AAL Programme

AAL-CP-2018-5-149-SAVE

01/09/2019 - 31/08/2022

AAL Programme

Project - SAfety of elderly people and Vicinity Ensuring - "SAVE"

Deliverable: D.1.3. Cloud infrastructure

Version: 1.0

Main editor: VS

Contributing partners: ISS, UNITBV

Table of contents

1. SAVE solution overview

The SAVE solution is built using the latest technologies for applications designed to run in cloud environments (in an IAAS context) and taking advantage of what containerization has to offer (less overhead, increased portability, more consistent operation, greater efficiency in terms of scalability and velocity, better application development [2]).

An overview of the SAVE solution is depicted synthetically in Figure 1.

The sensors included in the SAVE kit (wearable, ambient and biological) provide raw data about the elderly (end-user) (data about his/her well-being, activity, environment, location) by connecting through Internet to dedicated services of the SAVE cloud application. At this moment, the S.O.S. and fall detection features are not mediated by the SAVE cloud application. The smartwatch chosen for the pilot phase and considered for the SAVE kit offers
out-of-the-box these two features. Because of the importance and interest shown by the end-users (as highlighted by the results of WP2) for these two features, a smartwatch with a direct GSM (LTE) connection was selected, to be independent of WiFi or other device constraints (e.g. the existence of a smartphone connected to the smartwatch). However, work has been planned and is still being carried out to implement a cheaper, alternative solution for these two features, that will be governed by the SAVE cloud application.

The users of the SAVE solution have dedicated user interfaces, in form of responsive web applications and mobile applications, for accessing its features:

- For the end-users: the SAVE smartwatch face and application and the SAVE web application
- For the caregivers: the SAVE web application
- For the SAVE solution maintenance staff and for the SAVE researchers: the SAVE Admin Centre web application.

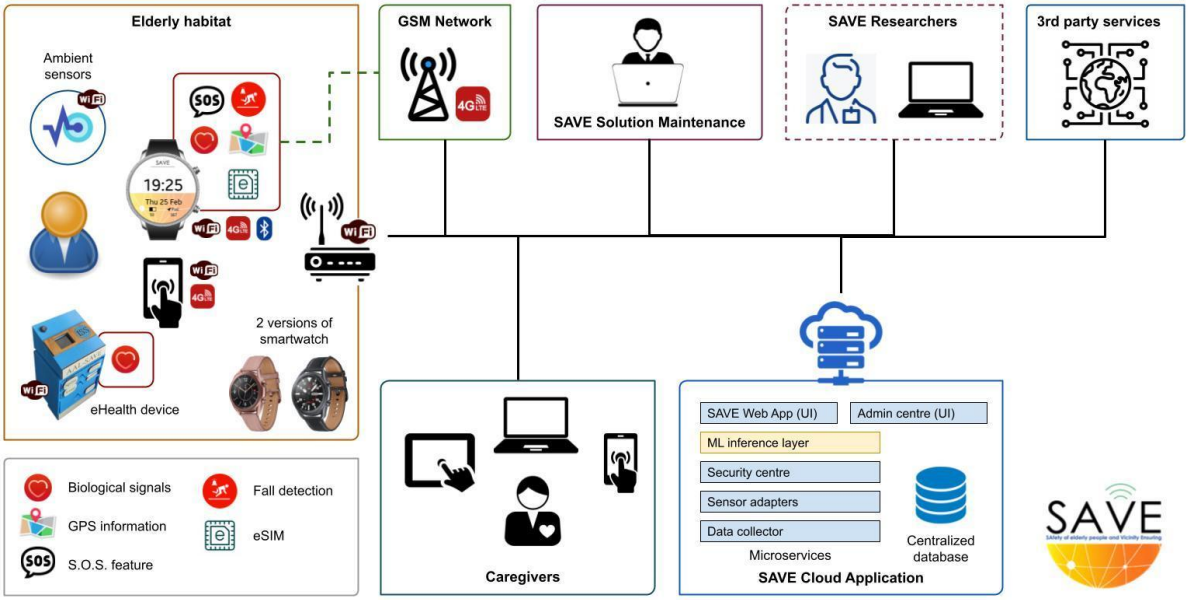


Figure 1 - SAVE solution overview

All web applications are responsive, so they can be viewed from a vast range of devices (desktops, laptops, tablets, smartphones).

The SAVE cloud application is split in several functional units (in fact microservices), that deal with particular aspects of the business logic:

- Data collector: collects data coming from sensors and stores them inside the database. It also offers a communication interface (a REST API) for retrieving this data.
- Sensor adapters: are actually an array of micro-services that connect directly to the sensing devices and relays their data, in a standardized format, to the Data collector.
- Security centre: offers REST APIs for authentication and authorisation operations.
- SAVE Web App: is the web-based user interface for end-users and caregivers.
- SAVE Admin centre: is the web-based user interface for maintenance staff and researchers.
- ML inference layer: is a machine-learning powered service that analyses the data collected by the sensors and assesses the wellbeing of the end-users (by conformity with the current profile of the end-user). The development of this layer it's forecasted for the immediate period.

A relational database management system (Oracle's *MySQL*) was used to store both the configuration and data of the SAVE software solution. To assure high speeds of access to the data, a partitioning algorithm was implemented at cloud application level (based on the kit identifier).

The SAVE solution was designed as a maximum inclusive system, so it can incorporate other sensor kits, developed by 3rd parties, with their own services, as long as they do not require permanent maintenance and configuration. Tests were made with the Xiaomi Smart Home Sensor Set in this context, with good results. Also, the Data collector service offers an inclusive REST API for connecting with ease any other sensor systems to the SAVE solution. Work is in progress to include other sensors in the SAVE kit, that will make use of this API.

2. SAVE system architecture

From an architectural point of view, the SAVE system is designed on a service-based architecture. This involves splitting the source code for services or features into independent, smaller software components that need to perform very specific tasks. These components, called microservices, are independent components, but they communicate with each other through a common, message-based language using a standardized interface. Usually, the communication protocol used by microservices is the HTTP (*Hypertext Transfer Protocol*) or HTTPS (*Hypertext Transfer Secured Protocol*) - www protocol, and the messages exchanged between services are independent of the technology used, the JSON (*JavaScript Object Notation*) format being usually used.

Microservices are an alternative to the monolithic application development system. The latter implies developing an application as a single unit. In the case of a monolithic application, all the components are strongly interconnected and thus the application becomes difficult to modify, maintain, but also to move into a cloud environment. Any changes made to the application will cause a process of creating another version of the system (Figure 2).

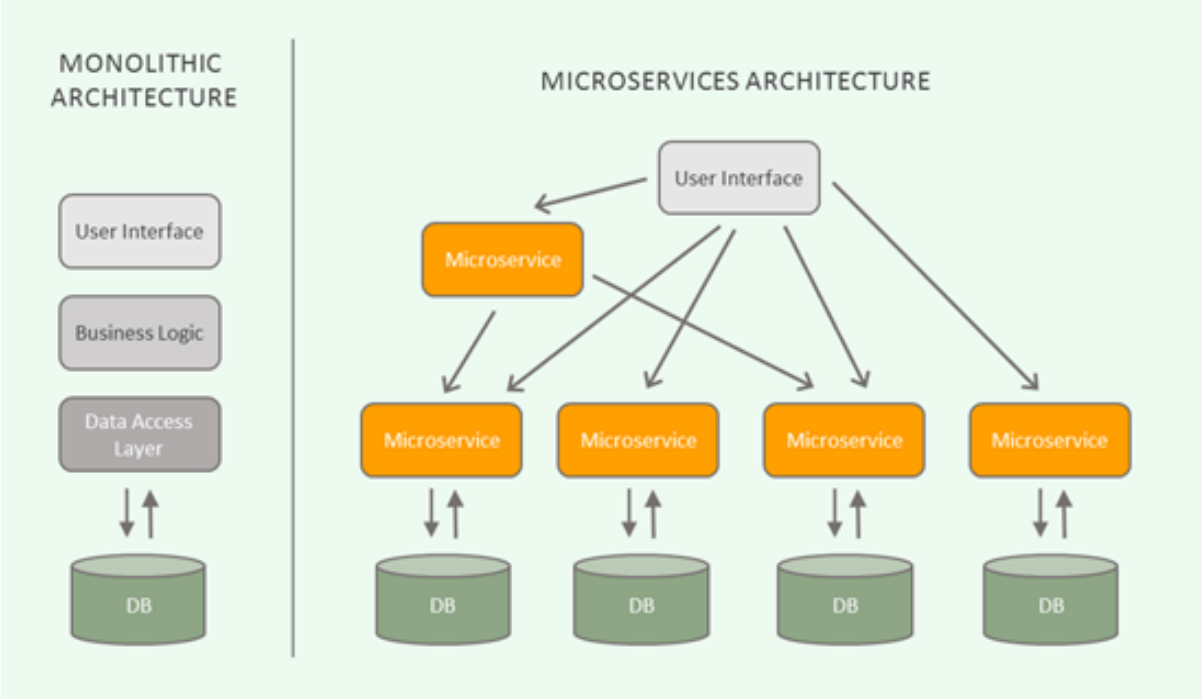


Figure 2 - Monolithic architectures – microservices

The use of a microservice-based architecture has both technical and implementation logic advantages, which make it easier and more efficient to develop applications in a cloud context:

- **Implementation** - there is more agility in delivering new versions of services due to the shorter times required for the construction and testing processes and implicitly the delivery times of the application.
- **Availability** - most of the time, the delivery of a new version of the application that uses the monolithic system requires the restart of the entire system, while the system based on microservices requires little downtime.
- **Efficient management** - since microservices behave as small portions of a whole, software developers who are part of the team will have the opportunity to work independently and with a higher degree of productivity, the team is also divided into smaller parts.
- **Possibility of modification and adaptation** - the greater flexibility in the use of different frameworks, data sources and libraries will allow the adaptation of the application as it develops.
- **Reliability** - in case of a fault, only one module is affected. Instead, failures in a monolithic system can cause the entire system to shut down.
- **Scalability** - microservices allow independent scaling of each component.

The general architecture of the system, as described in Figure 3, is composed of several independent components that communicate with each other to serve certain functionalities.

In this context, the microservices-based architecture of the SAVE system allows running on a cloud infrastructure, scalability, and speed in developing new facilities.

The system has several main components that are hosted in the cloud and perform specific tasks; it also allows interaction with external systems, provided by third parties, such as sensors, mobile devices or web applications that provide data and other monitoring applications that can process data retrieved from the SAVE system.

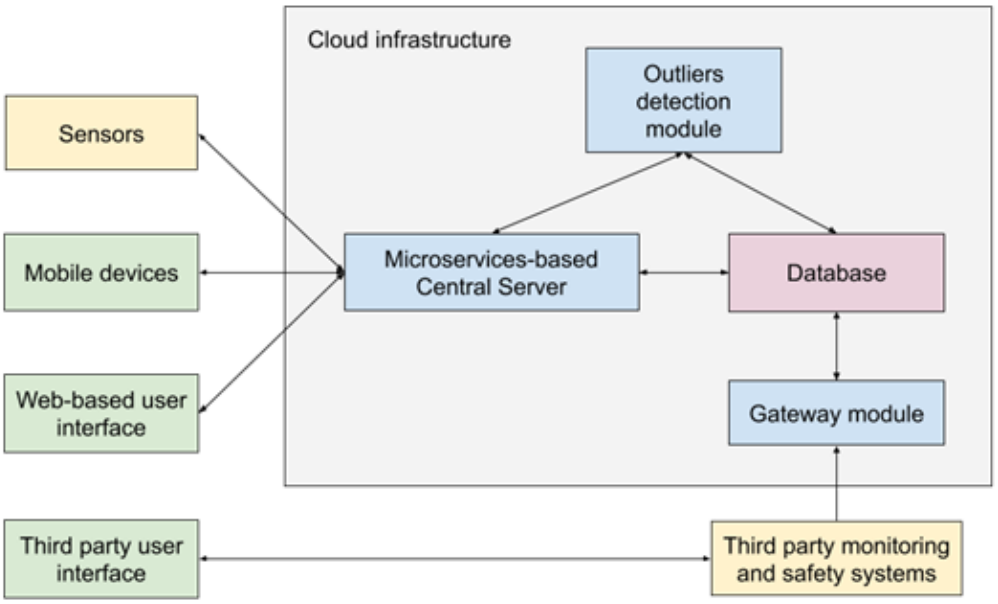


Figure 3 - General architecture of the SAVE system

Looking in detail, the main components of the system are represented by:

- The main microservices, which constitute a virtual central server
- Microservices for data processing (a machine learning algorithm find deviations from a pattern, anomalies, trends, etc.)
- Microservices for user interaction
- Microservices for interaction with external systems
- User interfaces
- Database management system

In terms of implementation, the essential components are developed in the Java programming language using the Spring Boot framework. This solution offers advantages for the development of microservices applications, their running and maintenance over time. Also, updating them is more efficient, it can be done quickly and independently, without affecting in any way other components.

All these microservices expose REST (Representational State Transfer) communication interfaces for interconnection, so they use the HTTP or HTTPS protocol. The messages they exchange with each other are standardized using the JSON format. Thus, communication between internal components and communication with external systems is easy to ensure.

3. The SAVE database

To ensure data persistence, a relational database management system (Oracle's *MySQL*) is used, providing a more efficient solution for organizing information in the system.

Figure 4 shows the organization of the collected data database tables.

The Kits are registered into the *kits* table and they receive a user-friendly name (field *kt_name*).

The recognized device types are stored in the *device_types* table; they receive an acronym and a default description. The acronym can be used by the user interfaces to know what component must be employed to handle the data came from that particular type of devices. Also, this acronym can be used for localizing the user interfaces; if the language dictionary does not contain the acronym, the default description will be used.

The SAVE devices are registered into the *devices* table. They can be included in one kit and are linked to a device type. The short and long descriptions are used on the user interface and are provided by the end-users, to make easier the identification and to discriminate between multiple sensors of the same type.

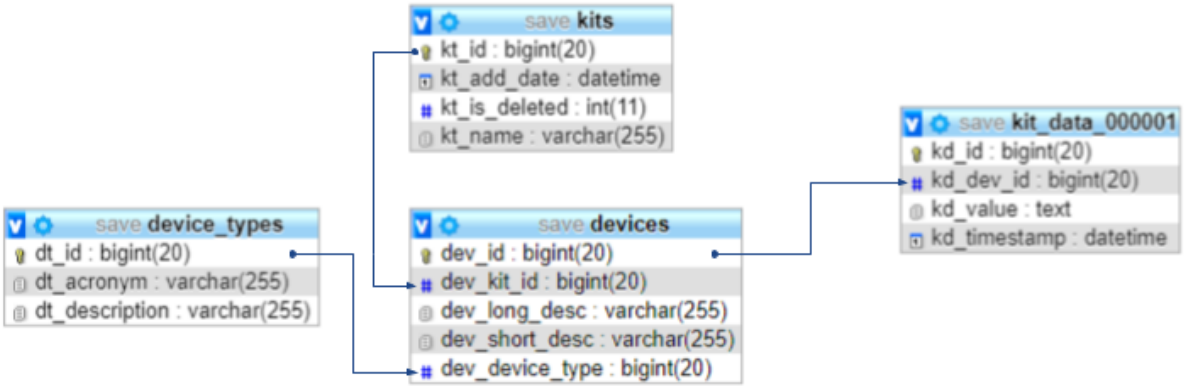


Figure 4 - Tables for the data collecting system

The values read from the sensors (the „data") are persisted, partitioned at kit level, in the „kit_data_*" tables; there is one table for each kit. The *kd_value* field contains the sensor data, as transmitted by the sensors itself or a sensor adapter.

Figure 5 shows the structure of *users* table and several other tables used by the microservices that implement the user interfaces and the link to the smartwatch applications. The password is not kept in clear text, but only the SHA-1 hash is stored.

The *not_data* table stores the scheduled notifications of the end-users. These are transferred periodically (30 seconds to 1 minute or soon as the smartwatch connects to the SAVE cloud application) to the smartwatch. The SAVE web application user interface manages these records.

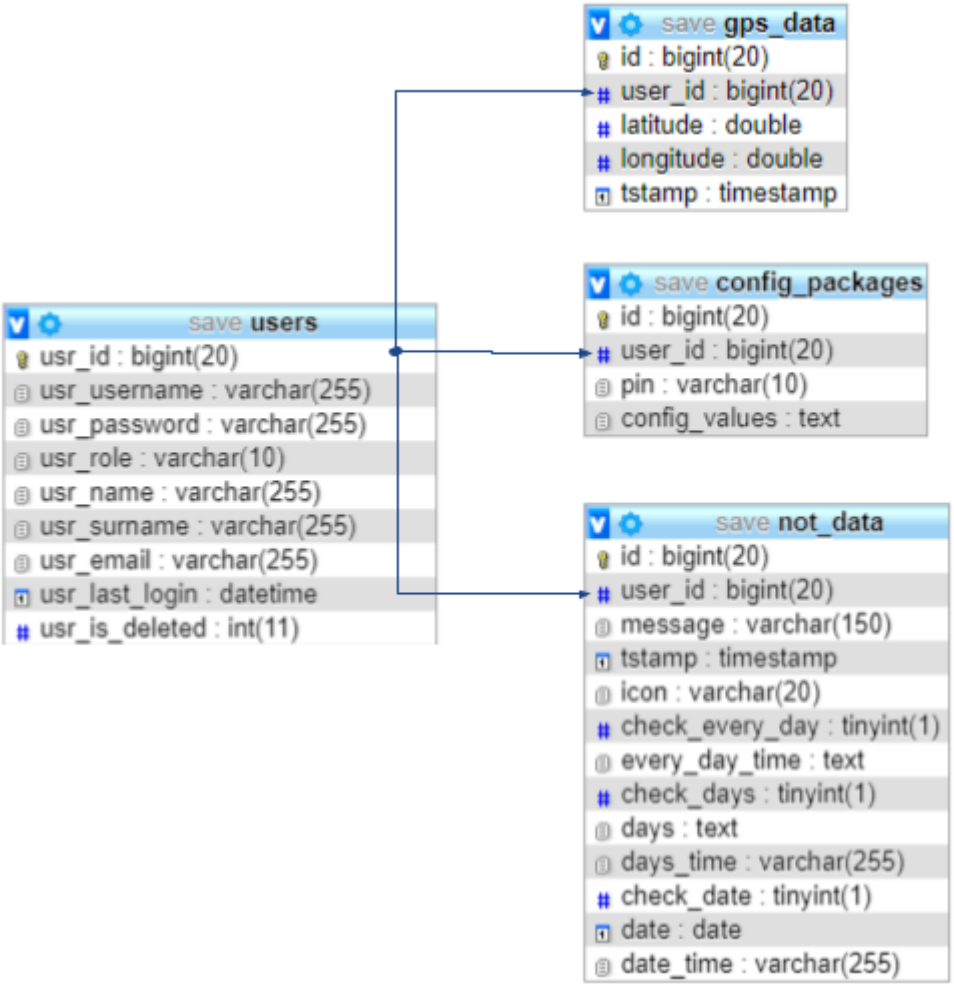


Figure 5 - Tables for the users, notifications, smartwatch app configuration and GPS data

It must be highlighted that the data coming from the sensors are not linked directly to the end-users, but to the kits. Accessing only this data does not reveal anything about the users' identities.

The config_*packages* table persists the settings of the smartwatch application. The records are automatically generated when an end-user is created in the SAVE Admin centre web application and contains the information needed to "pair" the smartwatch with the right user. When first installed, the SAVE smartwatch application asks for a pin, and based on this loads the corresponding configuration package from this table.

The temporary GPS data is collected in the *gps_data* table. The maximum duration for which the records are kept is 10 days. Work is in progress to reduce the number of the records, by eliminating consecutive records that have the coordinates closer than 2 meters.

4. The technologies

Figure 6 depicts the simplified technology stack for the microservices included in the SAVE cloud application.

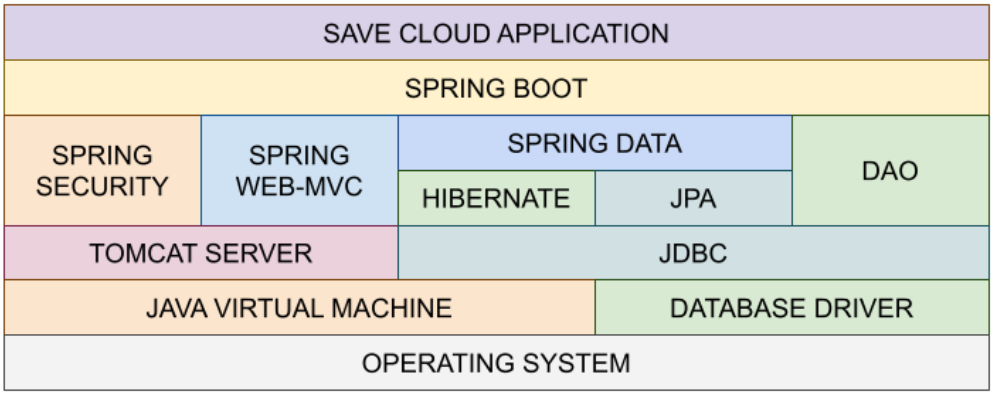


Figure 6 - Technology stack

The user interfaces (the frontend) of the web applications are developed using the *Angular* framework and the data is retrieved using the REST APIs of the backend (developed using the *Spring Boot* framework).

The smartwatch face application and the smartwatch application are Tizen web application, developed in HTL, CSS and JavaScript.

5. The Data collecting system - Data collector

One of the most important components of the SAVE system is the data collection service. This service is implemented as a microservice, to be easily scalable with the increase of number of sensors. The communication interface is a REST API (over HTTPS), that accepts JSON formatted data coming from devices that are registered inside SAVE's database.

The minimal structure of the JSON message accepted by the data collecting API is:

```
{
"kdDevId": <integer_value>,
"kdValue": <string_value>
}
```

When the sensors cannot provide the data in the accepted format, a data adapter must be implemented, so it will wrap the sensor's native format (binary, text, JSON, XML, etc) into the accepted structure. The eHealth devices developed by ISS have a data adapter running inside the SAVE cloud application that intermediates between the devices and the Data collecting system (*Data collector*).

As illustrated in Figure 7, the devices connect via the HTTPS protocol to a device type-specific adapter interface that transmits the data to the collection system which will persist it in the database.

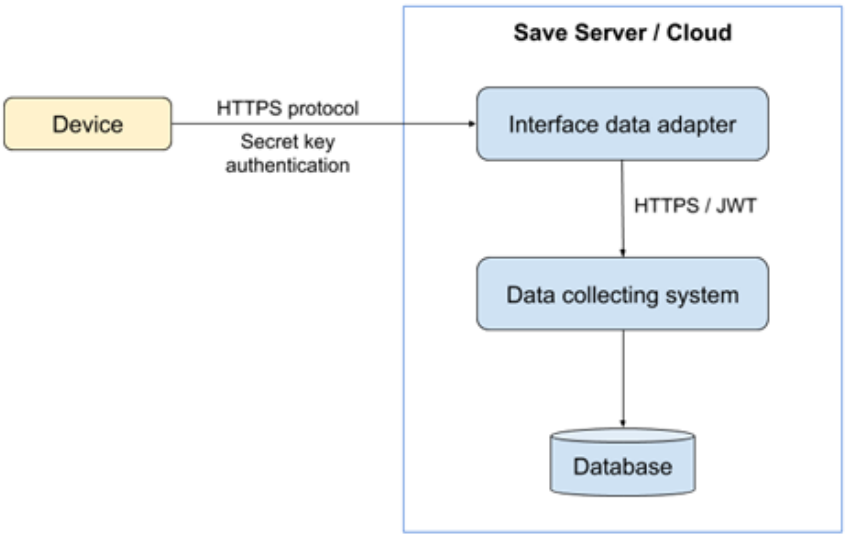


Figure 7 - System communication model

The data adapter interface is a software module that is developed specifically for each type of device recognized by the system. Its role is to expose a communication interface with external devices and to convert the format of the received data into a standard format recognized by the data collection component and to send it to the latter. Thus, the adapter streamlines communication between devices and system, abstracting the data format and providing it to be persisted. In this sense, the adapter is specific to the type of device with which it interacts and can be developed using any technology; it must, however, maintain the standard communication with the central system.

The communication between the adapter and the Data collector is done via the HTTPS protocol, doubled by an authentication based on a secret key (API Key).

The adapters are implemented as microservices, using the Java language and the Spring Boot framework.

Once in the system and transposed into a standard format, data from the devices is organized and stored. Figure 8 illustrates how the data is organized.

A device must be registered in advance; it is associated with a device type (which defines the corresponding adaptation interface). This device will receive a name so that it can be easily identified by users.

The collected data are stored in tables in the database. To optimize the speed of data access, they are partitioned by kits, so that the data from a kit is stored in the same table. The data storage mode is a general one, allowing the saving of any data structure.

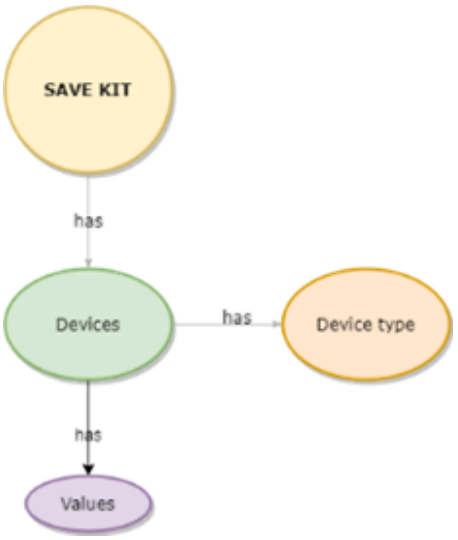
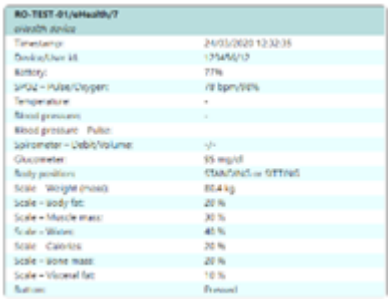


Figure 8 - Data organization in the SAVE system

6. Adapter interface for the eHealth device

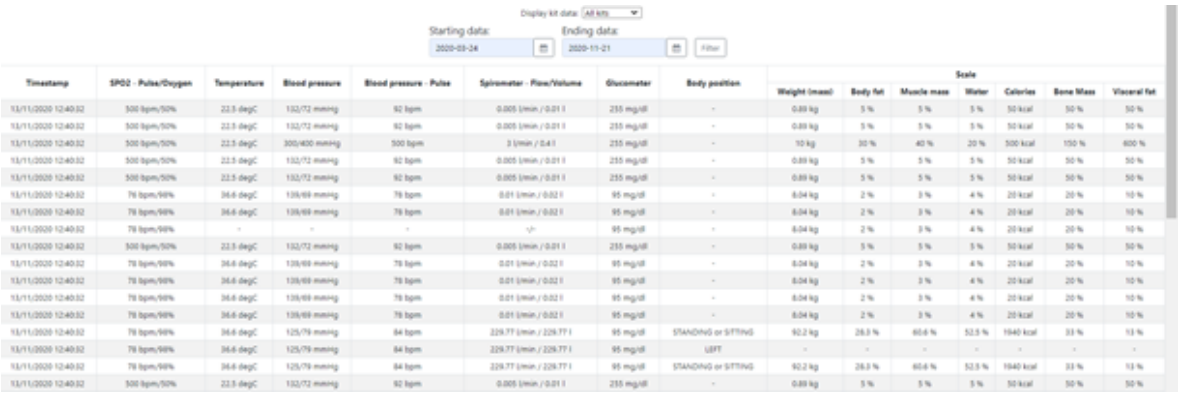
The first adaptation interface developed within the SAVE project ensures the connection of eHealth devices, developed by the ISS partner, to the data collection system.



a. Card – version 1

b. Card – version 2

c. Tabular view – unprocessed data



d. Tabular view - processed

Figure 9 - Data presentation for the eHealth device

Figure 9 shows 2 iterations for the Dashboard user interface and visualization of raw and processed data.

In addition to this adapter, a user interface has been included in the Admin Centre application, which allows you to view the data, both raw and tabular, processed. A card-based representation component has also been developed for synchronous tracking of data collection from the eHealth device.

7. Admin Centre web application

The Admin Centre is a web application developed to efficiently and securely manage data within the SAVE system. It is design mainly for the researchers and the system maintenance staff. The application offers a centralized display, the data being displayed in raw and tabular format, with attached meaning.

The Admin Centre was developed using the Angular framework and runs on an Apache Tomcat server.

The access to the application's features is protected using the authentication with the username-password pair and the authorization uses JWT (*JSON Web Tokens*).

The web application consists of several modules: Dashboard, RO Data, Data Kit, Kits, Devices, Device Types, Users.

The *Dashboard* module contains a synchronous view (pseudo real-time) of the collected data in the form of cards (Figure 10).

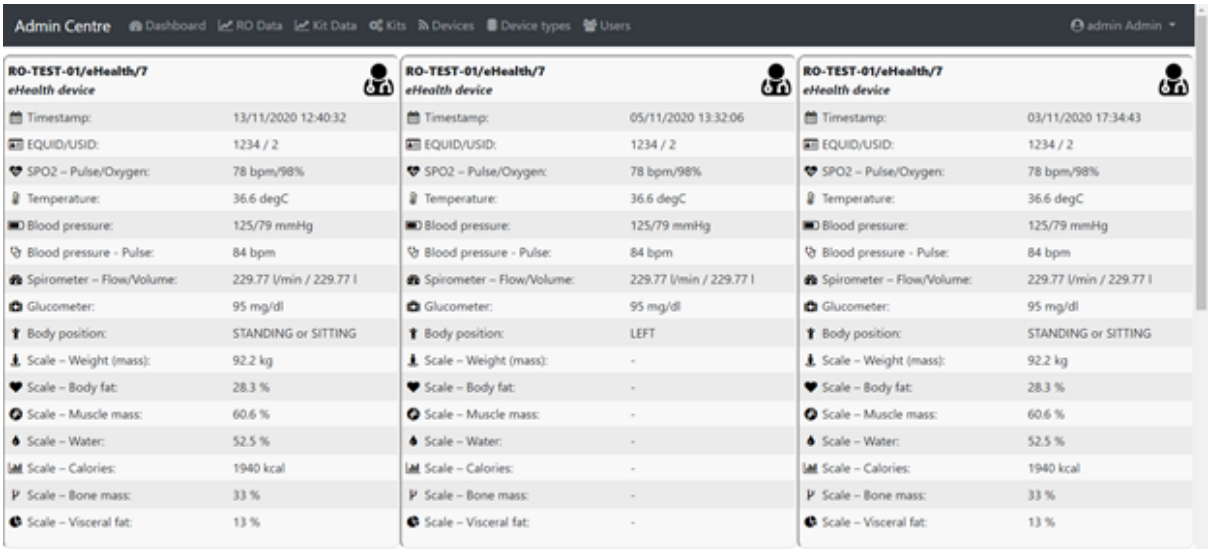


Figure 10 - The Dashboard module

The *RO Data* module allows the visualization of the data coming from the sensors, in raw format, as it is received from the adapter interfaces. It is a component intended only for SAVE system administrators (Figure 11).

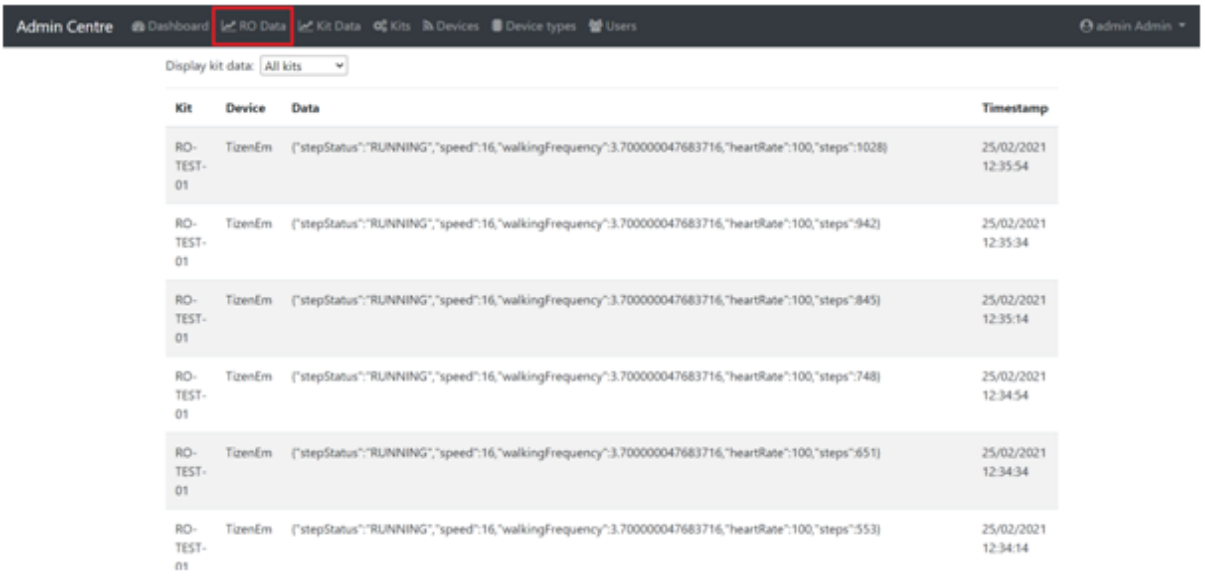


Figure 11 - RO Data module

The *Kit Data* module displays the data in tabular format, processed to be understood by a human operator (Figure 12).

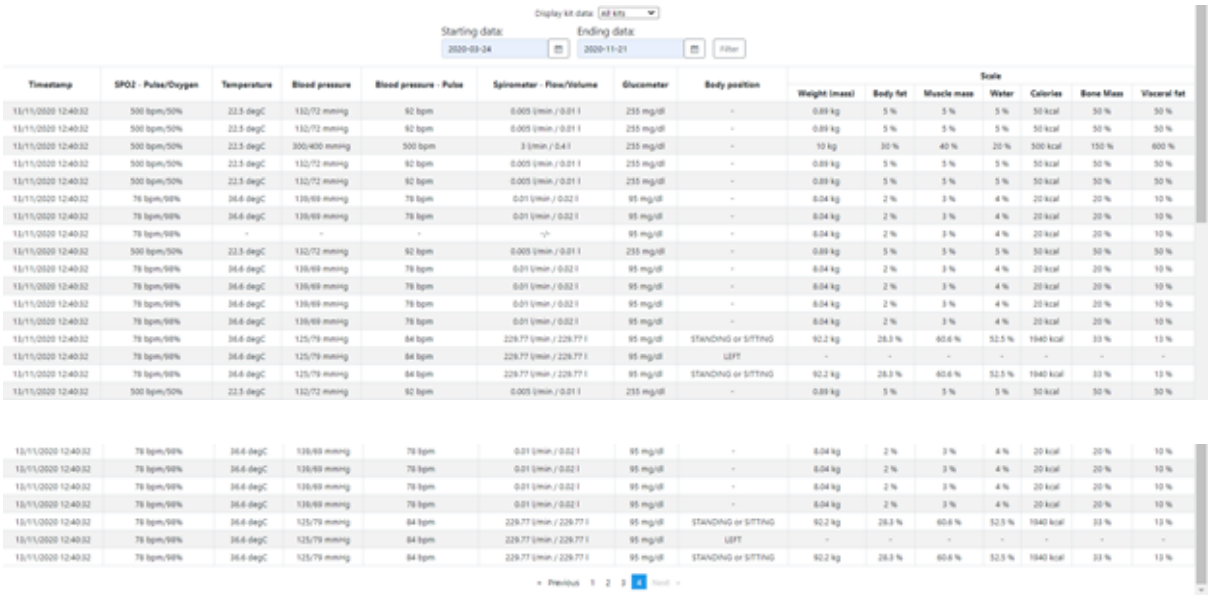


Figure 12 - Kit Data component

The *Kits* module offers the facility to manage the kits recognized by the SAVE system (Figure 13).

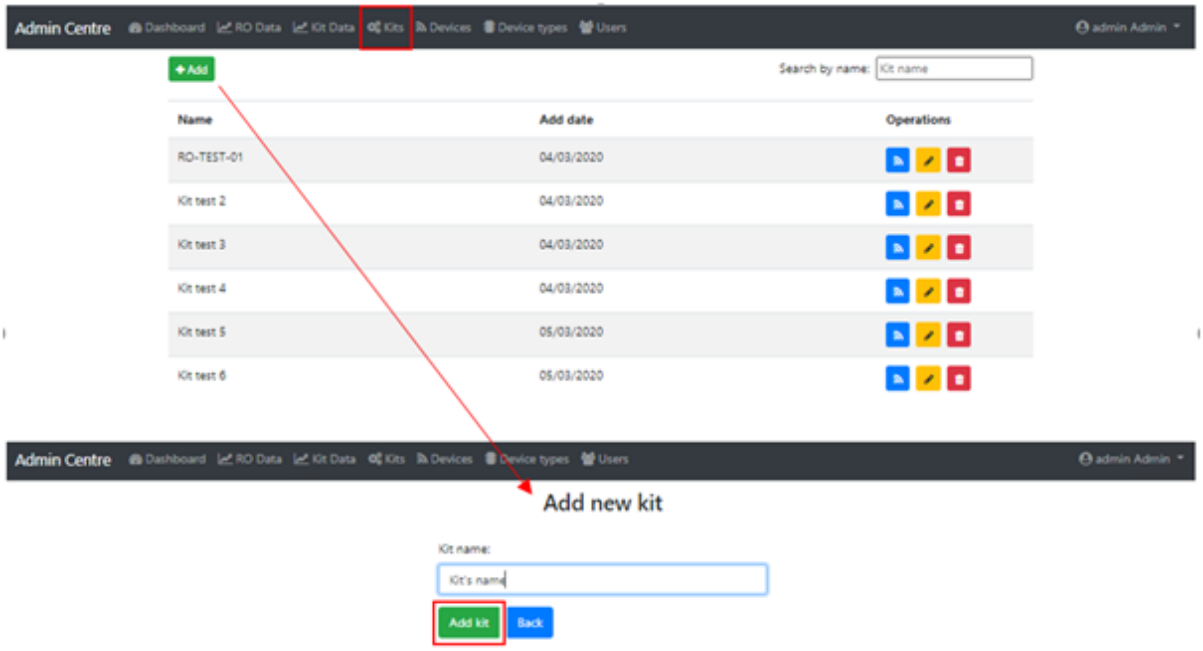


Figure 13 - Kits module

The *Devices* module allows the addition, editing and removal of a device and assigning it to a kit (Figure 14). These devices have two descriptions - one long and one short - that will be used by the user interfaces to allow them to be easily identified.

Data from unknown devices is ignored by the data collection system.

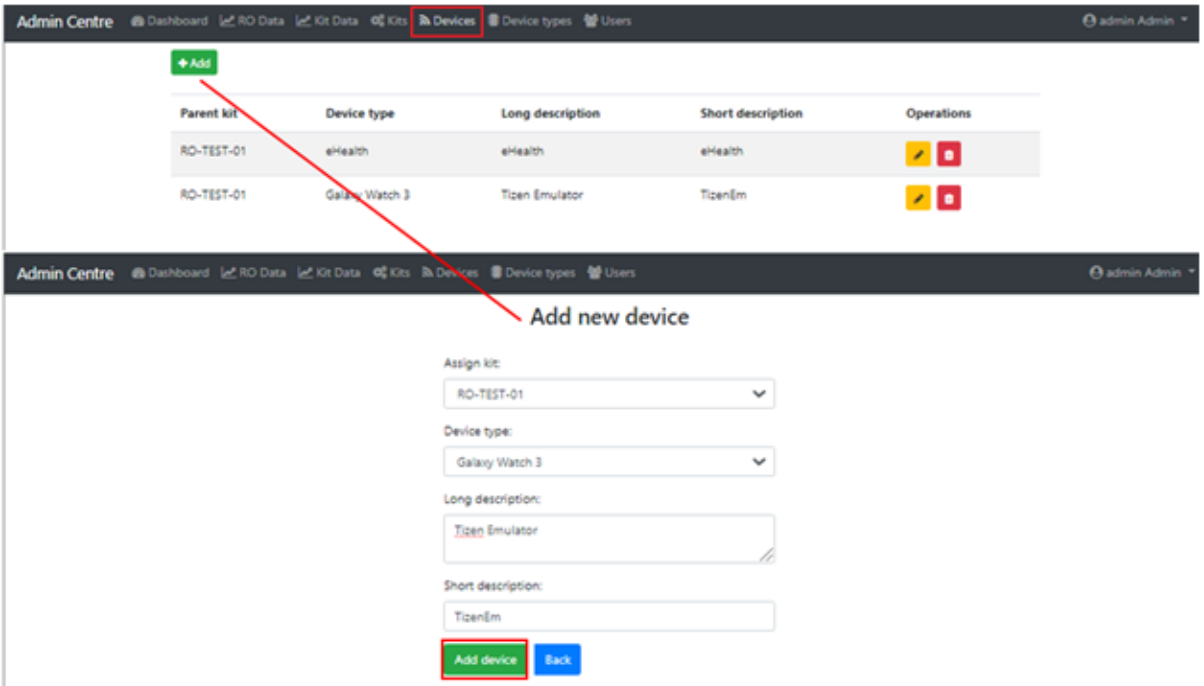


Figure 14 - Device module

The *Device types* module is the tool for the management of the types of devices recognized by the SAVE solution (Figure 15).

Figure 15 - Device types module

8. eHealth System - Cloud Input

8.1. Client - Server Communication Unit

Using the network card Wi-Fi SoC in order to send information to the Cloud server, the eHealth system will communicate via Wi-Fi with the user's home wireless router. The system is intended to communicate with the server using an encrypted channel with HTTP protocol and secured with SSL/TLS certificates, to send data only when it has real values recorded from the sensors and at a configurable period of about 10 seconds.

Client-Server communication is performed through a built-in SoC Wi-Fi ESP 8266 network card that ensures internet connectivity and allows data transmission to the server through an encrypted channel via HTTPS protocol. The server receives the data, processes it, stores it and displays it in an Admin Center application.

8.2. Software Relationship: eHealth System to Cloud

Currently, the data transmission, see Figure 16, to the SAVE Cloud has been successfully completed and the data transmission format has been established as a JSON template based on JavaScript principles. Prior to this step, the data transmission between PC and ESP device (HTTP POST) was emulated in order to simulate the data transfer into the SAVE Cloud. Finally, tests and demonstrations were performed to observe the operation of the system in real cases.

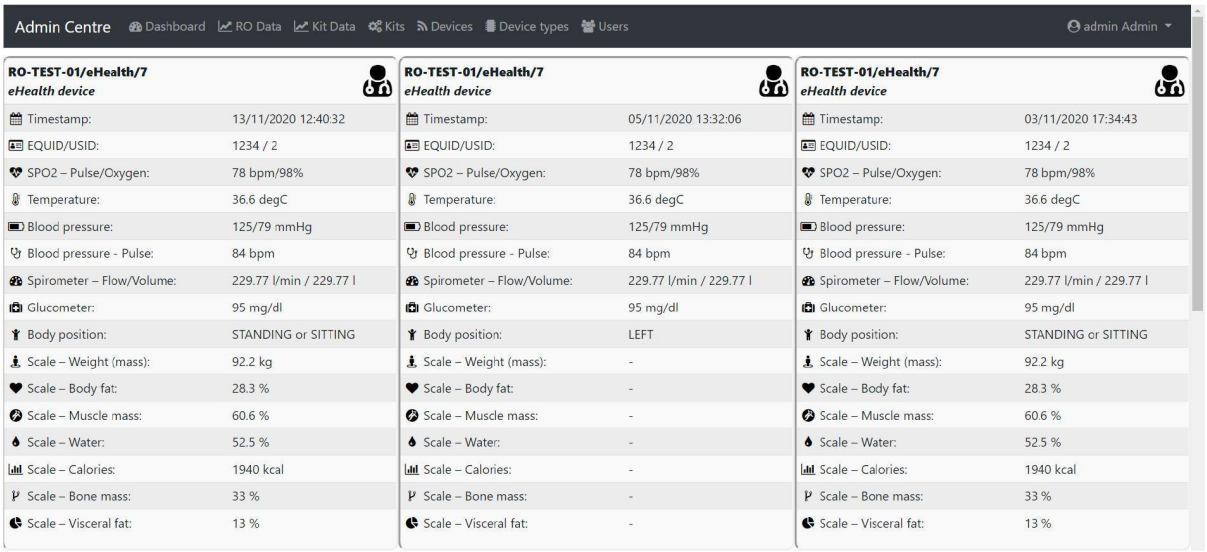


Figure 16 - SAVE Cloud interface for Admin Centre showing last 3 measurements
(in the middle one the scale was not used)

8.3. Hardware eHealth trade-off analysis relevant to cloud communication perspective

As an additional remark, regarding the secure connection of eHealth Unit and SAVE Cloud, Arduino UNO Wi-Fi Rev2 was the optimal option, despite the limited program memory. The compatibility of the Arduino UNO and Arduino UNO Wi-Fi Rev2 development boards (Figure 17) was achieved and any inconsistencies of the pins or at the architectural level were identified due to the different microcontrollers of the ATmega 328P and ATmega 4809 development boards. Changes were made to the software in order to be able to use the I2C data bus for the ATmega 4809 controller.

Figure 17 - Arduino UNO Wi-Fi Rev2 (microcontroller Atmega 4809)
Source: https://www.arduino.cc/Products

After several tests, **the decision was made to replace the Arduino Mega development board with Arduino UNO Wi-Fi Rev 2 (ATmega4809) for the following reasons: program memory is sufficient (it has 48K bytes and 6K bytes SRAM), integrated network board and options for easier implementation of the SSL/TLS encryption component**. Thus, it would no longer be necessary to use the external SOC Wi-Fi ESP card. In order to work properly, the internal driver for the SPI bus and the timer driver have been modified, but as specified before, Arduino UNO Wi-Fi Rev2 is limited in terms of the program memory, which makes the development board Arduino UNO Wi-Fi Rev2 can no longer be used.

9. Security aspects

Regarding the security aspects, the SAVE solution uses the current standards to protect the data and the access to its features.

The communication between the sensors and its adapter or the Data collector, and between the adapter interfaces and the Data collector is done by using HTTPS and the authentication by API key.

The web applications (SAVE and Admin centre) use authentication by the username-password pair, JWTs for authorization and work on HTTPS.

The passwords are not kept in clear text, but only the SHA-1 hash is stored in the *users'* table.

The data coming from the sensors are not linked in the database directly to the end-users, but to the kits. Accessing only this data does not reveal anything about the users' identities.

10. Deployment environment

The SAVE cloud application, being design on the microservices architecture and built using the Sprig Boot framework, can be deployed o vast number of infrastructures. The microservices can run independently of a servlet engine/web server (e.g. Tomcat) or can be deployed as a classic Java web application.

In the test environment, the microservices are deployed in a Tomcat server, installed in a virtual machine.

For the production environment, the microservices will be deployed using the Docker containerization engine and Kubernetes will be employed for the orchestration.

11. Research carried out on LISP

The "SAVE LBS" (Location-Based Services) are based on the state-of-the-art protocol LISP – "Location (from) Identity Separation Protocol".

To enable easier identification of an end-user in need LISP separates the ID of the wearable devices from their location – thus, direct connection to the device(s) is used and, in the case of multi-homing, reliability is improved by providing multiple paths for this data connection.

In the case that a mobile phone is used as a gateway for the sensors, the OpenOverlayRouter software is used, as it has versions for the Android OS and for the GNU/Linux OS (for the second one, there were ordered by UniTBv two Sony Xperia mobile phones that can run Sailfish and Lineage versions of Linux mobile). These mobile phones and their attached (directly or wireless) sensors are managed as IoT devices.

The main advantages of using LISP are:

A) Easiness for end-to-end installation in the network and on the Device

As expressed in RFC 6830 <<No changes are required to either host protocol stacks or to the "core" of the Internet infrastructure. The Locator/ID Separation Protocol (LISP) can be incrementally deployed, without a "flag day">>. So only some elements/routers in the network need “to know” the LISP protocol (the IoT Gateway in case of IoT), or only a mobile device, in case the LISP-MN option of the protocol in used. The rest of the network and the routing on the Internet remains unchanged. There are schemes for LISP to non-LISP addressing and LISP proxy implementations, so that LISP deployment is facile.

LISP is commonly used with IPv4 or IPv6, but it can be used with any other type of addressing for the EID or RLOC addresses, as the key to the routing is part of the DNS-like mapping servers (MS). For IoT this could bring a great advantage, thus the end-to-end addressing of non-IP IoT devices - like, for instance, IEEE 802.15.4 devices - could be possible directly in the IP header as long as IANA (Internet Assigned Number Authority) is supported, without the need of addressing the gateway.

B) Mobility

In case of IoT it can be used as host mobility the EID based addressing is a method for simplification and abstraction of network infrastructure and the RAN used as point of attachment in the network.

Separation between location and identifier is considered and acknowledged as the optimal method for user/host mobility. LISP mobility was previously compared to Mobile IPv6 mobility. It can be considered that also proxy mobility – PMIP, used in LTE networks, is a separation of edge mobility in relation to the core network, mobile hosts keep their IP address as long as they are part of the same proxy mobility domain.

Mobility management RFCs proposals also refer LISP as one candidate solution. “Mobility Management for 5G Network Architectures Using Identifier-Locator Addressing” specification describes the Mobility Management Architecture for 5G Networks Using Identifier-Locator Addressing in IPv6 for virtualized mobile telecommunication networks. Identifier-Locator addressing differentiates between

location and identity of a network node, and it is considered the key method for 5G mobility. [1]

The two main possible SAVE use cases for the protocol are:

- LISP is only implemented on the routers which serve the wireless connection on the locations with the advantage that no additional software is installed on the gateway devices
- LISP is implemented (via OOR) on the gateway devices (mobile phones) with the advantage that service is maintained even if the data connection switches from the wireless connection to the mobile data connections. In this case also a RTR node has to be used, due to the fact that most mobile operators are using NAT in their networks and the devices are not using a public IP address.

The configuration described below was firstly implemented in the *LISP beta network* that stopped providing services in May 2020. The UniTBv IT office provided a **lisp.unitbv.ro** server address and the SAVE team @ UniTBv is in the process of deploying an own LISP network, currently working to a specific security solution.

As mentioned above, LISP communications networks are able to facilitate the mobility of portable devices (smartphones, wearable devices etc) and their transition to IPv6. As the number of users (human or instrumental) is constantly growing, routing policies, multi-homing and traffic engineering have significantly increased the size of routing tables. The contribution of LISP to meet these challenges is to replace IP addresses with two very useful and at the same time new components. These two components are: RLOC (Route Locator) which describes how a device is attached to the network and can also be used to redirect packets over the network and EID (Endpoint Identifier) that which gives information about the identity of the device.

Implementing a communication network using the LISP protocol can create a complex, incrementally implementable, network-based architecture that allows service providers to simplify multi-homing routing, facilitate highly scalable network virtualization, support machine mobility related virtual resources in the Cloud and reduce operational complexities. The two factors that improve the scalability properties of Internet routing are: Multi Homing and Security. With the help of these two particularly important IP pillars there can be achieved goals that come to the aid of users such as reaching the availability of hosts through Internet connection in a proportion of almost 100%. Multi-homing refers to a host computer that has multiple IP addresses on connected networks.

A "multihomed" host is physically connected to several data links that may be on the same or different network. For example, a Windows platform with more IP addresses may be called "multihomed" and may serve as an IP router. Using Stream Control Transmission Protocol (SCTP), multihoming allows a single point SCTP to support multiple IP addresses, which means that one session is more likely to survive a network failure. In a single-homed session, a network failure can isolate the final system or make the transport temporarily unavailable.

Multihoming means that redundant local area networks (LANs) can be used to support local access. To reduce the effects of failures, various approaches can be taken, such as using addresses with different prefixes to force routing via different operators or even using redundant networks, which can reduce failure. It is commonly used in Web management for load balancing, redundancy, and disaster recovery. LISP is more than just a scalability solution; it provides both inbound and outbound traffic techniques, it can be used as a routing-level IPv6 transition and can be used for inter-domain multicast.

LISP has the potential to support Internet mobility of devices and to support the mobility of virtual machines in multi-location data centres and VPNs. These last two uses are not discussed further, as they do not apply to SAVE. Multi-homing can have three implementation methods: host level, classic multi-homing and multi-homing with multiple addresses. A single host can be connected to multiple networks. For example, a mobile phone may be connected to a WiFi and 4G network simultaneously, and a desktop computer may be connected to both a home network and a VPN. A multihomed host is usually assigned multiple addresses, one on a connected network.

In classic multihoming, a network is connected to several providers and uses its own range of addresses (usually from an independent provider). Network edge routers communicate with providers using a dynamic routing protocol, usually BGP (Border Gateway Protocol) which announces the network address range. If one of the connections fails, the dynamic routing protocol recognizes the failure in seconds or minutes and reconfigures the routing tables to use the remaining connections transparently to the hosts. This is expensive because it requires the use of the address space accepted by all providers, an autonomous public system (AS) number and a dynamic routing protocol. Because the multihomed address space cannot be aggregated, it increases the overall routing table.

In this approach, the network is connected to multiple providers and is assigned multiple address ranges, one for each provider. Hosts are assigned multiple addresses, one for each provider. Multi-address multihoming is cheaper than classic multihoming and can be used without any cooperation from providers (for example, in a home network), but requires additional technology to perform routing:

- For incoming traffic, hosts must be associated with multiple DNS A or AAAA records so that they can be accessed by all providers.
  A) DNS A means that the record returns a 32-bit IPv4 address, most commonly used to map the host name to a host IP address, but is also used for DNSBL, storing subnet masks. DNSBL (Domain Name System-based Blackhole List) is an effort to stop email spam.
  B) DNS AAAA means that the record returns a 128-bit IPv6 address, most commonly used to map the host name to a host IP address.

- For outbound traffic, a technique such as source-specific routing must be used to route packets through the correct provider, and reasonable source address selection policies must be implemented by hosts.

For LISP, most threats can be mitigated using current best practices, i.e. with careful implementation and configuration (e.g. filtering), by activating only the functions that are really needed, and by checking all the information. obtained from third parties. Unless cleaning functions are used, the LISP data plan has the same level of security as other IP-over-IP technologies. From a security perspective, the control plan remains the critical part of the LISP architecture. To mitigate threats to the mapping system, authentication for all messages in the control plan should be used. The current specification defines security mechanisms that can reduce threats in open network environments.

The LISP specification defines a generic authentication data field for control plan messages, which could be used for a general authentication mechanism for the LISP control plan, while remaining backward compatible.

LISP means a network architecture and a set of network-mapped and encapsulated protocols based on the requirements of the Internet Engineering Task Force (IETF) in RFC 6830 that documents IP address separation in two new numbering spaces: Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). By introducing this separation, new capabilities for mobility, scalability and security are made available. This approach contrasts with the traditional practice of using a single numbered address - an IP address - to represent both identity and location.

Packets reaching ITR (Input Tunnel Router) or PITR (Proxy Ingress Tunnel Router) cannot be encrypted, which means that no data protection or privacy is added. When the source host encrypts the data stream, the encapsulated packets must not be encrypted by LISP.

Addressing in the LISP architecture:

A) Endpoint Identifier (EID)
   An EID is an IPv4 or IPv6 address used in the source and destination address fields of the first (innermost) LISP header of a packet. In the case of IPv4 it has a value of 32 bits and in the case of IPv6 it is a value of 128 bits. These values are used in the source and destination fields of the first LISP header of a packet.

B) Routing Locator (RLOC)
   A RLOC is an IPv4 or IPv6 address of an output tunnel router (ETR). An RLOC is the result of an EID-to-RLOC mapping search. The RLOC namespace is used in the kernel. RLOCs are used as infrastructure addresses for LISP routers and ISP routers and are routed globally in the core infrastructure, just as they are today. The hosts do not know about the RLOC, and the RLOCs do not know about the hosts.

The main components of the LISP configuration are:

1) Ingress Tunnel Router (ITR)
   An ITR is a device that is the starting point of the tunnel; receives IP packets from the final systems of the site on the one hand and sends LISP encapsulated IP packets, on the Internet to an ETR, on the other side. An ITR is responsible for finding mappings between EID and RLOC for all traffic dedicated to LISP sites. When the ITR receives a packet dedicated to an EID, it first searches for the EID in its cache. If ITR finds a match, it encapsulates the packet inside a LISP header with one of its RLOCs as the source IP address and one of the RLOCs in the mapping cache entry as the IP destination. The ITR then routes the packet normally. If no entry is found in the ITR mapping cache, ITR sends a Map-Request message to one of its configured map resolvers and then discards the original packet. When ITR receives a response to its Map-Request message, it creates a new mapping cache entry with the contents of the Map-Reply message. When another packet arrives, such as a retransmission for the original packet and now discarded, the new mapping cache entry is used for encapsulation and forwarding.
2) Egress Tunnel Router
   An EID is an IPv4 or IPv6 address used in the source and destination address fields of the first (innermost) LISP header of a packet. An ETR connects a site to a LISP-capable part of a core network (such as the Internet), publishes EID-to-RLOC mappings for the site, responds to Map-Request messages, and decapsulates and delivers LISP-encapsulated user data to systems. final the site. During operation, an ETR sends periodic map recording messages to all its configured map servers. Map-Register messages contain all EID-to-RLOC entries for EID numbered networks that are connected to the ETR site. An ETR that receives a Map-Request message verifies that the request matches an EID for which it is authoritative, constructs an appropriate Map-Reply message that contains its configured mapping information, and sends this message to the ITR whose RLOCs are listed in the Map-Request message. An ETR that receives a LISP encapsulated packet that is directed to one of its RLOCs decapsulates the packet, verifies that the inner header is intended for a final EID counting system at its site, and then transmits the packet to the final system using the site - inner routing. The ETR feature is typically implemented in the customer-premises equipment (CPE) router and does not require hardware changes on software switched platforms, such as a Cisco Integrated Services Router (ISR). The same CPE router will often offer both ITR and ETR functions and, in this regard, is referred to as xTR.
3) Proxy Ingress Tunnel Router (Proxy ITR)
   A PITR is used for inter-networking between non-LISP and LISP sites. A PITR acts as an ITR, but does so on behalf of non-LISP sites that send packets to destinations on LISP sites. A LISP PITR implements database-based ITR searches and LISP encapsulation functions on behalf of non-LISP sites. PITRs are typically deployed near major internet exchange points (IXPs) or ISPs to allow non-LISP customers on those networks to connect to LISP sites. In addition to implementing ITR functionality, a PITR also

advertises some or all of the EID prefix space that cannot be directed to the non-LISP Internet part that it serves so that non-LISP sites will direct traffic to PITR for encapsulation and redirection to LISP sites.

4) Proxy Egress Tunnel Router (Proxy ETR)
   A LISP PETR implements ETR functions on behalf of non-LISP sites. A PETR is typically used when a LISP site needs to send traffic to non-LISP sites, but the LISP site is connected through a service provider that does not accept non-reusable EIDs as packet sources. PETR implements ETR functions on behalf of non-LISP sites. A PETR is typically used when a LISP site needs to send traffic to non-LISP sites, but the LISP site is connected through a service provider's access network that does not support unrecoverable EIDs as sources of packets. A PETR can also serve as a method for EIDs and RLOCs to communicate in a LISP site that contains EIDs in a family of addresses and RLOCs in a family of different addresses. A dual-stacked PETR also provides multi-address family support for LISP EIDs within an address family so that it can communicate with non-LISP destinations in the same address family through a core network within an address family. families of different addresses.

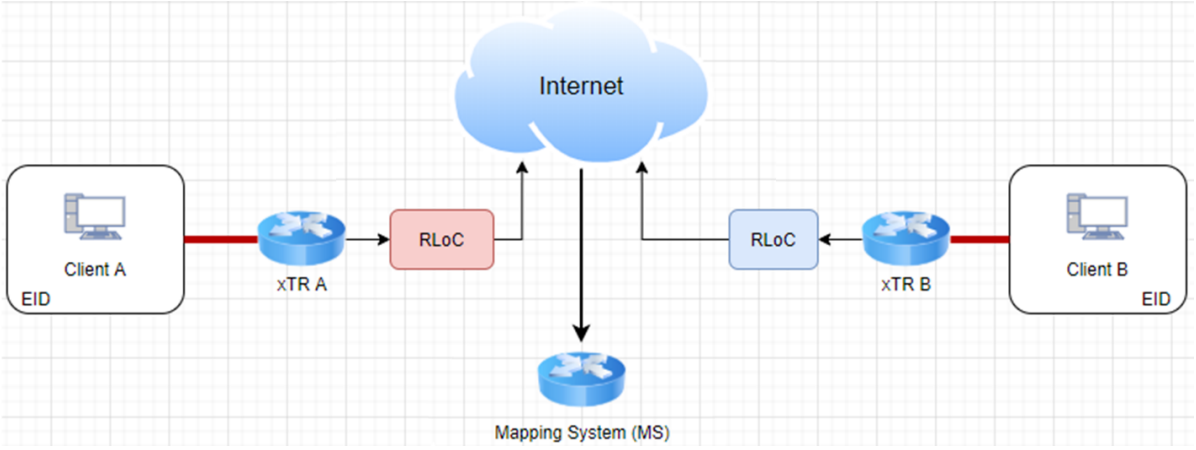


Figure 18 - LISP topology sample

REFERENCES

[1] Balan, T., Robu, D., & Sandu, F. (2016). LISP Optimisation of Mobile Data Streaming in Connected Societies. Mobile Information Systems, 2016

[2] What are containers, Available: https://www.netapp.com/devops-solutions/what-are-containers/