assistants
Testing per smart home sicure, private e sostenibili

D3.2

Pilot Implementation

WP3 – Pilot

ASSISTANTS

Assessing Software Security, prIvacy, and SusTAiNability through Testing techniqueS

Due date of deliverable: 30/09/2025 (M22)
Actual submission date: 30/09/2025

Responsible partner: Politecnico di Torino (POLITO)
Editor: Luca Ardito
E-mail address: luca.ardito@polito.it

28/02/2026
Version 1.1

Project funded by the European Union – NextGenerationEU within the PRIN 2022 PNRR programme		
Dissemination Level		
PU	Public	X
CO	Confidential, only for members of the consortium	



The ASSISTANTS project (P2022WEAH7, CUP E53D23016400001) is funded by the European Union – NextGenerationEU under the PRIN 2022 PNRR programme of the Italian Ministry of University and Research (D.D. 1409 of 14/09/2022), Mission 4, Component 2, Investment 1.1.

Authors: Luca Ardito (POLITO)

Approved by: Luca Ardito (POLITO), Principal Investigator

Revision History

Version	Date	Name	Partner	Sections Affected / Comments
1.0	30/09/2025	Luca Ardito	POLITO	All. Final version
1.1	28/02/2026	Luca Ardito	POLITO	All. Revision for publication on the project website

Disclaimer

Funded by the European Union – NextGenerationEU. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Executive Summary

This deliverable accompanies the software of the ASSISTANTS pilot, produced in Task T3.2, *Use Case Design and Implementation*. The pilot integrates the results of the project into Smartotum, a real home-automation system whose gateway hosts local services and whose Distributed Hash Table (DHT) is the meeting point for home configuration, devices, capabilities, privacy rules, telemetry, commands and state.

The implementation consists of four software components, all released as open source under the MIT licence:

- **tosca**, the Rust framework of the project, which offers one set of APIs to build device firmware and another to build controllers that interact with those devices;
- **demo-light**, the firmware of a smart light for the ESP32-C3 microcontroller, with manual, motion-sensor and light-sensor modes, a temperature sensor and an OLED display;
- **smartotum-controller**, the bridge running on the Smartotum gateway, which discovers ASSISTANTS devices on the local network, collects their events and keeps their state aligned in the DHT;
- **privacy-manager**, the service running on the gateway that enforces the privacy rules: it marks restricted devices in the DHT, so that the Smartotum interface disables their actions, and for cameras it also blocks all network traffic to the device on the gateway.

The Privacy Dashboard (D2.3) is integrated with the same DHT: it publishes the rules defined by its users and the automatic deny rules derived from withdrawn consents, and it synchronises homes, rooms, devices and applications from Smartotum. The controller and the privacy manager run as Docker containers that start automatically with the gateway. The document describes each component, the integration flows, the deployment on the gateway and the repositories where the code is available.

Table of contents

Executive Summary 3
1 Introduction 5
1.1 Purpose and scope 5
1.2 Relation with other deliverables 5
1.3 Structure of the document 5
2 Overview of the Pilot Implementation 6
3 The tosca Framework 8
3.1 Framework structure 8
3.2 Passive controller model 8
4 Smart Light Demonstrator 9
4.1 Hardware 9
4.2 Firmware 10
5 Smartotum Controller 12
5.1 Device topic lifecycle 12
6 Privacy Manager 15
6.1 Rule model 15
6.2 Behaviour 15
6.3 Enforcement on devices 15
6.4 Camera-specific enforcement 16
7 Integration with the Privacy Dashboard 18
8 Deployment on the Smartotum Gateway 20
9 Repositories and Licence 21
10 Conclusions 22
References 23
Glossary 24

1 Introduction

1.1 Purpose and scope

This deliverable is the output of Task T3.2 of WP3. It is of type *Software*: the main result is the source code of the pilot, published in public repositories. The document describes the architecture of the implementation, the role and behaviour of each component, how the components are integrated in Smartotum and how they are built and deployed.

1.2 Relation with other deliverables

The implementation realises the use cases and requirements defined in D3.1 (identifiers UC, FR, NFR and SPR in this document refer to D3.1). The Privacy Dashboard described in D2.3 is the entry point for the definition of rules and consents, and the policy concepts come from D1.2 and D2.2. The evaluation of the pilot is reported in D3.3.

1.3 Structure of the document

Section 2 gives an overview of the pilot implementation. Section 3 describes the tosca framework. Sections 4, 5 and 6 describe the smart light, the Smartotum controller and the privacy manager. Section 7 describes the integration with the Privacy Dashboard. Section 8 describes the deployment on the Smartotum gateway, and Section 9 lists the repositories and their licence. Section 10 concludes the document.

2 Overview of the Pilot Implementation

The pilot was built around the Smartotum DHT, which the Smartotum services use following a repository pattern: components do not talk to each other directly, but read and write shared information in the DHT, which is the single reference for the state of the devices [1]. This decouples user interface, controllers and physical devices, and it allowed the ASSISTANTS components to be added without changing the existing architecture (NFR-08).

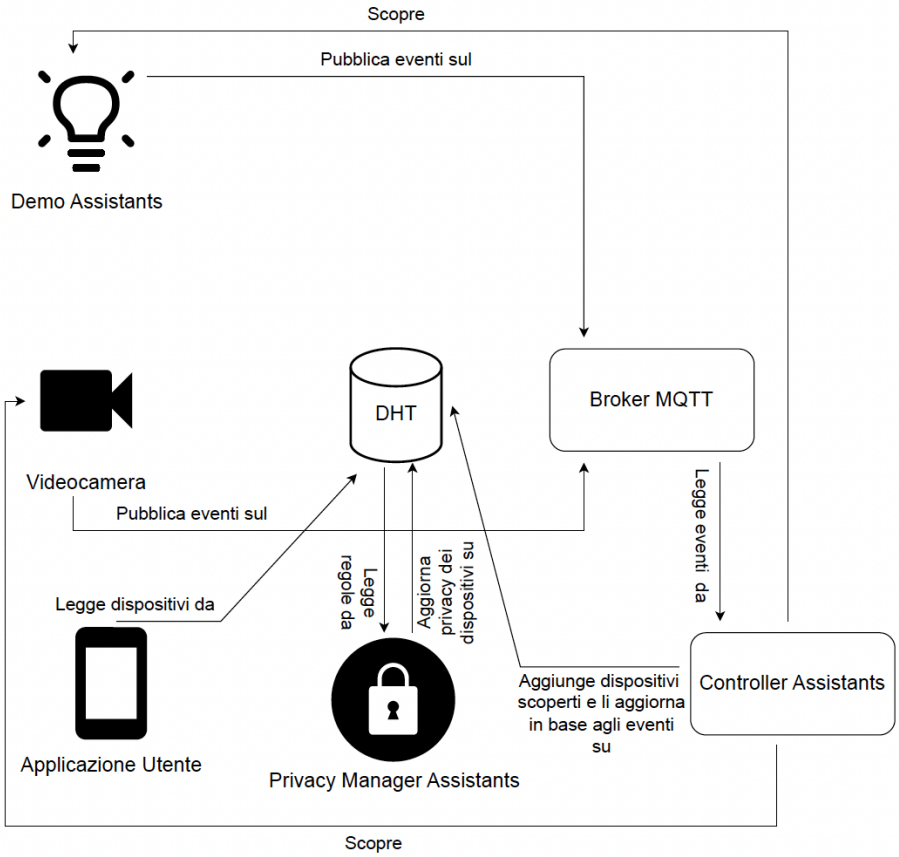


Figure 1: Integration of the ASSISTANTS components in the Smartotum ecosystem (source: [1])

Figure 1 shows the resulting integration:

- the smart light publishes its events to the MQTT broker of the gateway;
- the Smartotum controller receives the events and writes the state of the light into the DHT, where the Smartotum interface reads it;
- privacy rules are stored in the DHT as `privacy_rule` topics, created from the Privacy Dashboard or from the Smartotum interface;
- the privacy manager reads the rules and enforces them on the target devices, including the camera;
- the user application shows the devices and their state, and disables the actions on restricted devices.

Table 1: Software components of the pilot

Component	Language	Runs on	Role
tosca	Rust	Device and gateway	Framework: firmware APIs and controller APIs
demo-light	Rust	ESP32-C3	Smart light firmware

smartotum-controller	Rust	Smartotum gateway (Docker)	Device discovery, events, DHT state
privacy-manager	Python	Smartotum gateway (Docker)	Enforcement of privacy rules
Privacy Dashboard	Python, TypeScript	Server (Docker)	Rules, consents, GDPR roles, synchronisation with Smartotum

3 The tosca Framework

tosca is the IoT framework developed in the project [2]. It is written in Rust, chosen for its performance and for an ownership model and type system that guarantee memory and thread safety at compile time and prevent whole classes of vulnerabilities (NFR-07). The framework offers two distinct sets of APIs: the first is used to build firmware for devices running either a general-purpose operating system or bare-metal embedded systems; the second is used to build *controllers*, software components that interact with those devices.

3.1 Framework structure

The framework is a Cargo workspace organised around a common interface crate:

Table 2: Crates of the tosca framework

Crate	Side	Content
`tosca`	Interface	REST routes, route parameters and responses, device description, hazards; `no_std` for embedded targets
`tosca-esp32c3`	Firmware	Library to build firmware for ESP32-C3 boards
`tosca-os`	Firmware	Library to build firmware running on operating systems
`tosca-drivers`	Firmware	Architecture-agnostic sensor drivers built on `embedded-hal` and `embedded-hal-async`
`tosca-controller`	Controller	APIs to discover, orchestrate and interact with devices built with the framework

The `tosca` crate is the interface between a device and a controller. It can:

- create and manage REST routes through which a controller sends commands to a device, with parameters mirroring those used by the device; responses can be a simple `Ok`, a `Serial` response with data about the operation, an `Info` response with metadata about the device, or an optional `Stream` response delivering multimedia data;
- describe a device, including the structure of its firmware, its data and methods, and information about its resource consumption at economic and energy level;
- associate *hazards* to routes, to describe the risks of a device operation in three categories: Safety (risks to human life), Financial (economic impact) and Privacy (data management).

The APIs are designed as composable blocks: a firmware that supports events adds the event APIs to its server, while a firmware that does not use them leaves them out. Optional features reduce binary size and compilation time.

3.2 Passive controller model

The framework adopts a passive model: devices are discovered automatically on the network by the controller, which asks each device for its structural description. Based on this description, the controller sends commands consistent with the capabilities that the device exposes. The same model is used by the Smartotum controller of the pilot (Section 5).

4 Smart Light Demonstrator

4.1 Hardware

The demonstrator is a smart light built on an ESP32-C3 microcontroller with the firmware APIs of the framework [3]. Its components are listed in Table 3 and the wiring is shown in Figure 2.

Table 3: Hardware components of the smart light

Component	Function
ESP32-C3 microcontroller	Central control unit; runs the ASSISTANTS firmware, manages sensors and actuators and communicates with the MQTT broker of the gateway
Switch	Physical on/off input in manual mode
AM312 PIR motion sensor	Detects movement and switches the light on in motion mode; the light is switched off when no more movement is detected
BH1750 light sensor	Measures ambient light and switches the light on below a threshold in light-sensor mode
5 V relay	Main actuator; opens or closes the 220 V lamp circuit in all modes
220 V lamp	Electrical load controlled by the system
DS18B20 temperature sensor	Measures the ambient temperature, made available to the Smartotum interface
SSD1306 OLED display	Shows the temperature measured by the DS18B20 sensor

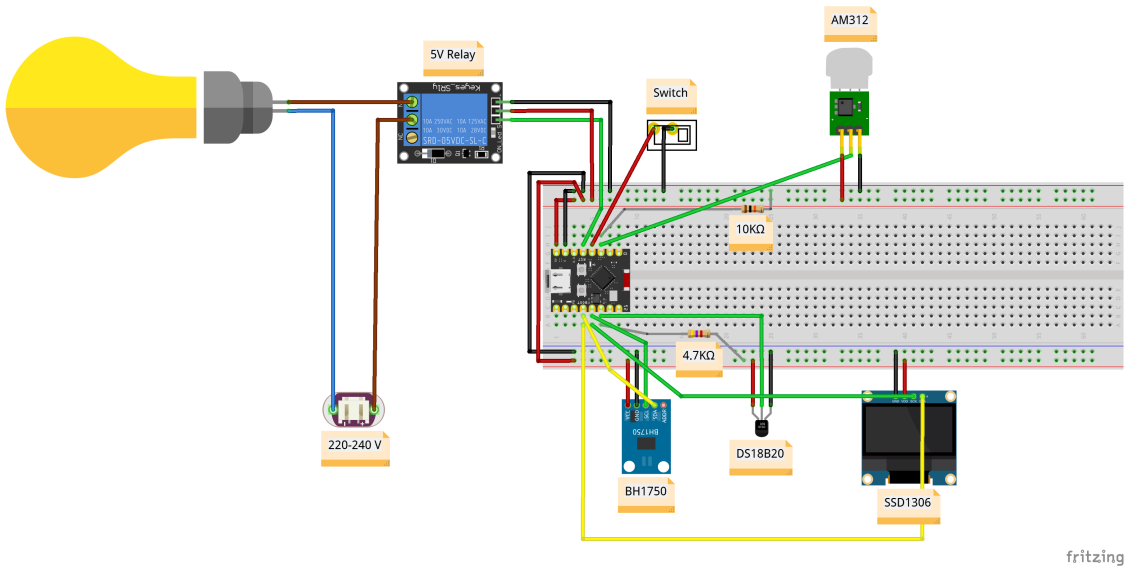


Figure 2: Wiring of the smart light demonstrator (source: [3])

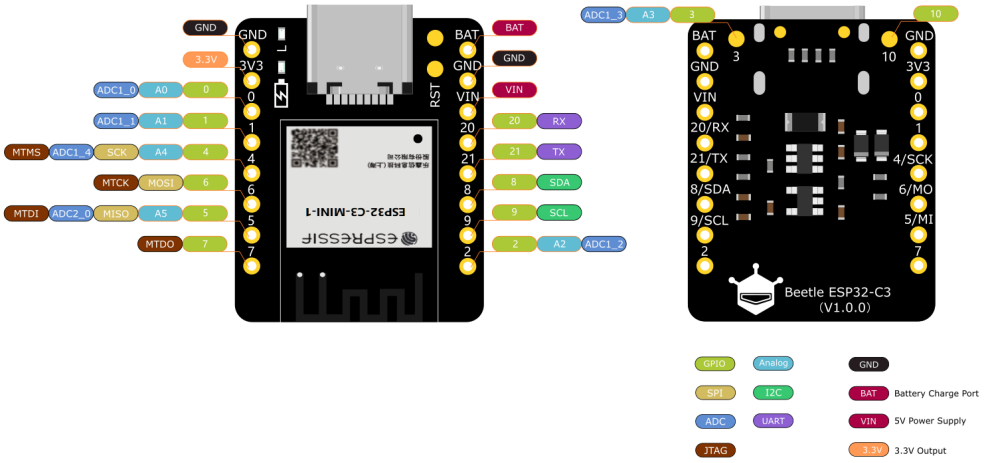


Figure 3: ESP32-C3 board used in the demonstrator (source: [1])

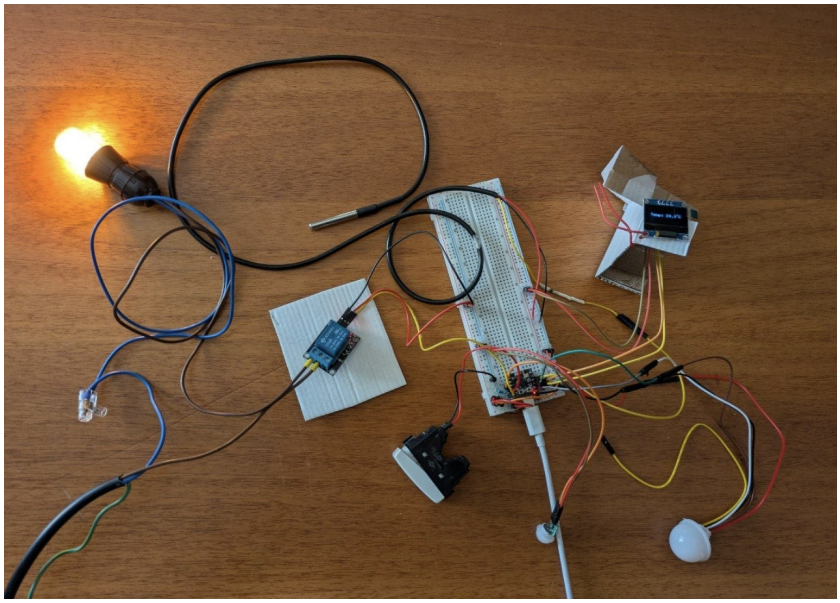


Figure 4: Components of the demonstrator (source: [1])

4.2 Firmware

The firmware runs an HTTP server that manages the state of the light through REST requests (FR-13):

Table 4: REST routes of the smart light

Route	Effect
`light/on`	Switches the light on
`light/off`	Switches the light off
`light/toggle`	Toggles the light
`light/manual`	Enables manual mode: control through the physical switch or the on, off and toggle routes
`light/motion-detection`	Enables motion mode, driven by the AM312 sensor
`light/ambient-light`	Enables light-sensor mode, driven by the BH1750 sensor

Every change of state generates a JSON message describing the current state of the device, published on the topic `/light/<mac_address>/events` of the MQTT broker (FR-14). The board can be

discovered by other nodes on the same network through an mDNS-SD service. The firmware is built on the Embassy asynchronous runtime.

The repository provides two binaries: `demo-light`, for the board with the real sensors and relay, and `mock`, which simulates the sensors with the built-in LED and the button of the board, for development and testing without hardware. Both are built and flashed with Cargo using the release profile, after copying `cfg.toml.example` to `cfg.toml` and setting the Wi-Fi credentials and the address and port of the MQTT broker.

In all three modes the light can also be switched on and off from the Smartotum interface. Figure 5 shows the card of the smart light in the Smartotum interface, with the three operating modes and the temperature.

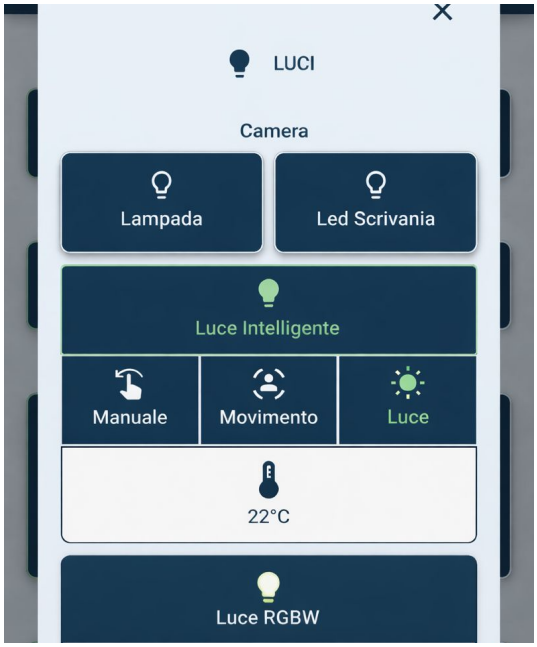


Figure 5: The smart light in the Smartotum interface (source: [1])

5 Smartotum Controller

The Smartotum controller is the bridge between ASSISTANTS devices and the Smartotum ecosystem [4]. It is built with the controller APIs of tosca, runs on the Smartotum gateway and operates entirely on the local network, without external services (NFR-05). It follows the passive controller model:

- it periodically discovers the devices on the local network (every 30 seconds) and requests their description (FR-15);
- for each new device it creates a topic in the DHT, of type `smart_light_assistants`, assigned to a room of the home; for known devices it reuses the existing topic;
- it receives the events generated by the devices and updates their topic with the current state: name, network address, room, on/off status, operating mode and temperature;
- it preserves the privacy fields written by the privacy manager when it updates the state of a device;
- it retries DHT operations that fail temporarily, so that the state converges when the DHT becomes available again.

5.1 Device topic lifecycle

When the light is detected and registered, a new topic is created in the DHT with the information about the device, including its network identifiers, operating mode and current state, initially off (Figure 6). The topic is immediately visible to all Smartotum services, including the interface.


```
{
  "topic_name": "smart_light_assistants",
  "topic_uuid": "caba8a46-9486-4333-8d80-36cf6ddee387",
  "value": {
    "area_name": "bf673d0c-13ff-4e7c-9169-9a2cf2d28bb4",
    "ip_address": "10.0.1.136",
    "mac_address": "4a:31:b7:3e:46:6c",
    "mode": "manual",
    "name": "Luce Intelligente",
    "status": false,
    "temperature": 22
  }
}
```


Figure 6: Topic of the smart light in the DHT (source: [1])

When the light is not powered, its topic is absent or not yet populated, and the interface shows the device as unavailable (Figure 7). Once the light is powered, the controller updates the topic and the interface shows the light as available but off; the temperature sensor is already visible (Figure 8).

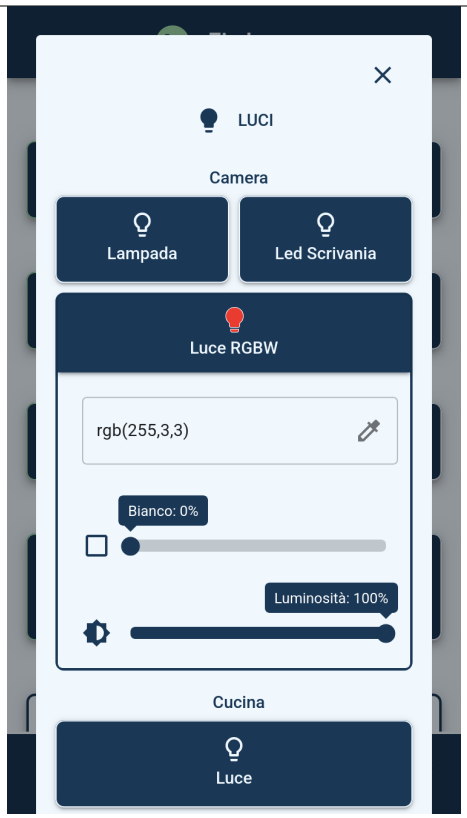


Figure 7: The Smartotum interface without the smart light (source: [1])

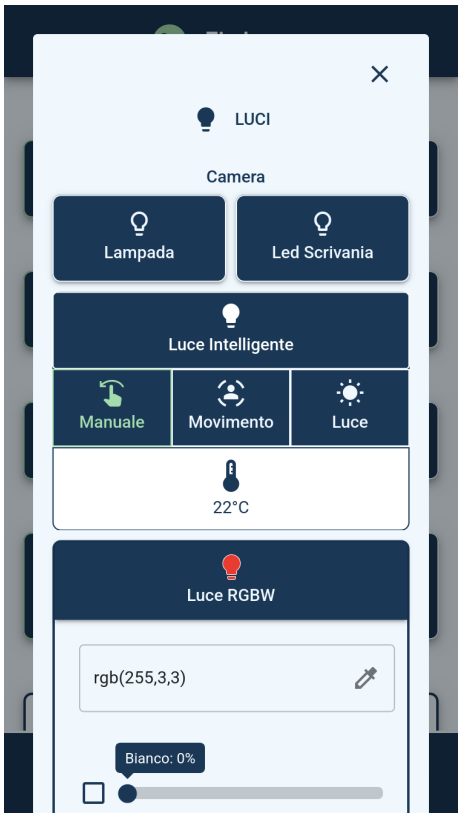


Figure 8: The Smartotum interface with the smart light connected and off (source: [1])

When the user switches the light on from the interface, the command is handled by the controller, which updates the topic: the status changes from `false` to `true` (Figure 9). The update is propagated to all the components subscribed to the DHT, and the interface shows the light as on (Figure 10).


```
{
  "topic_name": "smart_light_assistants",
  "topic_uuid": "caba8a46-9486-4333-8d80-36cf6ddee387",
  "value": {
    "area_name": "bf673d0c-13ff-4e7c-9169-9a2cf2d28bb4",
    "ip_address": "10.0.1.136",
    "mac_address": "4a:31:b7:3e:46:6c",
    "mode": "manual",
    "name": "Luce Intelligente",
    "status": true,
    "temperature": 22
  }
}
```


Figure 9: Topic of the smart light after the switch-on command (source: [1])

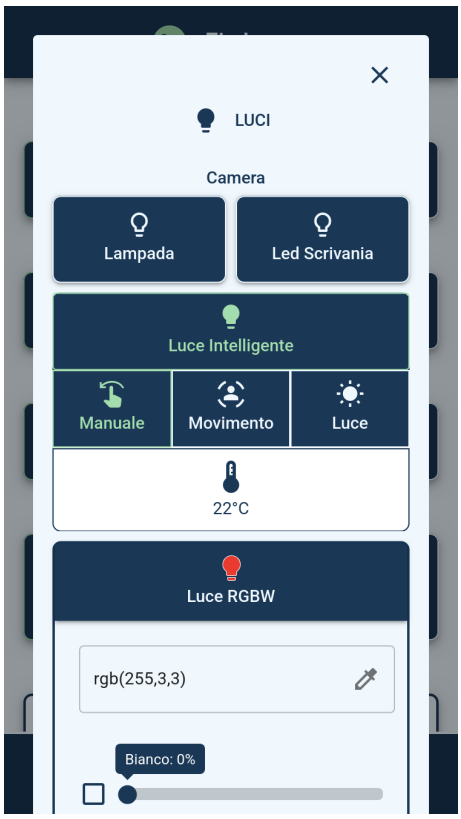


Figure 10: The Smartotum interface with the smart light on (source: [1])

6 Privacy Manager

The privacy manager enforces the privacy rules defined by the inhabitants of a Smartotum home [5]. It is written in Python with `asyncio` and runs as a periodic background service on the gateway, keeping the state of the devices consistent with the rules stored in the DHT.

6.1 Rule model

A privacy rule is a DHT topic with `topic_name` equal to `privacy_rule`. It identifies its target through `target_topic` and `target_uuid` and defines a daily time window, the days of the week in which it applies and a final expiration date (FR-01, FR-02):

```
{
  "topic_name": "privacy_rule",
  "topic_uuid": "a79882ed-5aa0-451b-bd5b-c5147e801d0c",
  "value": {
    "target_topic": "domo_light",
    "target_uuid": "caba8a46-9486-4333-8d80-36cf6ddee387",
    "time_start": "21:00",
    "time_end": "22:00",
    "days": ["Monday", "Wednesday"],
    "expiration_date": "2026/08/01"
  }
}
```

The absence of a rule for a device means that its actions are allowed; a valid rule denies all the actions on that device in the period it covers (FR-03). The rule structure is the same for every supported device; only the target changes.

6.2 Behaviour

Every minute the privacy manager reads the rules from the DHT and:

- detects newly added rules;
- decides whether each rule is active, from the current day, the time window and the expiration date;
- applies the restriction to the target device when a rule becomes active (FR-04);
- removes the restriction when the rule is no longer active (FR-07);
- deletes expired rules from the DHT (FR-08).

In addition, it listens to the DHT for rule deletions: when a user removes a rule, the devices restricted by that rule are released immediately (FR-09). At start-up it resets any restriction left from a previous run, so that the state of the devices always reflects the current rules.

6.3 Enforcement on devices

A restriction is applied by updating the topic of the target device in the DHT with two fields: `privacy`, which signals that a temporary restriction is active, and `privacy_until`, which gives the time at which it ends. The privacy rule therefore becomes part of the state of the device, and every component of the system can detect it. The Smartotum interface reads these fields and disables the actions on the device, telling the user that they are temporarily blocked (FR-05).

```
{
  "topic_name": "smart_light_assistants",
  "topic_uuid": "caba8a46-9486-4333-8d80-36cf6ddee387",
  "value": {
    "area_name": "bf673d0c-13ff-4e7c-9169-9a2cf2d28bb4",
    "ip_address": "10.0.1.136",
    "mac_address": "4a:31:b7:3e:46:6c",
    "mode": "manual",
    "privacy": true,
    "privacy_until": "22:00",
    "name": "Luce Intelligente",
    "status": true,
    "temperature": 22
  }
}
```


Figure 11: Topic of the smart light with an active privacy rule (source: [1])

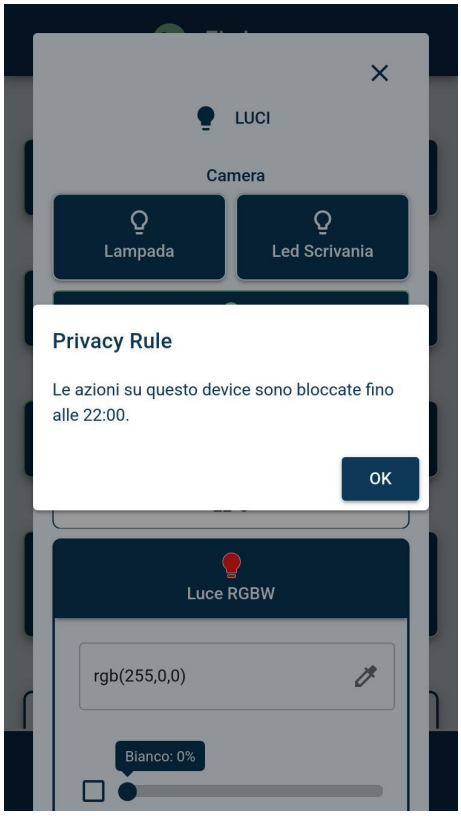


Figure 12: Actions on the smart light blocked by a privacy rule (source: [1])

6.4 Camera-specific enforcement

For cameras (`domo_camera` topics), in addition to updating the DHT topic, the privacy manager enforces a block at network level (FR-06, SPR-04). It maintains a dedicated `iptables` chain on the gateway, linked to the forwarding and output paths, and adds a rule that drops all traffic directed to the address of the camera while the privacy rule is active. Direct access to the camera is thus prevented during the privacy period, including attempts made outside the Smartotum application. The firewall rule is removed when the privacy rule ends, expires or is deleted.

```
{
  "topic_name": "privacy_rule",
  "topic_uuid": "a79882ed-5aa0-451b-bd5b-c5147e801d0c",
  "value": {
    "target_topic": "domo_camera",
    "target_uuid": "d6422b30-f800-4be1-8fd4-ccc36a7efcb8",
    "time_start": "15:00",
    "time_end": "16:00",
    "days": [
      "Monday",
      "Wednesday",
      "Friday"
    ],
    "expiration_date": "2026/01/29"
  }
}
```


Figure 13: Topic of a privacy rule for the camera (source: [1])

During the validity of a rule, the Smartotum application does not start the stream and shows a message stating that access to the camera is temporarily disabled (Figure 14).

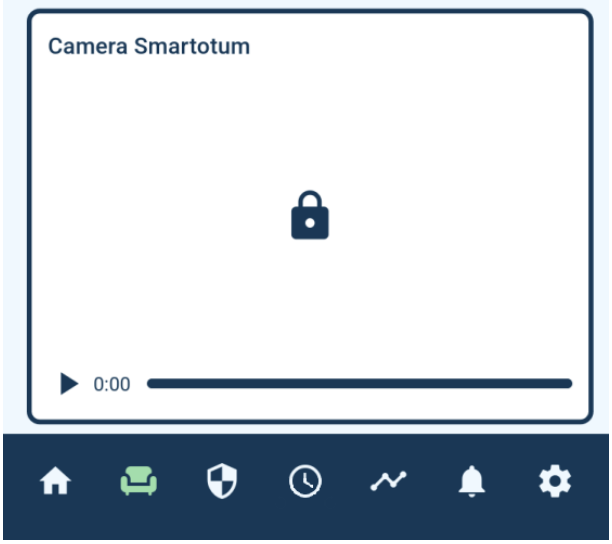


Figure 14: The camera blocked by a privacy rule in the Smartotum interface (source: [1])

7 Integration with the Privacy Dashboard

The Privacy Dashboard (D2.3) is the main access point for defining, managing and verifying the privacy rules applied to the devices of a smart home [6]. It is a separate component from the home-control interface, with a specific role in the governance of privacy. Its users are the inhabitants, as Data Subjects, and the actors responsible for the privacy of the applications, the Data Controller and the Data Protection Officer, who work at the level of the applications they own or manage and have no direct access to the home or to its devices.

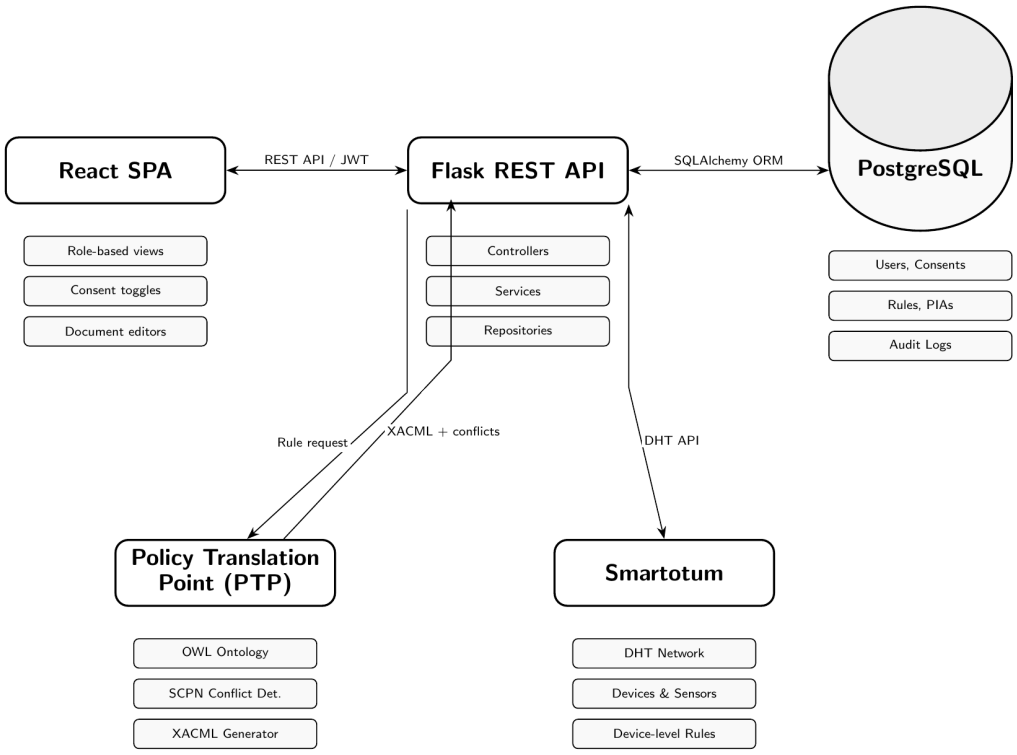


Figure 15: Architecture of the Privacy Dashboard and its integration with Smartotum

The dashboard interacts with Smartotum only through its backend and the DHT:

- **Authentication.** Inhabitants log in with their Smartotum account. The backend forwards the credentials to Smartotum and verifies the signature of the returned token against the public keys published by Smartotum (JWKS) before accepting the session (SPR-06). Access to the functions depends on the GDPR role (SPR-07).
- **Synchronisation.** A refresh operation retrieves from Smartotum all the homes to which the user has access, their rooms and devices, and the installed applications with their consents and the corresponding home-automation actions (FR-11).
- **Rule publication.** Rules defined in the dashboard are published as `privacy_rule` topics in the DHT, with the same structure used by the privacy manager (FR-10). A rule that targets the whole home is stored once in the dashboard and published as one derived rule for each compatible device of the house. The rules can be retrieved per device or per device category, querying the DHT by `target_topic` or `target_uuid`.
- **Consents.** When an inhabitant withdraws a consent linked to a home-automation action (for example video recording by the camera manager application), the backend creates the corresponding automatic deny rule, publishes it to the DHT and records the withdrawal (SPR-05). When the

consent is given again, the automatic deny rule is removed from the DHT before the consent is marked as given.

The dashboard does not interact with devices or with the home-control interface: its role is limited to the creation and management of rules, which are then interpreted by the privacy manager and the other components through the DHT. This keeps the functional control of the home and the governance of privacy clearly separated.

8 Deployment on the Smartotum Gateway

The integration of the components in Smartotum followed the existing architecture of the platform (NFR-06, NFR-08):

- the smart light firmware is configured with the SSID and password of the Wi-Fi network created by the gateway and with the address of its MQTT broker;
- the controller and the privacy manager are built as Docker images for the Smartotum gateway (OpenWrt-based, ARM64) with the build scripts and Dockerfiles provided in their repositories;
- the containers are configured to start automatically when the gateway is switched on; the privacy manager runs with host networking and the network administration capability needed to manage the firewall rules, and its entry point restarts the service if it terminates abnormally;
- both services reach the local DHT of the gateway, whose address and port are set through environment variables.

The Privacy Dashboard is deployed separately with Docker Compose (frontend, backend and PostgreSQL database) and connects to the Smartotum APIs and to the DHT of the homes of its users.

9 Repositories and Licence

The software of the pilot is publicly available on GitHub under the MIT licence.

Table 5: Repositories of the pilot software

Repository	**Content**
github.com/Assistants-Project/demo-light	Smart light firmware for ESP32-C3, mock binary, wiring diagram (Fritzing)
github.com/Assistants-Project/smartotum-controller	Smartotum controller, Dockerfile and build script for the gateway
github.com/Assistants-Project/privacy-manager	Privacy manager, Dockerfile and build script for the gateway
github.com/Assistants-Project/Privacy-Dashboard	Privacy Dashboard: backend, frontend, documentation of the Smartotum integration
github.com/ToscaLabs/tosca	tosca framework (firmware, drivers and controller crates, examples)

10 Conclusions

The ASSISTANTS pilot was implemented on Smartotum by adding a small number of components that use the DHT as their only integration point. The tosca framework provides the Rust APIs for firmware and controllers; the smart light demonstrates them on an ESP32-C3 device with three operating modes; the Smartotum controller makes ASSISTANTS devices visible and controllable in the Smartotum interface; the privacy manager turns the rules defined by the inhabitants into restrictions on the devices and, for cameras, into firewall rules on the gateway. The Privacy Dashboard closes the chain from the user, who expresses a rule or withdraws a consent, to the device, which is actually blocked.

The integration was straightforward and consistent with the existing architecture: the new services run as containers that start with the gateway and do not require changes to the platform. The software is open source and reusable. The evaluation of the pilot is reported in D3.3.

References

[1] Smartotum, *Specialist consultancy service in home automation for the implementation of security policy enforcement mechanisms in smart home systems*, final report for the ASSISTANTS project, signed by D. De Guglielmo.

[2] tosca framework, https://github.com/ToscaLabs/tosca.

[3] Assistants-Project, demo-light, https://github.com/Assistants-Project/demo-light.

[4] Assistants-Project, smartotum-controller, https://github.com/Assistants-Project/smartotum-controller.

[5] Assistants-Project, privacy-manager, https://github.com/Assistants-Project/privacy-manager.

[6] Assistants-Project, Privacy-Dashboard, https://github.com/Assistants-Project/Privacy-Dashboard.

Glossary

Acronym	Meaning
API	Application Programming Interface
DHT	Distributed Hash Table
DPO	Data Protection Officer
JWKS	JSON Web Key Set
JWT	JSON Web Token
mDNS-SD	Multicast DNS Service Discovery
MQTT	Message Queuing Telemetry Transport
OLED	Organic Light-Emitting Diode
PIR	Passive Infrared
REST	Representational State Transfer
VPN	Virtual Private Network