 encryption instead of going through the SIFIS-Home API gateway. The reason MQTT is used is to be able to securely communicate through the firewall protected the Cyber Security perimeter, this works since MQTT is a publish/subscribe protocol so every session will originate from inside the house.

4.2 Allocation of components

The SIFIS-Home architecture is the representation of the devices and actors interacting with the SIFIS-Home Framework (see deliverable D1.4 and D5.2). In order to design and later implement the testbed, we first found out all about the partners' capabilities, experience, commercial interests, and availability of legacy codes, tools and devices. Then, from matching the partners' expertise with the architecture and requirements of the SIFIS-Home framework, it enabled us to identify the optimal

allocation of the modules and components to the partners to be able to realize both the emulated and the physical testbed. It was decided early that the emulated and physical testbed should look alike, with the only difference being the connection of either physical devices or emulated devices to the SIFIS- Home network.

Component / Subcomponent / Module	Partner	Notes
User Interface		Cloud interface / Mobile App
Home	SEN / RIOTS	Cloud interface / Mobile App
Device Management (Android App, show list of devices and turn on/off)	SEN / RIOTS	Cloud interface / Mobile App
Settings	SEN / RIOTS	Cloud interface / Mobile App
Alarm / Logs (Android App, show notification on user relevant events)	SEN	Cloud interface only
Marketplace	SEN	Cloud interface only
FIWARE API	SEN	FIWARE Context Broker Cloud interface
Input Collection	DOMO	Component that allows to capture audio and images from the microphones and cameras of the user smartphone.
Policy Manager	POL	Application Framework
Secure Lifecycle Manager		
Application Manager	CNR	Code in-lining routines
Node Manager	CNR	Set of functionalities used by DHT-Distributed hash table
Authentication Manager	RISE	Set of functionalities and protocols
Key Manager	RISE	Set of functionalities and protocols
System Protection Manager	CNR	Set of functionalities
Secure Communication Layer		
Secure Message Exchange Manager	RISE	Set of functionalities and protocols
Content Distribution Manager	RISE	Set of functionalities and protocols
NSSD Manager		
CoAP Manager	RISE	Pub-sub-based command interface
WoT Manager	LUM/DOMO /RIOTS	It is an application that receives commands from the DHT and forwards them to WoT enabled devices. It also updates the state of WoT enable devices on the DHT.
Proactive Security Management Layer		
Monitors (Kernel / DHT)	CNR	Set of functionalities
Monitor (Network)	F-Secure	Set of functionalities
Distributed Trust	CNR	Set of functionalities
Self-Healing	CNR	Set of functionalities
Application Toolboxes		

Data Analysis Toolbox	CNR	Set of analysis functionalities and tools for data format management
Pre-Processing Layer/Post-Processing	DOMO	Software component that takes care to change the format of data coming from the devices such that it can easily be used by the analytics applications.
Behavioral Analysis (Faulty Device Detection)	CNR	Tools for data analysis
Physical Analysis (Parental Control)	CNR	Real time network monitor and analyzer
Physical Analysis (Face Recognition)	CNR	Tools for data analysis
Physical Analysis (Object Detection)	CNR	Tools for data analysis
Application Analysis (Software Intrusion Detection)	CNR	Monitoring and storage
Network Analysis (DOS\DDOS Detection)	CNR	Monitoring and storage
Multi-level Intrusion Detection	CNR	Container of analysis functionalities
Multimedia Analysis (Speaker verification through voice analysis)	CNR	Tools for audio data analysis
Multimedia Analysis (Speaker identification through voice analysis)	CNR	Tools for audio data analysis
Multimedia Analysis (Speech to text for managing smart-home components)	CNR	Tools for audio data analysis
Multimedia Analysis (Voice anonymization)	CNR	Tools for audio data anonymization
Multimedia Analysis (Anomaly detection in audio signal analysis)	CNR	Tools for audio data analysis
Multimedia Analysis (Parental control analyzing audio signal:)	CNR	Tools for audio data analysis
Behavioral Analysis (Activity Monitoring)	RIOTS	Tools for data analysis
Network Analysis (Home Network Edge Device)	CENTRIA	Tools for data analysis
Network Analysis ()	F-Secure	Tools for data analysis
IoT Anomaly detection (xAnomaly)	SEN	Tools for sensor data analysis
Anonymization Toolbox	CNR	Anonymization functionality
Policy Enforcement Engine	CNR	Enforcement functionalities
SIFIS-Home API Gateway		
Mobile Application API	Centria	Joining of device, view status, send commands.
3 rd party API	LUM / POLITO	E
Integration components		
DHT manager	DOMO	Set of APIs exposed by a Web Service that allow a) adding a new SD to a SIFIS-Home system, b) adding new users.
NSSD interface for joining	DOMO	Set of APIs provided by NSSDs through which it is possible to send them joining

		information
DHT-MQTT integration component	DOMO	Software component that allows exchanging messages between the SIFIS-Home DHT and the Yggio MQTT broker.
DHT-KeyCloak integration component	CNR, DOMO	Software component that allows user-related data synchronization between the DHT and KeyCloak
WoT Discovery component	LUM, DOMO	Software component that allows discovering WoT-enabled devices
Policy Creation Component (Web service, connect via OAuth)	POLITO	The Application Manager allows installing third-party SIFIS-Home applications on a SD. The Marketplace component is a cloud service providing the list of available third-party applications that can be installed on an SD.
Testbed - Simulated devices (Docker containers in virtual server)		CI / CD Setup
SIFIS-Home Smart Device MultiArch (x86, ARM - Manually Deployment, ansible?)	RIOTS	Initial Deployment
SIFIS-Home Smart Device MultiArch - Auto Deployment (CI system, Docker watch tower)	RIOTS	Continuous deployment
SIFIS-Home Smart Devices Emulated (Docker compose config image)	RIOTS	Emulated SIFIS-Home Smart Device
SIFIS-Home Smart Device Raspberry PI (Docker compose config image)	RIOTS	Physical SIFIS-Home Smart Device

Table 1 Component allocation per partner

The final component allocation list is available in Table 1. At the time of preparing deliverable D5.3 about 85% of the total implementation had been completed. Notably, the components considered essential for executing the user scenarios were more complete, such as:

- Cloud interface incl. Home, Device Management and Settings
- Node manager
- WoT manager
- Data analysis toolbox
- Mobile Application API
- 3rd party API
- DHT Manager
- NSSD Interface for joining
- DHT - MQTT integration
- WoT discovery
- Deployment of emulated and physical device

Most of the components can be implemented directly in the SIFIS-Home architecture but some have specific integration requirements in order to create a fully working SIFIS-Home system.

4.3 Definitions of testbeds

In SIFIS-Home there are two different testbeds in WP5 plus the pilot in WP6 to validate and verify the implementation of the security architecture in the SIFIS-Home framework. In WP5 we name the testbeds the “emulated” testbed and the “simulated” testbed. All testbeds are being built around the Panarea server as the central server. Since available hardware is different depending on the environment of the testbed, not all components can be executed in each testbed. Table 2 below defines the current list of components and modules that are planned to be executed in each testbed.

Component / Subcomponent	Partner	Simulated (WP5 Raspberry Pis)	Emulated (WP5 Virtual device)	Pilot (WP6)
User Interface	SEN / RIOTS			
- Home	SEN / RIOTS	Yes / Yes	Yes / Yes	Yes / Yes
- Device Management (Android App, show list of devices and turn on/off)	SEN / RIOTS	Yes / Yes	Yes / Yes	Yes / Yes
- Settings	SEN / RIOTS	Yes / Yes	Yes / Yes	Yes / Yes
- Alarm / Logs (Android App, show notification on user relevant events)	SEN / RIOTS	Yes / Yes	Yes / Yes	Yes / Yes
- Marketplace	SEN	Yes / Yes	Yes / Yes	Yes / Yes
- FIWARE API	SEN	Yes	Yes	Yes
- Input Collection (voice / face)	DOMO / CNR	Yes / Yes	Yes / Yes	No / No
- Policy Manager (webservice running in SIFIS-Home framework)	POL	Yes	Yes	Yes
Secure Lifecycle Manager	CNR			
- Application Manager	CNR	Yes	Yes	Yes
- Node Manager	CNR	Yes	Yes	Yes
- Authentication Manager	RISE	Yes	Yes	Yes
- Key Manager	RISE	Yes	Yes	Yes
- System Protection Manager	CNR	Yes	Yes	No
Secure Communication Layer	RISE			
-Secure Message Exchange Manager	RISE	Yes	Yes	Yes
- Content Distribution Manager	RISE	Yes	Yes	Yes
NSSD Manager	DOMO	No	No	Yes
- CoAP Manager	RISE	Yes	Yes	Yes
- WoT Manager	LUM/DOMO/RIO TS	Yes	Yes	Yes
Proactive Security Management Layer	CNR			
- Monitors (Kernel / DHT)	CNR	No / Yes	Yes	No / Yes
- Monitor (Network)	F-Secure	Yes	Yes	Yes
- Distributed Trust	CNR	Yes	Yes	Yes
- Self-Healing	CNR	Yes	Yes	Yes
Application Toolboxes				

- Data Analysis Toolbox (frontend)	CNR	Yes	Yes	Yes
-- Behavioral Analysis (Faulty Device Detection)	CNR	No	Yes	No
-- Physical Analysis (Parental Control)	CNR	Yes	Yes	No
-- Physical Analysis (Face Recognition)	CNR	Yes	Yes	No
-- Physical Analysis (Object Detection)	CNR	Yes	Yes	No
-- Application Analysis (Software Intrusion Detection, 3rd party analysis)	CNR	No	Yes	No
-- Network Analysis (DOS\DDOS Detection)	CNR	No	Yes	No
-- Multi-level Intrusion Detection	CNR	No	Yes	No
-- Multimedia Analysis (Speaker verification through voice analysis)	CNR	No	Yes	No
-- Multimedia Analysis (Speaker identification through voice analysis)	CNR	No	Yes	No
-- Multimedia Analysis (Speech to text for managing smart-home components)	CNR	No	Yes	No
-- Multimedia Analysis (Voice anonymization)	CNR	No	Yes	No
-- Multimedia Analysis (Anomaly detection in audio signal analysis)	CNR	No	Yes	No
-- Multimedia Analysis (Parental control analyzing audio signal:)	CNR	No	Yes	No
-- Behavioral Analysis (Activity Monitoring)	RIOTS	No	Yes	No
-- Network Analysis (Home Network Edge Device)	CENTRIA	Yes	Yes	No
-- Network Analysis ()	F-Secure	Yes	Yes	Yes
-- IoT Anomaly detection (xAnomaly)	SEN	No	Yes	No
- Anonymization Toolbox	CNR	Yes	Yes	Yes
- Policy Enforcement Engine	CNR	Yes	Yes	Yes
SIFIS-Home API Gateway				
Mobile Application API (Rest API)	Centria	Yes	Yes	Yes
3rd party API (RPC)	LUM / POLITO	Yes	Yes	Yes
Integration components				
DHT manager	DOMO	Yes	Yes	Yes
NSSD interface for joining	DOMO	Yes	No	Yes
DHT-MQTT integration component	DOMO	Yes	Yes	Yes
DHT-KeyCloak integration component	CNR, DOMO	Yes	Yes	Yes

Table 2 Definition of components and modules to execute per testbed.

The emulated testbed will be completely hosted on the Panarera server and consists of the virtual SIFIS-Home smart devices on the x86 hardware architecture.

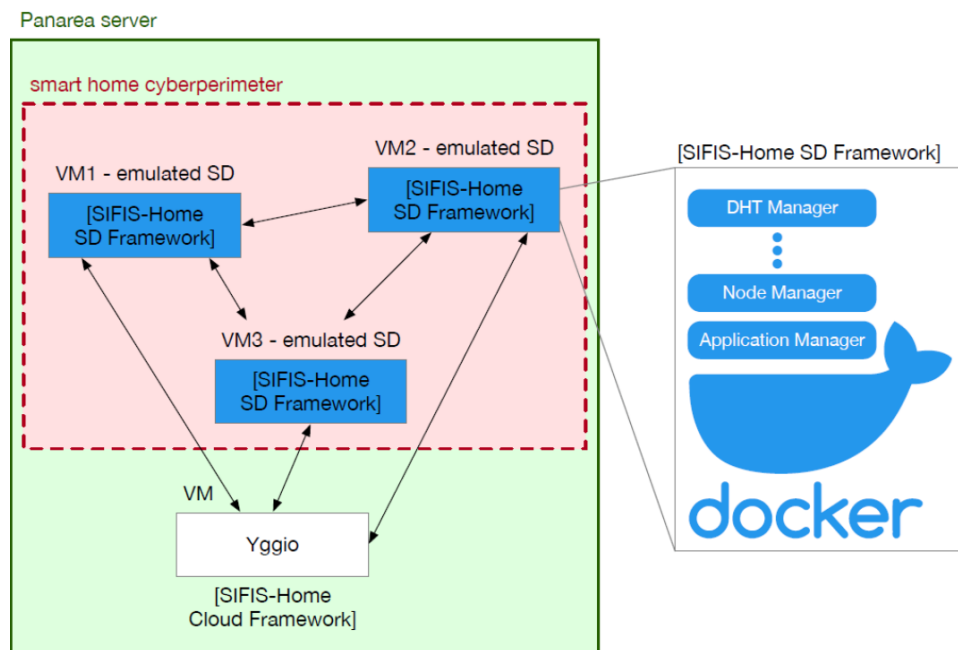


Figure 6 Details of the emulated testbed

On each of these virtual devices in the emulated testbed, we expect to execute the complete SIFIS-Home framework except the “NSSD interface for joining”. The reason the “Not so smart device” - “NSSD interface for joining” - cannot be run is because it requires a physical, not-so-smart device to be connected to a smart device, and because emulated devices are virtual devices, it cannot be done.

The simulated testbed is being built using Raspberry PI devices with ARM hardware architecture. Since not all components are available on ARM architecture, especially in the analytics area where many components are dependent on TensorFlow AI/ML framework [TENSORFLOW], fewer components and modules from the SIFIS-Home framework will be executed in the simulated testbed.

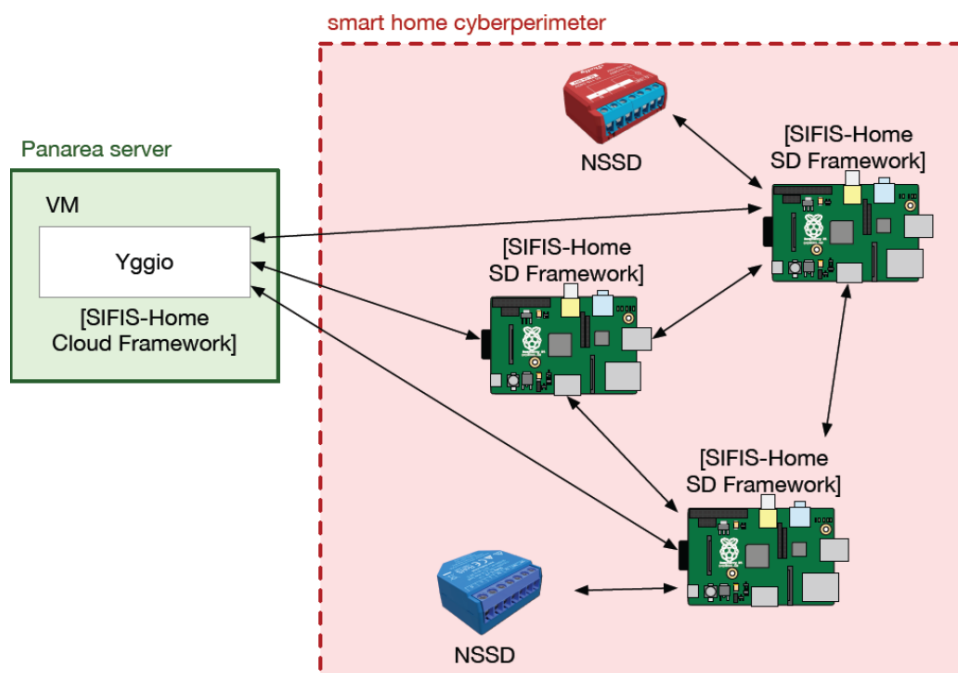


Figure 7 Details of the simulated testbed

4.4 Component implementation and integration highlights

The following section describes implementation considerations for each main building block.

- **User Interface:** this component provides the Graphical User Interfaces to all the different kinds of users that use the SIFIS-Home Framework. There are two user interfaces:
 1. **Mobile Application:** this is a mobile application that uses the SIFIS-Home framework component called “API gateway” to connect directly to the SIFIS-Home framework inside the house. Via the Mobile application the end users can then do basic management of the SIFIS-Home network.
 2. **Cloud interface:** this is built upon Sensative Yggio platform [YGGIO] with the secure FIWARE Context Broker Ratatosk that in its turn uses the open source KeyCloak component for access and authentication. The cloud interface can be used to interact with the SIFIS-Home framework both within the house and outside the house. Since the FIWARE Context Broker does not implement all the APIs required by the cloud interface UI, Yggio’s legacy background proprietary API’s as well as the KeyCloak security APIs are used as well, in order to make it all work well in the testbed.
- **DHT Manager – Distributed hash table:** this component provides a completely distributed publish/subscribe mechanism through which SIFIS-Home applications can exchange messages. The SIFIS-Home DHT allows publishing both persistent and volatile messages. Persistent messages need to be stored in a persistent way, so that they are available even after a node reboot operation. In detail, the persistent messages are stored inside a 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 reader can refer to D1.4 and D6.2 for additional details about the DHT implementation.

The SIFIS-Home framework implements a component called “DHT-MQTT Integration component” that is used to pass data securely through the Cybersecurity perimeter and to communicate with the cloud interface. The cloud interface then provides a FIWARE API interface through which it is possible to access the DHT data. In detail, every DHT topic is represented as a dedicated device in Yggio and a specific MQTT topic is associated with every device. Hence, the DHT component can update the state of the Yggio device associated to a certain DHT topic by publishing its content to the dedicated MQTT topic.

- **Secure Lifecycle Manager:** this module handles the standard workflow of the SIFIS-Home framework. This module acts as the orchestrator of the framework lifecycle, regulating the presence and behaviour of both Smart Devices and applications. In particular, the Secure Lifecycle Manager enables and handles new device registration and deregistration, as well as management/provisioning of device-to-device keying material and management/enforcement of access rights for device-to-device resource access. Moreover, it manages installation and removal of third-party applications, according to the received instructions from the user or from the policy engine and the intrusion detection system.
- **Application Manager:** this component was developed by CNR. It consists of a docker container which can start, run, stop and remove other containers and, in order to do that, it needs the support of the host machine otherwise it will not have the appropriate permissions.

The Application Manager receives directives from a sender through DHT and it forwards them to the host machine which takes care of starting, running, stopping or removing the container specified in the directive forwarded by the application manager container that waits until the completion of the operation. At the end of the operation, the host machine communicates the outcome of the operations specified in the directive to the application manager, which forwards this information to the original sender.

- **System Protection Manager:** The system protection manager acts as an interface between the Data Analysis Toolbox and, respectively, the Node Manager and the Application Manager. More precisely, the system protection manager triggers the continuous analytics relevant to perform intrusion detection and is triggered by their responses. Once an intrusion is detected, the System Protection Manager triggers the Node Manager or the Application Manager to stop and remove malicious applications or smart devices.
- **The Node Manager** component of the Secure Lifecycle Manager consists of three components: the first component is implemented as a Rust library crate and provides the core functionalities. The second component provides a component-local testbed for manual component testing and validation in addition to the present unit tests of the first component. The third component that is in development provides the integration with the DHT and communicates with the first component. The Node Manager provides the shared keys used for DHT communication. Voting procedures ensure that consensus of nodes is reached on removal decisions, in accordance with input received from the System Protection manager component. We provide the link to the implementation at [NODE-MANAGER-CODE].

[NODE-MANAGER-CODE] <https://github.com/sifis-home/node-manager>

The following security solutions developed in WP3 pertain to the “Security Lifecycle Manager” module and a link to their implementation from RISE is also provided. These implementations build on the open-source Eclipse Californium CoAP framework available at [CALIFORNIUM], which provides the CoAP protocol and the OSCORE security protocol. A single codebase collecting these implementations is accessible at [WP3-CODEBASE], as available for use, integration and testing within the SIFIS-Home project and especially in the interest of the WP5 testbed.

[CALIFORNIUM] <https://github.com/eclipse/californium>

[WP3-CODEBASE] <https://github.com/sifis-home/wp3-solutions>

- **OSCORE profile of the ACE framework**, as documented in Section 5.1 of deliverable D3.2. Together with the main ACE framework, the implementation is available at <https://bitbucket.org/marco-tiloca-sics/ace-java>
- **Key provisioning for Group OSCORE communication using the ACE framework**, as documented in Section 6.1 of deliverable D3.2. Together with the main ACE framework, the implementation is available at <https://bitbucket.org/marco-tiloca-sics/ace-java>
- **EDHOC key establishment**, as documented in Section 6.3 of deliverable D3.2, including specific profiling for CoAP and OSCORE. The implementation is available at <https://github.com/rikard-sics/californium/tree/edhoc>

- **Secure Communication Layer:** this module is responsible for the secure communication between the devices of the SIFIS-Home architecture. The testbed should be able to cover multiple scenarios of the Secure Communication Layer. The testbed serves as an environment for testing the efficiency of the solution to be created and it must be a controlled environment where actions are carried out in a secure and isolated manner, without influencing the process being controlled. The secure communication layer includes mechanisms to ensure standard encryption features on the communication associations to be used among the devices. In particular, the component handles security for communication between Smart Devices and among Smart Devices and NSSDs. In the implemented testbed, we will consider different communication models, implementing different possible topologies: in particular, we are going to consider a first emulated testbed where all devices are supposed to be connected through a Wi-Fi access point and another configuration where devices will be connected via Ad-Hoc Wi-Fi or an alternative point-to-point physical protocol. This last configuration will be particularly needed to test the DHT-based routing of SIFIS-Home messages among Smart Devices and to model possible issues of network partitioning.

The following security solutions developed in WP3 pertain to the “Secure Communication Layer” module and a link to their implementation from RISE is also provided. These implementations build on the open-source Eclipse Californium CoAP framework available at [CALIFORNIUM], which provides the CoAP protocol and the OSCORE security protocol. A single codebase collecting these implementations is accessible at [WP3-CODEBASE], as available for use, integration and testing within the SIFIS-Home project and especially in the interest of the WP5 testbed.

[CALIFORNIUM] <https://github.com/eclipse/californium>

[WP3-CODEBASE] <https://github.com/sifis-home/wp3-solutions>

- **Group OSCORE**, as documented in Section 4.1 of deliverable D3.2. The implementation is available at https://github.com/rikard-sics/californium/tree/group_oscore
- **OSCORE profile of the ACE framework**, as documented in Section 5.1 of deliverable D3.2. Together with the main ACE framework, the implementation is available at <https://bitbucket.org/marco-tiloca-sics/ace-java>
- **Key provisioning for Group OSCORE communication using the ACE framework**, as documented in Section 6.1 of deliverable D3.2. Together with the main ACE framework, the implementation is available at <https://bitbucket.org/marco-tiloca-sics/ace-java>
- **EDHOC key establishment**, as documented in Section 6.3 of deliverable D3.2, including specific profiling for CoAP and OSCORE. The implementation is available at <https://github.com/rikard-sics/californium/tree/edhoc>
- **NSSD Manager:** the SIFIS-HOME NSSD Manager is the SIFIS-HOME component responsible for interacting with the NSSD devices present in the house. It is composed of two different components, the WoT Manager component that manages communication with NSSD devices providing a WoT API, and the CoAP Manager component that manages communication with CoAP devices. A detailed description of the WoT Manager component is reported in deliverable D6.2. In the following, we briefly describe the CoAP Manager component.

The “CoAP Manager” component of this module has been developed in order to seamlessly

enable the use of CoAP communication within the SIFIS-Home architecture.

Consistently with the early integration plan defined in Section 4.3 of deliverable D5.2, this component enables a remote control & output interface for devices acting as CoAP Client (expected to be Smart Devices), which can then interact with devices acting as CoAP Server (expected to be Not So Smart Devices) by using CoAP as per the intended application behavior. Such a remote control & output interface is integrated with the DHT-based pub-sub broker system used in the SIFIS-Home architecture, by specifically relying on WebSocket for issuing commands to (retrieving output from) a CoAP client device.

The implementation of the “CoAP Manager” component and its related functionalities discussed above is available within the same, single codebase collecting the implementations of the security solutions developed in WP3 [WP3-CODEBASE]. The WebSocket communication with the DHT-based pub-sub broker has relied on the Open Source WebSocket library Tyrus, which is available at [TYRUS].

[WP3-CODEBASE] <https://github.com/sifis-home/wp3-solutions>

[TYRUS] <https://projects.eclipse.org/projects/ee4j.tyrus>

- **Application toolboxes:** this component collects related and interconnected sub-components that are all services inside the SIFIS-Home infrastructure.
 1. **Alarm / Log:** The log storage component is stored in the cloud interface since it needs to be managed through a central point. This will enable all SIFIS-Home devices and analytics from any SIFIS-Home network to create log events that can then be viewed and acknowledged in the UI by authorized users in the Smart Home where the log events were generated.
 2. **Analytics toolbox:** the component of the analytics toolboxes will be used by the testbed to support intrusion detection tests. As this testbed is, in fact, focused on verifying the fulfillment of non-functional requirements, the analytic toolbox does not play a key role. Still, the testbed will verify the possibility of correctly invoking the analytics, by using synthetic data for functionality verification.
 3. **Anonymization Toolbox:** this component will be included in the testbed and its functionality verified on top of synthetic data.
- **Proactive security management layer:** This component is responsible for proactively preserving the security properties of the SIFIS-Home infrastructure. Proactivity implies taking preemptive measures before an incident occurs.

The following proactive security measures have been developed as part of SIFIS-Home project and will be deployed on the WP5 testbed:

1. **AUD Manager** is a proactive security measure for IoT anomaly detection in a consumer environment. The AUD manager operates on information from the network and transport layers (L3 & L4), as underlying the session and application layers where vulnerabilities are often exploited, e.g., through session hijacking and man-in-the-middle attacks. The analytic developed in WP4 strives to spot the anomalous network events before data packets reach their destinations, i.e., the target devices.

2. **Manufacturer Usage Description** (MUD, RFC 8520) mimicking support for devices without MUD support from the vendor. MUD is a proposed standard for describing the expected network behavior of an IoT device. The usage description itself is an access control list defined by the vendor. It explicitly defines protocols, ports and endpoints with which a particular IoT device is allowed to communicate with. However, as of today, vendors that provide MUD support are quite uncommon. By leveraging the usage description models created by the AUD manager described above, we can create an access control list for any local IoT device. By enforcing such an access control measure for a home IoT-device, one would proactively block any incoming harmful network intrusion attempts.

4.5 Authorization and Access management integration

The authorization and access management integration challenges concerning the SIFIS-Home framework relate to the fact that devices, users and their access rights need to be handled inside the home and in the Yggio cloud interface. The key issue is deciding how to synchronize the states of users and devices so that the SIFIS-Home policy enforcement engine inside the home can decide what activities each user can do in each device.

The cloud interface uses the well-known open-source component KeyCloak for access and authorization management, which is a well proven industry standard component that meets the security requirements for a secure authentication component. Within the SIFIS-Home network, we use DHT (Distributed Hash Table) for both device and user management. Hence, the key access and authentication integration activity required is to synchronize users by letting the SIFIS-Home DHT use the cloud interface KeyCloak authentication mechanism to retrieve a valid JWT - JSON web token. The JWT token then gives the SIFIS-Home framework full access to the cloud interface and is able to synchronize users, access rights and devices.

- Users, access and authorization: The SIFIS-Home DHT will, after it receives a valid JSON web token for the new SIFIS-Home networks admin user, “Create the Smart Home” by activation of the home admin user, and then create the default Smart Home users in the cloud interface KeyCloak component via its API. This is possible since the cloud interface is IP addressable from inside the SIFIS-Home network. Once users are created, depending on their roles, the policy enforcement engine in the SIFIS-Home framework will be responsible for determining what activities each user is allowed and not allowed to execute. This is the DHT to KeyCloak integration component.
- Devices: In order to achieve secure communication across the Cyber-Perimeter, we decided to start with using MQ Telemetry Transport (MQTT) with TLS 1.3 and to utilize the publish/subscribe mechanism to synchronize the devices between FIWARE Context Broker and the DHT within the Smart Home. MQTT communication can pass through the firewall and makes it feasible to use the UI on the API gateway to control both Smart Devices and Not So Smart Devices inside the SIFIS-Home network. This is the DHT to MQTT integration component.

5 Interfaces of the testbed

5.1 Mobile Application API

The Mobile app gets the API key needed to use the Mobile API from the QR code that comes with the device. This key allows the device to be configured on the home network.

After initial configuration, the Smart Device is connected to the SIFIS-Home network using the Yggio service. The Mobile application gets a JWT access token from Yggio. The JWT access token is signed by Yggio and is a Base64 encoded string containing the username of the user that requested the token and its expiration date.

Once the mobile application gets the access token, it sends it to the Mobile API component to access the DHT. Then, the Mobile API component verifies that the token is valid (by using the public key of Yggio) and retrieves the username from the token. Then, the mobile application allows access to the SIFIS-Home framework if the token is valid.

After these steps, the Mobile API component behaves like a proxy towards the DHT.

5.2 *WebOfThings Smart Devices*

The WebOfThings standards focus on the key concept of enabling the devices to self-describe. The Thing Description is the cornerstone of it. Within SIFIS-Home, we augment the Thing Description with a custom ontology to deliver information regarding the risks involved in interacting with the device. We developed a from-scratch Rust implementation of WoT 1.1 Thing Description, Discovery and the HTTP SSE Profile and used it to implement simulated devices. The WoT devices integrate to the SIFIS-Home network via the DHT.

5.3 *Ratatosk, the FIWARE Context Broker*

FIWARE NGSI v2 [FIWARE, 2021] Ratatosk is a publish/subscribe Context Broker that holds a state of a system via FIWARE entities. The Context Broker implements FIWARE NGSI v2 APIs as defined here <https://swagger.lab.fiware.org/>. 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 house that is instead built upon DHT (Distributed hash table) 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.

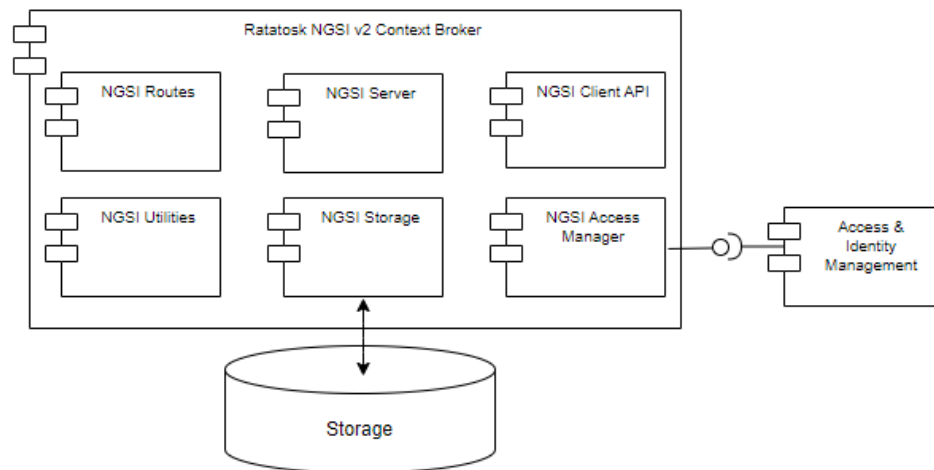


Figure 8: Ratatosk FIWARE Context Broker architecture

An important distinction with Ratatosk is that it 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 a user in the Smart Home who attempts to perform some actions has the required authorization to do so. The following are some Rest API examples of how to interface the Context Broker which will be used in the project:

Command	Modify	Syntax
Update description of existing node	<id> <token> hex string <user id> Yggio user id	curl -sS -X POST <a href="https://yggio.sifis-home.eu/ngsi/v2/entities/<id>/attrs?type=Device">https://yggio.sifis-home.eu/ngsi/v2/entities/<id>/attrs?type=Device -H "Fiware-UserToken:<token>" -H "content-type: application/json" -H "fiware-service: yggio" -H "fiware-servicepath: /" -H "fiware-userid: <user id>" -d @- <<EOF <pre>{ "description": "My new description - FIWARE" }</pre> EOF
Create entity	<token> hex string <id> 24 char hex <type> string	curl -sS -X POST https://yggio.sifis-home.eu/ngsi/v2/entities -H "Fiware-UserToken: <token>" -H 'Content-Type: application/json' -d @- <<EOF <pre>{ "id": "<id>", "type": "<type>", "name": { "value": "myNewRoom", "type": "String" }, "temperature": { "value": 23, "type": "Float" }, "humidity": { "value": 45, "type": "Float" } }</pre> EOF
Update values of existing entity	<token> hex string <id>	curl -sS -X POST <a href="https://yggio.sifis-home.eu/ngsi/v2/entities/<id>/attrs">https://yggio.sifis-home.eu/ngsi/v2/entities/<id>/attrs -H "Fiware-UserToken: <token>" -H 'Content-Type: application/json' -d @- <<EOF <pre>{ "temperature": { "value": 28, "type": "Float" }, "humidity": { "value": 48, "type": "Float" }, "lux": { "value": 1245, "type": "Float" } }</pre> EOF
Get entities of certain type	<token>	curl <a href="https://yggio.sifis-home.eu/ngsi/v2/entities?type=<type>">https://yggio.sifis-home.eu/ngsi/v2/entities?type=<type> -H "Fiware-UserToken: <token>"
Get entities that fulfill "q query"	<token> <root key> <key> <value>	curl " <a href="https://yggio.sifis-home.eu/ngsi/v2/entities?type=Device&q=<root key>.<key><condition>%27<value>%27">https://yggio.sifis-home.eu/ngsi/v2/entities?type=Device&q=<root key>.<key><condition>%27<value>%27 " -H "Fiware-UserToken: <token>" -H "fiware-service: yggio" -H "fiware-servicepath: /"

Create NGSI subscription	<token> <id> <url> <type>	curl -sS -X POST https://yggio.sifis-home.eu/ngsi/v2/subscriptions -H "Fiware-UserToken: <token>" -H 'Content-Type: application/json' -d @- <<EOF <pre>{ "description": "Initiate NSGI subscription", "subject": { "entities": [{ "id": <id>, "type": <type> }], "notification": { "http": { "url": <url> } } } }</pre> EOF
List NGSI subscriptions	<token>	curl https://yggio.sifis-home.eu/ngsi/v2/subscriptions -H "Fiware-UserToken: <token>"
Delete NGSI subscription	<id> <token>	curl -sS -X DELETE <a href="https://yggio.sifis-home.eu/ngsi/v2/subscriptions/<id>">https://yggio.sifis-home.eu/ngsi/v2/subscriptions/<id> -H "Fiware-UserToken: <token>"

Table 3 API call examples to Ratatosk FIWARE NGSI v2 multi-tenant and secure context broker

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 house and then execute some command, like turning on a lamp. The natural solution to use NGSI subscriptions will unfortunately not work, since subscription requests are required to be IP addressable and will get blocked by the Smart Home firewall. As described in previous chapters in this document, the solution we identified was to develop a MQTT to DHT bridge, i.e., the devices and analytics in the Smart Home will 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.

5.4 User Interface

The user interface is the main interaction element between the SIFIS-Home framework and the tenants and other users (e.g., administrator, maintainer...). The user interface is used to configure user preferences, interact with GUI capable applications, install and remove applications, as well as set-up usage, safety and security policies.

Since the UI should be reachable both from inside the SIFIS-Home network and externally, the web parts of the UI are provided by the cloud interface and uses mainly the FIWARE Context Broker API and the KeyCloak API to support all the functionality required in SIFIS- Home. From a testbed point of view, we have leveraged existing functionality in Sensative Yggio to be able to implement and get the web UI working well within the scope of the project. The Mobile Application UI interfaces with the API gateway components to give the mobile application access directly to the SIFIS-Home framework.

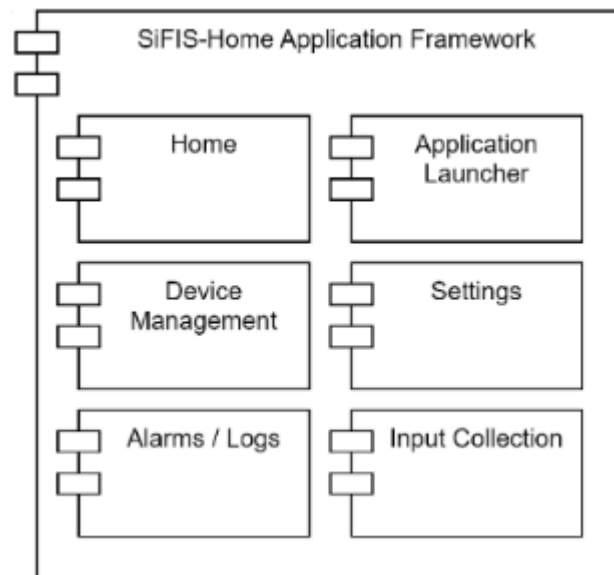


Figure 9: Components of the mobile User Interface module as defined in deliverable D1.4

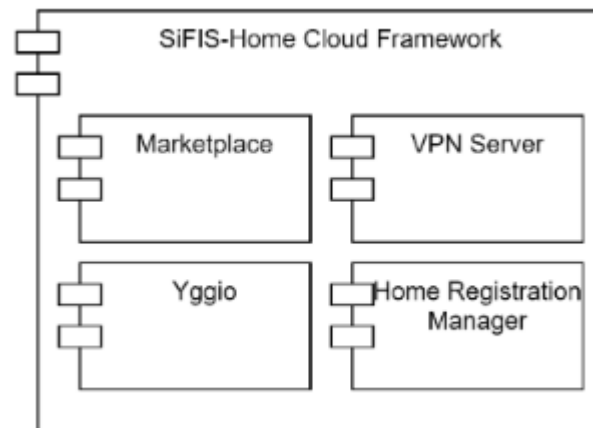


Figure 10: Components of the cloud User Interface module as defined in deliverable D1.4

The diagram in Figure 8 shows the structure of the User Interface component.

5.4.1 Mobile Phone Application Interface

The SIFIS-Home Mobile Application provides a user interface between the end user and the SIFIS-Home framework via the Mobile API gateway.

The Mobile Application lists the devices installed at home and makes it possible to execute different actions on the devices like reading temperature, turning on and off actuators.

Additionally, the Mobile Application makes it possible to view system logs and to install 3rd party applications to the SIFIS-Home framework.

The Mobile Application was written using NativeScript framework and it can run both on Android and iOS with the NativeScript preview application available at <https://preview.nativescript.org/>.

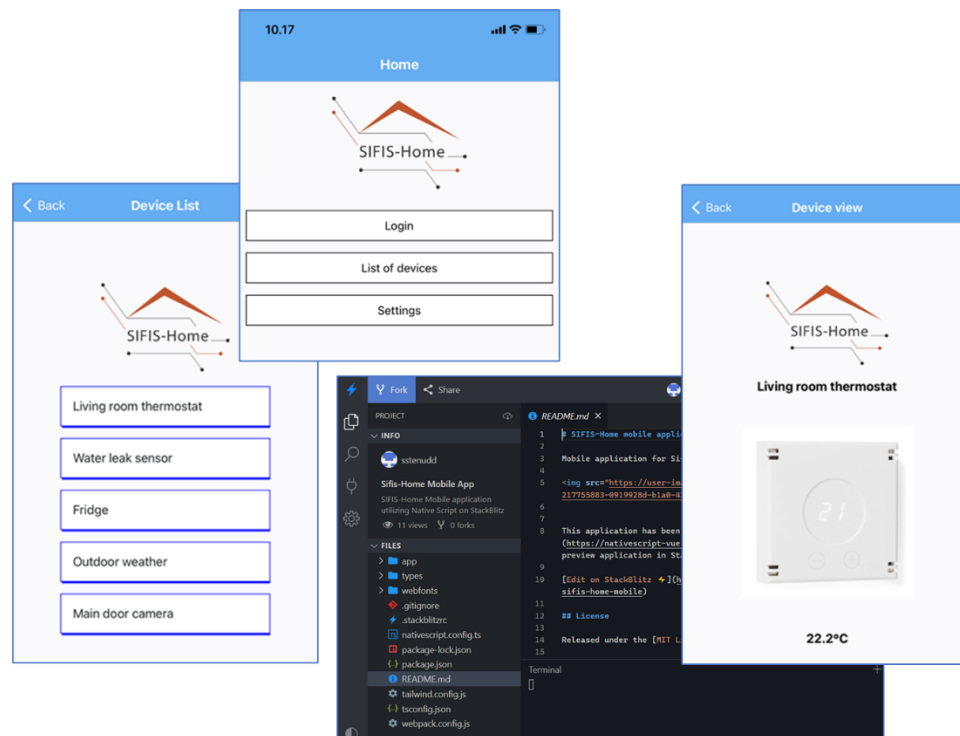


Figure 11 The mobile application user interface

5.4.2 Cloud interface

The cloud interface is built upon Sensative Yggio horizontal IoT integration platform that uses the open-source FIWARE Context Broker Ratatosk as its core component to hold device states and the open-source component KeyCloak for user authentication.

- **Home:** This is the main component of the User Interface and is used to launch other applications installed in the system for a Smart Home. The dashboard shows some key metrics of the system, such as the number of devices and installed third party applications.

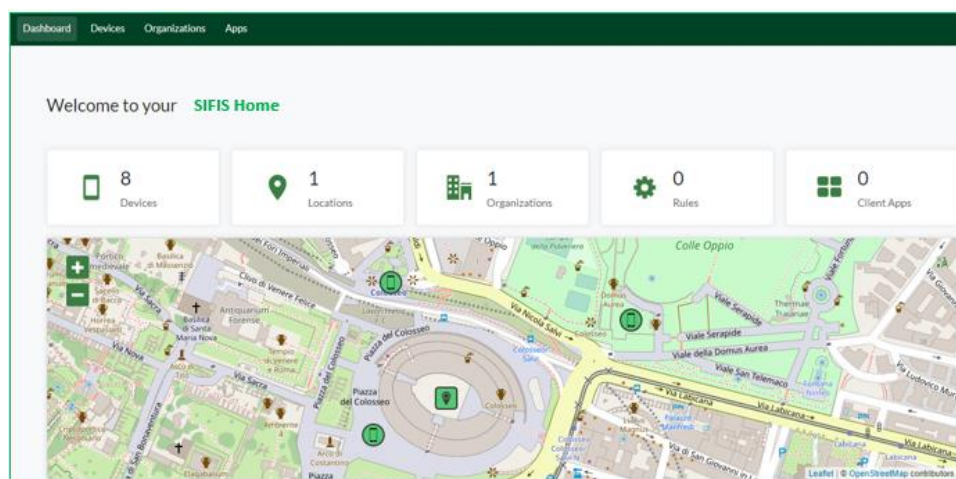
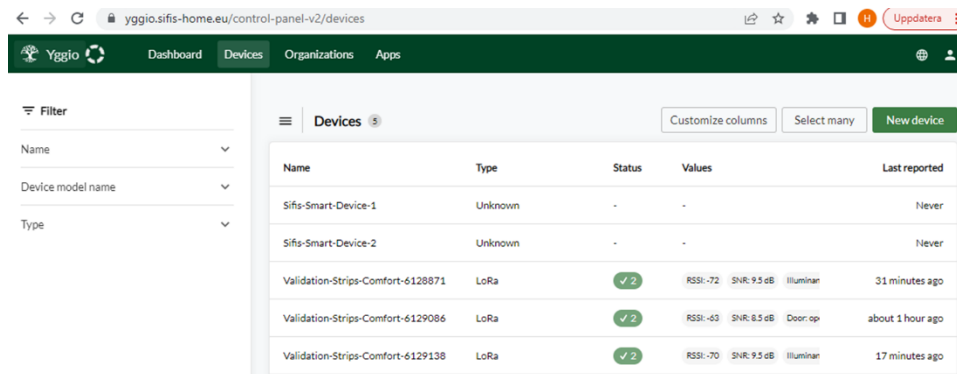


Figure 12: Home screen with a simple dashboard and a map

- **Device management:** This component enables the configuration of the devices in the SIFIS-Home network. This is a core component in the system 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.

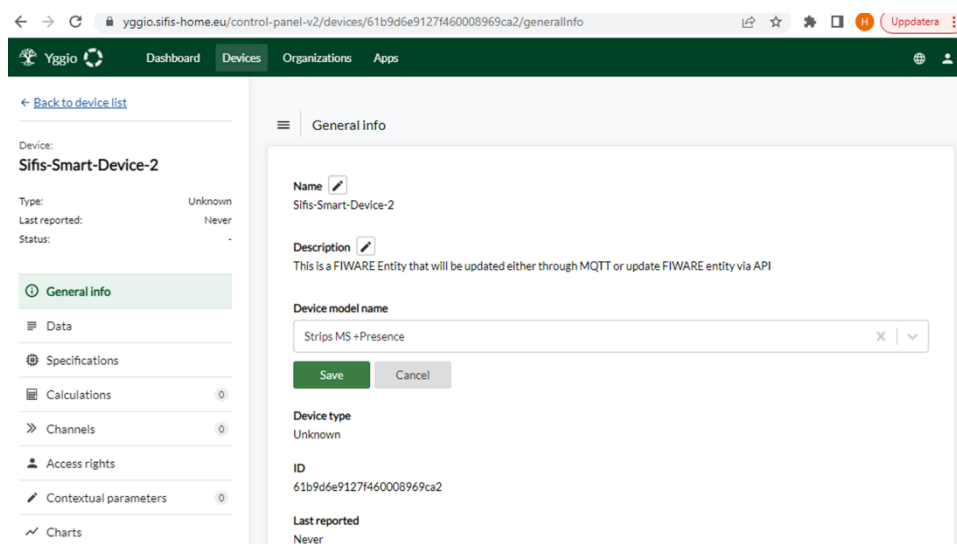


The screenshot shows the 'Devices' page in the Yggio control panel. It features a sidebar with filters for Name, Device model name, and Type. The main area displays a table of devices with columns for Name, Type, Status, Values, and Last reported. The table lists several devices, including 'Sifs-Smart-Device-1', 'Sifs-Smart-Device-2', and three 'Validation-Strips-Comfort' devices with their respective LoRa status, RSSI, SNR, and illuminance values.

Name	Type	Status	Values	Last reported
Sifs-Smart-Device-1	Unknown	-	-	Never
Sifs-Smart-Device-2	Unknown	-	-	Never
Validation-Strips-Comfort-6128871	LoRa	✓ 2	RSSI: -72 SNR: 9.5 dB Illuminan	31 minutes ago
Validation-Strips-Comfort-6129086	LoRa	✓ 2	RSSI: -63 SNR: 8.5 dB Door op	about 1 hour ago
Validation-Strips-Comfort-6129138	LoRa	✓ 2	RSSI: -70 SNR: 9.5 dB Illuminan	17 minutes ago

Figure 13: Device Manager with a list of devices

- Settings:** This component provides user interfaces for the configuration of the SIFIS-Home infrastructure and most items in the cloud interface, like devices, analytics, etc. Each item visible is represented by FIWARE entities in the Ratatosk Context broker. Then, this component provides a viewer and editor of the applicable values related to FIWARE entities.



The screenshot shows the 'General Info' page for the device 'Sifs-Smart-Device-2'. It includes a sidebar with navigation options like General info, Data, Specifications, Calculations, Channels, Access rights, Contextual parameters, and Charts. The main area displays the device's details, including its Name, Description, Device model name (Strips MS + Presence), Device type (Unknown), ID (61b9d6e9127f460008969ca2), and Last reported status (Never). There are 'Save' and 'Cancel' buttons for editing the device information.

Figure 14: View status and edit settings for a SIFIS-Home Smart Device

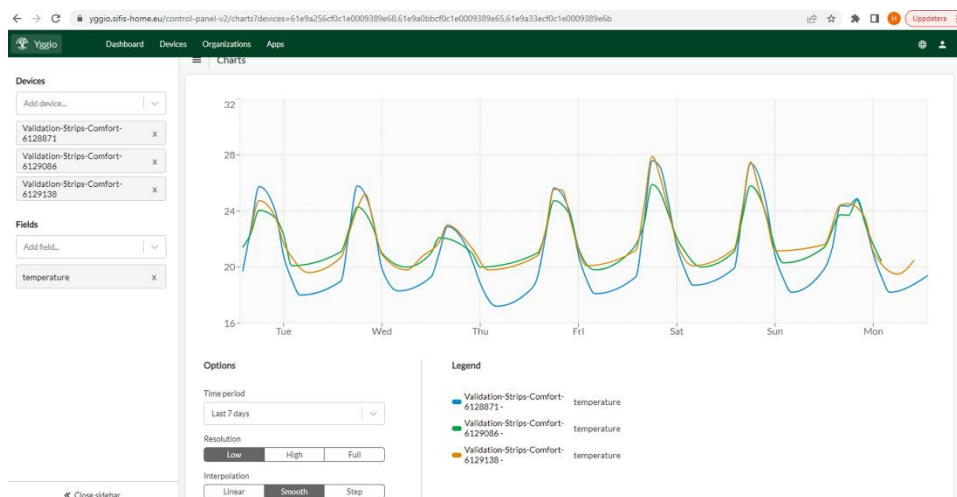


Figure 15: Compare time series data of 3 devices used for validation.

- **Alarms/Log:** This component provides features to show alarms to the user and to gather logs of the functioning of the SIFIS-Home infrastructure. At the time of writing, this component is still under development and is expected to soon be available in the cloud interface.
- **Marketplace:** This component provides graphical user interfaces through which the user can check available 3rd party applications stored in the SIFIS-Home Docker repository at <https://hub.docker.com>. This component is at the time of writing still under development but a first version has been available for some months with some generic applications added to it.

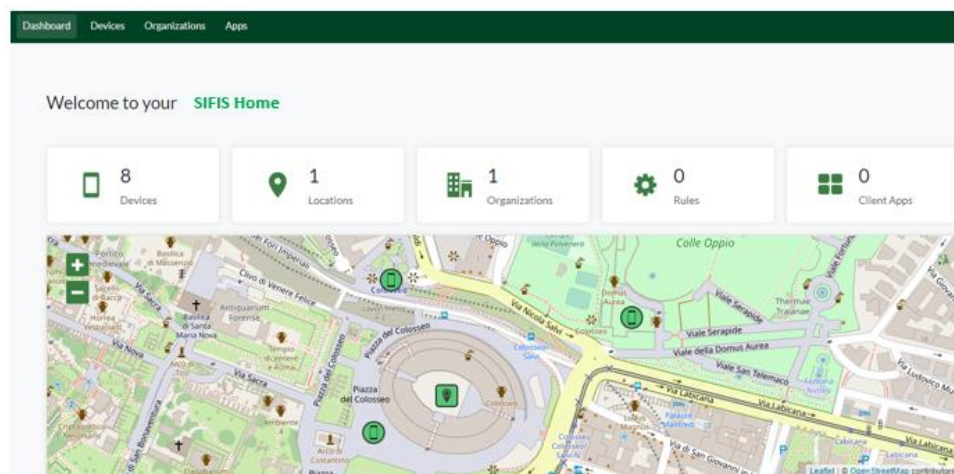


Figure 16 Figure 16: The Marketplace UI that visualizes how applications are visible to the end user.

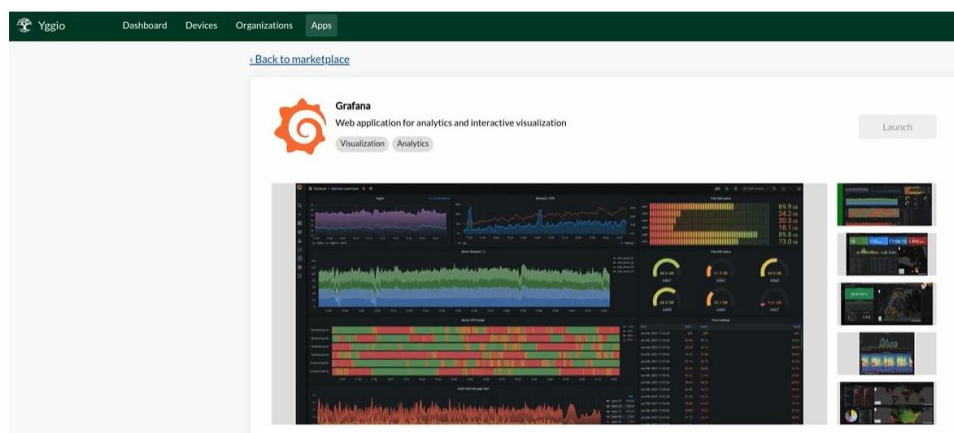


Figure 17: UI how an application is visualized to the user.

- **Organisations:** This component manages access rights and user roles. When creating a new home, one creates an organisation and a new admin user account. Remaining users will be created by the house admin user and assigned to different roles, which will then determine their role in the SIFIS-Home framework enforced by the Policy Enforcement Engine.

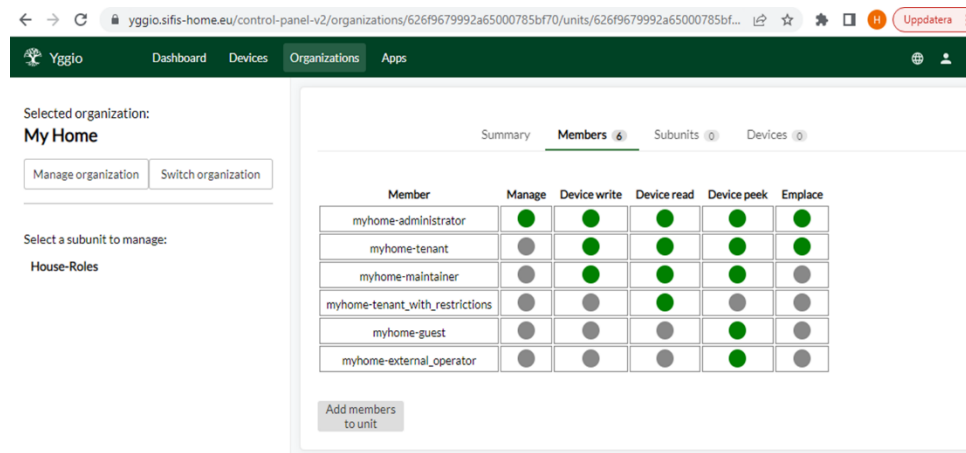


Figure 18: The home main account together with standard users were created.

5.5 GitHub

Most of the code developed in SIFIS-Home has already been uploaded to GitHub and reviewed by the partners as part of the WP2 activities. The SIFIS-generate tool is available to streamline the process of adding GitHub Actions to projects to have a working Continuous Integration.

sifis-generate

This tool generates either a new project for some build systems or configuration files for some Continuous Integration with the use of templates.

Templates define the layout for a project and allow developers to insert data at runtime.

Each template contains all files necessary to build a project with a build system, in addition to Continuous Integration and Docker files used to run tests and implement further checks.

Supported build systems

- ☒ meson
- ☒ poetry
- ☒ maven

Build systems CI files

- ☒ cargo
- ☒ yarn

Figure 19 The SIFIS generate tool

A checklist had been provided to the partners to make them aware of the status of their code and self-assess what is missing.

Code upload checklist

- ☐ Does the project have a `README.md` ?
- ☐ Does the project have a `LICENSE` file?
 - If possible use either [MIT](#) or other compatible licenses.
- ☐ Does the project have a [GitHub Action](#)?
 - ☐ Does it upload the code coverage somewhere?
 - ☐ Does it manage the release of the code? (Continuous Delivery)
- ☐ Does it have a `DockerFile` or does it need it?

Figure 20 Code integration check list

Currently there are 64 code repositories under the SIFIS-Home GitHub organisation. There are many activities listed and many code contributions are made everyday.

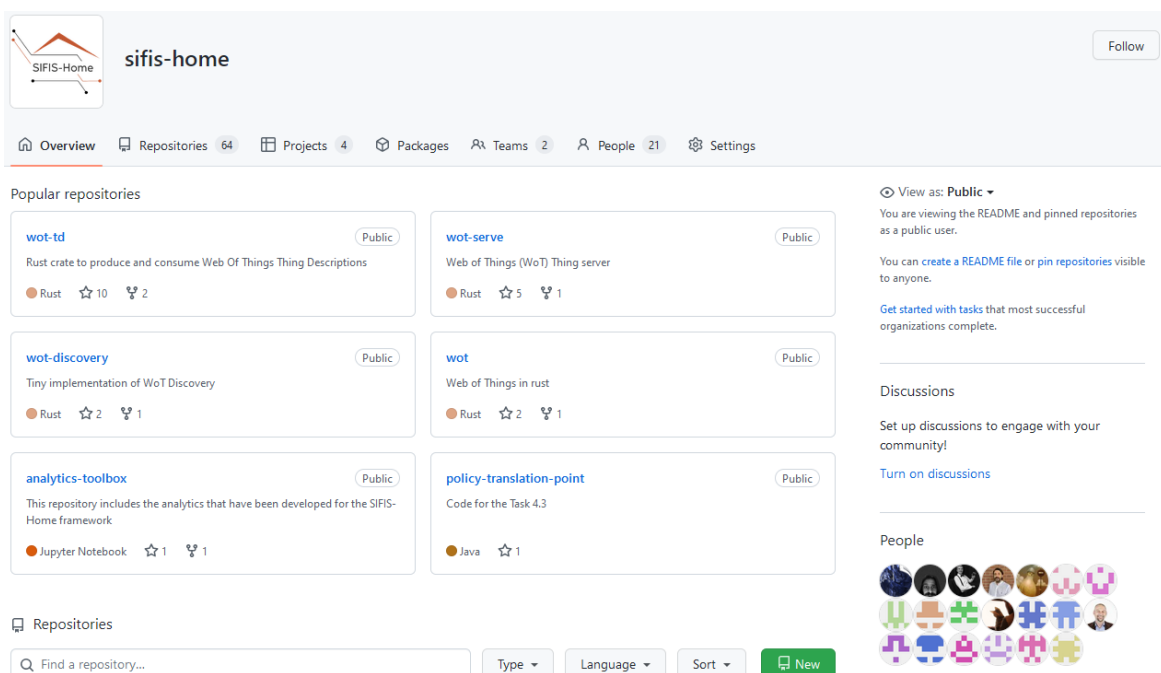


Figure 21 The SIFIS-Home repositories

GitHub links:

The source code for the developed SIFIS-Home components are available in GitHub via the following links:

- FIWARE Context Broker (Ratatosk): <https://github.com/sifis-home/yggio-ratatosk>
- SIFIS-Home Cloud UI (Yggio): <https://github.com/sifis-home/yggio-components>
- The Market Place: <https://github.com/sifis-home/yggio-components/tree/master/control-panel-v2/src/pages/apps>
- Californium (CoAP and OSCORE) <https://github.com/eclipse/californium>
- Secure Message Exchange Manager, Content Distribution Manager, Authentication Manager,

Key Manager, CoAP Manager (Group OSCORE communication, Group OSCORE key provisioning using the ACE framework, ACE OSCORE profile, EDHOC key establishment, DHT-based command/output interface for CoAP clients) <https://github.com/sifis-home/wp3-solutions>

- Network Protection manager: https://github.com/sifis-home/wp5_network_protection_manager.
- Network analysis: https://github.com/sifis-home/wp4-edge_ids/releases
- MUD adaptation and AUD Manager: https://github.com/sifis-home/wp4-aud_manager.
- DHT: <https://github.com/sifis-home/libp2p-rust-dht>.
- Smart Device Mobile API: https://github.com/sifis-home/wp6_mobile_application_api
- WebThings Arduino Library: <https://github.com/WebThingsIO/webthing-arduino>
- WoT Wi-Fi actuators Firmware: <https://github.com/sifis-home/domo-wot-actuator>
- Mobile app: <https://github.com/sifis-home/sifis-home-mobile>
- WoT rust:
 - <https://github.com/sifis-home/wot-td>
 - <https://github.com/sifis-home/wot-serve>
 - <https://github.com/sifis-home/wot-discovery>
 - <https://github.com/sifis-home/demo-things>
- Behaviour testing PoC for demo things: <https://github.com/sifis-home/wot-test>
- SIFIS-Home developer API: <https://github.com/sifis-home/sifis-api>

6 Continuous integration and deployment

The CI (Continuous integration) and CD (Continuous deployment) is implemented based on GitHub actions, Docker MultiArch files, the Docker repository Docker Hub (<https://hub.docker.com>) and a Docker feature called “Watchtower”. Once a Docker MultiArch file, which contains both the x86 and the ARM distribution, is built with the help of GitHub actions, it is uploaded to Docker Hub. The exact steps for GitHub actions to build each file are defined by the partner developing the component, but the last action should always be to move the Docker file to Docker Hub.

The first time a new SIFIS-Home Smart Device should be deployed, some manual interaction is required to install all the microservice Docker containers that the SIFIS-Home framework consist of on the target device. However, once a device is deployed, the Watchtower feature checks every five minutes if any newer version of any of the included Docker containers are available. If yes, then the Watchtower automatically updates the container to get the latest version. Since all SIFIS-Home devices run Watchtower within 5 minutes, all of them will have the same software combination of the different containers. Using Docker Watchtower is an elegant way to use state-the-art tools to complete the most difficult step in the CI/CD tool chain, namely the actual automatic deployment to all devices. Figure 22 below illustrates all the steps in the SIFIS-Home CI/CD process.

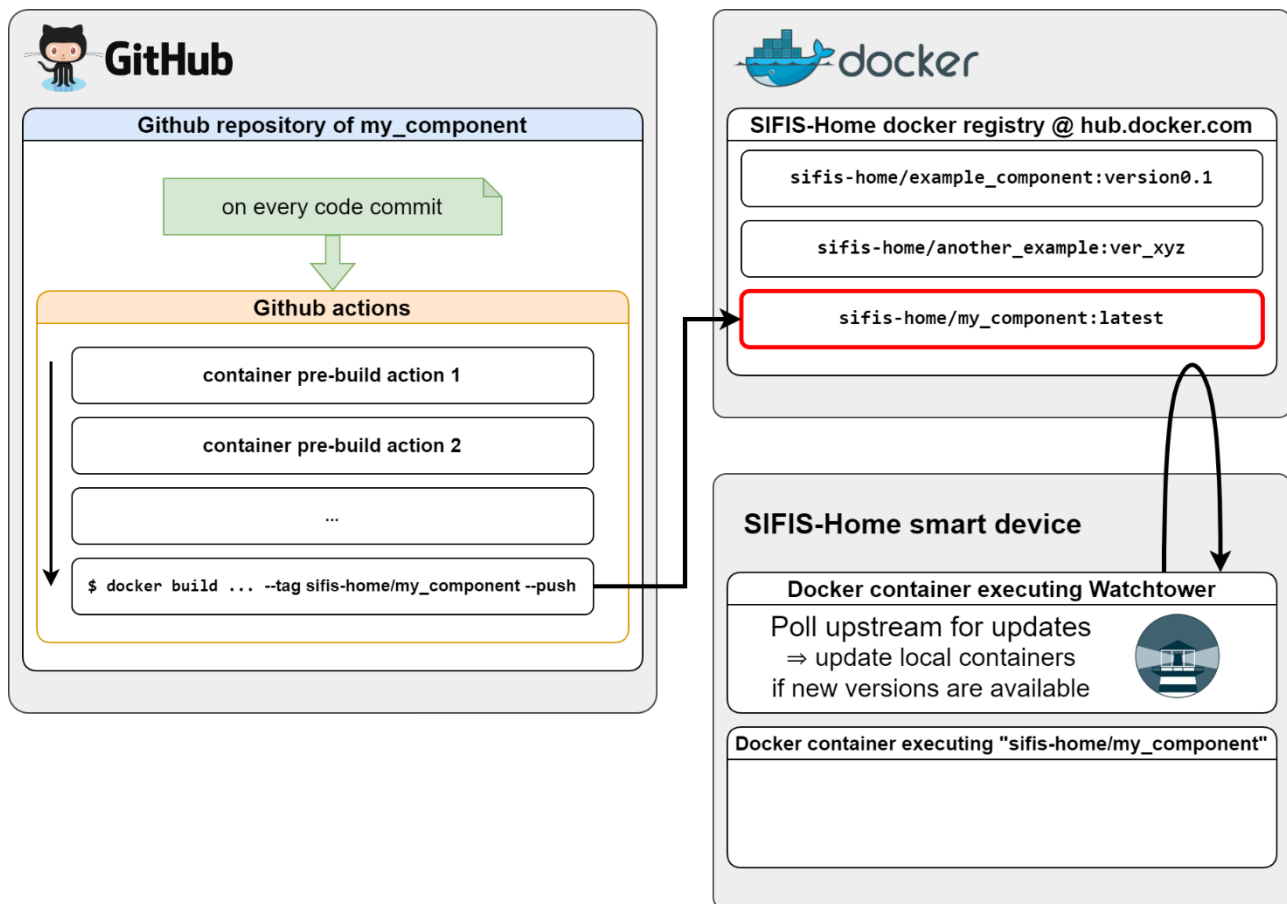


Figure 22 The SIFIS-Home CI/CD process

Example CI/CD scenario:

1. Developer commits new code to GitHub.
2. GitHub actions triggers on every commit.
3. The final GitHub action step is the build command that produces a docker image. The new docker image is pushed automatically to the docker registry hosted in Docker Hub.
4. For every SIFIS-Home Smart Device, once an updated version of a Docker image is detected by Watchtower, Watchtower will upgrade any local Docker image running older versions.

Since SIFIS-Home is a research project, our decision was to just use one step in the deployment phase of new software. On the other hand, in a commercial setup, there would be at least two more, and probably three or even more, steps in the process to deploy an upgraded software version. An example for a commercial deployment can be:

1. Development branch: Test devices to check latest integrated code.
2. System test: System verification and validation once development branch stabilized.
3. Commercial deployment stage 1: First commercial deployment of limited devices.
4. Commercial deployment stage 2: Second phase of commercial deployment

5. Commercial deployment final stage: Deploy to all devices.

At any stage, if deployment does not work out well, the process can be stopped and devices roll-backed to a previous software version.

7 Validation of implementation and of the testbed

This section reports the set of use cases of the SIFIS-Home framework that we used in order to design and verify the testbed. In the previous section, it was stated that it should be possible to verify and validate the NFRs (Non-Functional Requirements) as defined in deliverables D1.2 by using the testbed.

Independent of the programming language used to develop the different components, their integration and execution should be feasible, as well as the continuous integration and continuous deployment of related new code uploaded to the SIFIS-Home GitHub. Once the system is up and running, it is required that the communication flow between all devices works reliably (e.g., from Smart Device to Smart Device, from Smart Device to API gateway, from API gateway to all devices, etc.). Also, access to services and resources must be secured and verified in the light of access policies to enforce.

Req. ID	Req. description	FR	Priority	Validation strategy
PE-01	The user authentication shall happen in less than 2s.	F-02	Critical	UI performance test
		F-03		
PE-02	The user recognition (identification/ biometric-based) shall happen in less than 5s.	F-06	Critical	UI performance test
PE-03	Biometric-based authentication should be performed in less than 5 seconds.	F-03	Standard	Under consideration
PE-04	Activation of features based on user identity (biometric recognition) should be performed in less than 5 seconds.	F-04	Standard	UI performance test
		F-05		
PE-05	Recognition of the start of an interaction through voice command should be performed in less than 2 seconds.	F-06	Standard	Under consideration
PE-06	The interpretation of the voice commands provided by the user should be performed in less than 2 seconds.	F-07	Standard	Under consideration
PE-07	A command should be invoked within 5 seconds from the event that triggered its execution	F-08	Standard	UI performance test
PE-08	The maintainer must be able to access and watch a recording in less than one minute.	F-13	Standard	Under consideration
PE-09	If requested to, the SIFIS-Home system shall contact law enforcement or private surveillance services to receive assistance in less than 30 seconds.	F-14	Optional	Event performance test. Use test mobile as receiver.
PE-10	An abnormal (suspicious) behavior caused by malware shall be identified and notified within 60 seconds	F-19	Optional	Event performance test.
PE-11	The user should be informed of the presence of malware no later than 5 seconds after the malware is recognized.	F-20	Standard	Event performance test.
PE-12	Self-healing algorithms should be started in less than 60 seconds if available when malware is recognized.	F-21	Critical	Event performance test.
PE-13	The registration of a new device should be completed in less than 30 seconds.	F-23	Standard	Event performance test.
PE-14	The list of registered devices shall be shown by the SIFIS-Home system in less than 30 seconds.	F-24	Standard	UI performance test

PE-15	The de-registration of a device should be completed in less than 30 seconds.	F-25	Standard	UI performance test
PE-16	The correct configuration changes should be propagated successfully in less than 30 seconds.	F-26	Critical	
				UI performance test
PE-17	The current configuration of a device should be retrieved in less than 10 seconds.	F-26	Standard	UI performance test
PE-18	The marketplace should be accessible in less than 60 seconds.	F-28	Standard	UI performance test
PE-19	The configuration of policies for groups of users should be applied and enforced in less than 60 seconds.	F-32	Critical	UI performance test
PE-20	The configuration of policies for groups of devices should be applied and enforced in less than 60 seconds.	F-33	Critical	Event performance test
PE-21	The list of policies should be retrieved in less than 30 seconds.	F-30	Standard	Event performance test
PE-22	The configuration of profiles should be applied and enforced in less than 60 seconds.	F-37	Critical	
				Event performance test
PE-23	The change of current profile should be performed in less than 60 seconds.	F-38	Critical	Event performance test
PE-24	The statistics about usage and behavior of devices should be presented to the administrator in less than 30 seconds.	F-41	Standard	UI performance test
PE-25	The statistics about usage of profiles should be presented to the administrator in less than 30 seconds.	F-42	Standard	UI performance test
PE-26	Remote log-in should be performed in less than 60 seconds.	F-43	Critical	Event performance test
PE-27	In case of an incomplete or unsuccessful command execution, an error response should be sent within 5 seconds	F-08	Standard	Event performance test
PE-28	The used solutions for communication and system security shall be as much as possible lightweight to enforce in terms of performance and especially feasible also for resource-constrained devices.	All	Critical	By design
PE-29	The performance impact due to communication and system security shall not result in unacceptable impact on the user experience.	All	Critical	UI performance test
PE-30	The network infrastructure shall provide means also for one-to-many message delivery, e.g., over IP multicast.	F-47	Critical	By design
		F-48		
		F-49		
		F-50		
PE-31	It must be possible to have multiple security groups simultaneously active in the system.	F-47	Critical	By design
		F-48		
		F-49		
		F-50		

PE-32	When relevant, support shall be ensured for communication intermediaries performing, e.g., message forwarding and/or (transport-) protocol translation. This applies also in secure scenarios and in (secure) group communication scenarios.	All	Critical	By design
PE-33	When relevant, it shall be possible to enable one-to-many response messages, sent at once to multiple requesters. This applies also to secure communication scenarios and in presence of communication intermediaries.	All	Critical	By design
PE-34	When relevant and limited to read-only operations, it shall be possible to enable cache ability of response messages at communication intermediaries, also when protected end-to-end.	All	Critical	By design
PE-35	Devices should, if available, utilize low-power modes of operation to further mitigate the performance impact of ongoing (D)DoS attacks.	All	Standard	By design
PE-36	There should be a means to enable an optimized, combined establishment of a cryptographic secret with a first message protected with key material derived from that secret.	All	Standard	By design
PE-37	In case of an incomplete or unsuccessful configuration change, an error message should be returned within 5 seconds	F-26	Standard	UI performance test
RE-01	The system shall not fail more than once a week (on average).	All	Critical	Event performance test
RE-02	The system shall not take more than one day to be repaired (on average).	All	Critical	Event performance test
AV-01	The SIFIS-Home system services and devices shall be available 99% of the time	All	Critical	Event performance test
AV-02	The SIFIS-Home system shall ensure basic services availability in case of system failures.	All	Critical	Event performance test
AV-03	Support should be ensured for devices to dynamically react to (D)DoS attacks, by gradually adapting their availability. This includes relying on communication intermediaries for traffic offloading during intense (D)DoS attacks.	All	Standard	By design
AV-04	Devices under (D)DoS attacks should be able to continue providing a (best-effort) service to legitimate requests, i.e., by displaying a graceful degradation of quality of service.	All	Standard	Event performance test
US-01	The system shall be easy to use for users with no technical background	All	Critical	UI performance test
US-02	The SIFIS-Home system shall be autonomous and learn based on the users' habits, still according to defined privacy policies.	All	Critical	By design
US-03	The SIFIS-Home system shall consider special cases in its design, such as color blindness.	All	Optional	By design
US-04	The SIFIS-Home system shall preserve consistency among all devices, related databases, and constraints.	All	Critical	By design

US-05	The SIFIS-Home hardware components should be easy to use for the elderly and users with no engineering background.	All	Optional	Not Verifiable
US-06	The SIFIS-Home system shall have an explorable interface.	All	Standard	By design
US-07	Proper and easy hardware installation should be considered.	All	Standard	By design
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.	F-01	Standard	Event performance test
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.	F-09	Critical	
		F-13		UI performance test
US-10	An untrained user should be able to recognize a software intrusion in less than one minute.	F-19	Critical	UI performance test
		F-20		
US-11	An untrained user should be able to perform the device registration procedure in less than 5 minutes.	F-23	Standard	UI performance test
US-12	An untrained user should be able to perform the device de-registration procedure in less than 5 minutes.	F-26	Standard	UI performance test
US-13	An untrained user should be able to perform the configuration of devices in less than 5 minutes.	F-26	Standard	UI performance test
US-14	An untrained user should be able to perform the installation of an application in less than 5 minutes.	F-28	Standard	UI performance test
US-15	An untrained user should be able to complete the configuration of policies for groups of users in less than 5 minutes.	F-32	Standard	UI performance test
US-16	An untrained user should be able to complete the configuration of policies for groups of devices in less than 5 minutes.	F-33	Standard	UI performance test
US-17	An untrained user should be able to complete the configuration of profiles in less than 5 minutes.	F-37	Standard	UI performance test
US-18	An untrained user should be able to perform a profile change in less than 30 seconds.	F-38	Standard	UI performance test
US-19	An untrained user should be able to access the statistics for visualizing and interpreting them in less than 5 minutes.	F-41	Standard	UI performance test
US-20	The Multi-Level Anomaly Detection system (MLADS) must monitor network traffic provided by several input sources and several locations.	F-15	C	By design
		F-16		
		F-17		
		F-18		
US-21	The workload of the devices should be available to the MLADS.	F-15	C	By design
		F-16		
		F-17		
		F-18		
US-22	The list of applications running on each device should be available to MLADS.	F-15	C	By design
		F-16		
		F-17		

		F-18		
US-23	Raw sensor data must be available to be analyzed by MLADS.	F-15	C	By design
		F-16		
		F-17		
		F-18		
US-24	Features from different devices should be aggregable directly or by means of pre-processing through specific analysis tools.	F-15	C	By design
		F-16		
		F-17		
		F-18		
US-25	When possible, a dataset should not be present as a whole on a single device for analysis.	All	S	By design
US-26	The presence of a GPU is needed to perform DL-based analysis.	F-15	S	By design
		F-16		
		F-17		
		F-18		
DE-01	Identification through biometrics should be performed correctly in more than 95% of cases.	F-01	Critical	Test on public datasets
DE-02	The start of an interaction command should be recognized properly and correctly in more than 99% of cases.	F-06	Critical	Event performance test
DE-03	The commands to execute should be recognized properly and correctly in more than 95% of cases.	F-06	Critical	Event performance test
		F-07		
DE-04	Record of intrusions must be available for a configurable time (default six months) after the recording.	F-10	Standard	By design
DE-05	Identity of the successfully recognized intruders must be available for a configurable time (default six months) after the recording.	F-12	Standard	By design
DE-06	Core functionalities should be replicated on multiple devices to avoid single points of failure.	F-21	Critical	By design
DE-07	The registration of a new device should be successful in at least 99% of the cases.	F-23	Critical	Event performance test
DE-08	The de-registration of a new device should be successful in at least 99% of the cases	F-25	Critical	Event performance test
DE-09	The configuration changes should be propagated successfully to the devices more than 99% of the time.	F-26	Critical	Event performance test
DE-10	The SIFIS-Home system should be able to restore the previous configurations if there are errors in applying configuration changes.	F-26	Standard	Event performance test
DE-11	The installation of the selected app should be completed successfully in at least 95% of cases.	F-28	Critical	Event performance test
DE-12	The application of policies should always be completed successfully.	F-31	Critical	Event performance test
		F-34		
		F-33		
DE-13	The configuration of profiles should be completed successfully in at least 99% of cases.	F-37	Critical	Event performance test
DE-14	The change of current profile should be completed successfully in at least 99% of cases.	F-38	Critical	Event performance test
DE-15	The statistics must be shown correctly in at least 99% of cases.	F-41	Critical	Event performance test

DE-16	Remote log-in for the configure should be successful in at least 99% cases.	F-43	Critical	Event performance test
DE-17	The SIFIS-Home system should be able to distribute the processing among multiple machines in separate places if required.	All	Critical	Event performance test
DE-18	The SIFIS-Home system is required to be fault tolerant, it should continue to operate, even if one or more of the nodes fail.	All	Critical	Event performance test
DE-19	The SIFIS-Home system is required to be scalable dynamically by adding or removing nodes according to demand.	All	Critical	Event performance test
TE-01	The SIFIS-Home system needs Java version 8 or higher to interact with the ontology.		Critical	By Design
TE-02	The process for getting and inserting information into the ontology will be through APIs provided via HTTP(S).		Critical	By Design
TE-03	The software for handling the ontology should be hosted on a high-availability server.		Critical	By Design
TE-04	Internet connectivity should be present.		Standard	By Design

Table 4 Finalized list of Non-Functional requirements for the SIFIS-Home framework.

8 Conclusion

In this deliverable, we have presented how we analysed the key SIFIS-Home requirements and the consortium partners' assets and expertise to design and manage the development, integration and realization of the two WP5 SIFIS-Home testbeds, which are being built around the Panarea server provided by CNR.

- Emulated testbed: Smart devices being built upon virtual servers with x86 hardware architecture inside the Panarea server. These smart devices will be used to verify and validate the full SIFIS-Home framework except the NSSD devices.
- Simulated testbed: Smart devices being built upon Raspberry PIs with ARM hardware architecture. These smart devices will be used to verify and validate physical interaction with NSSD devices connected to the smart devices. Since not all SIFIS-Home components are able to execute on ARM architecture, a subset of the SIFIS-Home components will execute in the Raspberry PIs.

Both the emulated and the simulated testbed will be connected to the cloud interface for user interaction and visualization of the state of the testbeds. To make the development and validation process efficient, a CI (Continuous integration) and CD (Continuous deployment) process was defined built upon the state-of-the-art development tool Docker, GitHub and Docker Hub.

During the development and integration of the software components and the testbeds, it was not feasible to integrate the original architecture from deliverable D1.3. Therefore, it was further advanced in D1.4 / D5.2. Implementation of the components and the testbeds - as described in this deliverable D5.3 - is currently being undertaken based upon the further advanced architecture and our conclusion is that it is now feasible to implement the complete SIFIS-Home system. The main integration challenges relate to the Cyber-Security perimeter and the fact that users must be able to access the SIFIS-Home network from inside the home and from outside the home. However, this issue has now a feasible and secure solution, as described in this document. The remaining work now is to complete the development and integration of the components and to get the verification and validation of the SIFIS-Home components and modules on top of the started testbeds.

As visible from the UX snapshots, the final versions of the testbeds with the SIFIS-Home software are up and running on the Panarea server, even if some development work still remains. The partners continuously use the established CI/CD integration tool chain to provide code both to the SIFIS-Home GitHub as well as to deploy new components to the testbeds.

9 References

[GITHUB] GitHub is where over 100 million developers shape the future of software, together.
<https://github.com>

[DOCKER] Develop faster. Run anywhere, <https://www.docker.com/>

[DOCKER HUB] Build and Ship any Application Anywhere, <https://hub.docker.com/>

[WATCHTOWER] A process for automating Docker container base image updates.
<https://github.com/containrrr/watchtower>

[TENSORFLOW] TensorFlow, An end-to-end machine learning platform,
<https://www.tensorflow.org/>

[WoT, 2020] Web Of Things (WoT) Architecture, W3C recommendation 9 April 2020,
<https://www.w3.org/TR/wot-architecture/>

[FIWARE, 2021] What is FIWARE?, <https://www.fiware.org/developers/>

[RATATOSK] A secure and multi-tenant implementation of the FIWARE Orion Publish/Subscribe Context Broker [Ratatosk context broker - Sensative](#)

[YGGIO, 2021] Yggio DiMS, Digitalization infrastructure Management System,
<https://sensative.com/yggio/>

Glossary

Acronym	Definition
AODV	Ad-hoc On-demand Distance Vector
CH	Context Handler
DHT	Distributed Hash Table
FR	Functional Requirements
NFR	Non-functional requirement
NSSD	Not So Smart Device
OS	Operative System
P2P	Peer to Peer
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PTP	Policy Translation Point
SD	Smart Device
SIFIS-Home	Secure Interoperable Full Stack Internet of Things for Smart Home
UC	Use case
US	User story
XACML	eXtensible Access Control Markup Language