CALL H2020-SC1-FA-DTS-2018-2020
Trusted digital solutions and Cybersecurity in Health and Care
TOPIC DT-TDS-01-2019
Smart and healthy living at home

SMART BEAR

"Smart Big Data Platform to Offer Evidence-based Personalised Support for Healthy and Independent Living at Home"

D3.2 – Report on SMART BEAR @ Home Enabling Components v1

Due date of deliverable: 31/01/2021
Actual submission date: 31/01/2021

Grant agreement number: 857172
Start date of project: 01/09/2019
Revision

Lead contractor: CNR
Duration: 48 months

Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020	
Dissemination Level	
PU = Public, fully open, e.g. web	✓
CO = Confidential, restricted under conditions set out in Model Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC.	
Int = Internal Working Document	

D3.2 – Report on SMART BEAR @ Home Enabling Components v1

Editor
Fanis Kalatzis

Contributors
Fanis Kalatzis
Kostas Tsiouris
George Zissis
Giorgos Giotis
Mattia Occhipinti
Jonatan Maggesi
Paolo Ceravolo
Valerio Bellandi

Reviewers
Stéphane Railhet
Dimitrios I. Fotiadis

The work described in this document has been conducted within the project SMART BEAR, started in September 2019. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857172

Executive Summary

This deliverable presents the initial effort that was allocated during the first reporting period, to create an out of the box federation of devices and sensors that allows seamless integration for the creation of the smart automation of Smart Bear. The architecture and applied technology allow personal and home-enabled sensors to monitor a variety of measurements by minimizing user intervention through a centralized interface provided by the Smart Bear specific mobile application. The deliverable is composed of 6 chapters, each of which considers different tasks and activities.

At the introductory chapter is presented an overall view of the scope and effort's distribution to face the challenges and objectives of the work package 3.

Chapter 2 presents the home automation that is build using open-source technologies with the ability to overcome vendor specific communication and connectivity protocols; by supporting "off-the-shelf" devices our effort was focuses to vendor agnostic technologies from the design of the project.

Chapter 3 presents the initially developed mobile application (Android) of Smart Bear including the key technological frameworks and integration schemes to efficiently monitor the personalized heterogeneous devices. Vendor provided APIs and SDK are used to link each monitoring device with the mobile application and offer a unified solution (the SB@App) that minimizes the user intervention with the devices. The solution focuses on the automated personalized tracking and guidance.

Chapter 4 is an additional request by the reviewers' board, that presents all devices and sensors followed by the measurements, field types, data values and metadata where available. The chapter and its displayed material make easier the management of the data flow for all work packages.

Chapter 5 provides the design, the methodology and the technology of the Dashboard as a user-friendly component that will be used by the clinicians of the platform to support the management of the persons' data.

Finally, the Appendix of this document contains supplementary, secondary and more technical information that is needed to reference the main details of each chapter.

Contents

Definitions & Acronyms

Term	Meaning	Term	Meaning
IoT	Internet of Things	USB	Universal Serial Bus
openHAB	open Home Automation Bus - "open source, technology agnostic home automation platform which runs as the center of your smart home"	MQTT message	A formated structure of the MQTT protocol used for the machine-to-machine conversation on a topic
Bindings	are the openHAB component that provides the interface to interact electronically with devices	API	Application Programming Interface
Things	are the first openHAB (software) generated representation of your devices	REST	Representational state transfer
Channels	are the openHAB (software) connection between “Things” and “Items” (see below)	IP	Internet Protocol
Items	are the openHAB (software) generated representation of information about the devices	URL	Uniform Resource Locator
openHAB Rules	that perform automatic actions (in its simplest form: if "this" happens, openHAB will do "that")	HTTP	Hypertext Transfer Protocol
Sitemap	is the openHAB (software) generated user interface (web site) that presents information and allows for interactions	HTTPS	Hypertext Transfer Protocol Secure
openHABian	A self-configuring Linux system setup specific to every openHAB user	APK	Android Package Format for Android Applications
VPN	Virtual Private Network	JSON	Javascript Object Notation

LAN	Local Area Network	ECG	Electrocardiogram
Vue	A progressive framework for building user interfaces	BP	Blood Pressure - sBP=Systolic Blood Pressure, dBP=Diastolic Blood Pressure
JS	JavaScript programming language	BMI	Body Mass Index
Zigbee	A standard for low power, self-healing, mesh networks offering a complete and interoperable IoT solution for home and building automation	SPO_2	Peripheral capillary oxygen saturation
Zigbee2MQTT	A software platform that allows to use your Zigbee devices without the vendors bridge or gateway	UI	User Interface
MQTT	Message Queuing Telemetry Transport is a lightweight messaging protocol designed for machine-to-machine communication in low bandwidth environment	HA	Hearing Aids
MQTT Broker	Is synonym of an MQTT server	MVC	Model-View-Controller
OSGi	A specification that describes a modular architecture for the Java programming language	IDE	Integrated Development Environment
Apache Tomcat	A Java based Application Server that implements the web profile of the JAVA EE	SB	Smart Bear
JAVA EE	A full set of enteriprise specifications implemented in the Java programming language	OS	Operating System
JAKARTA EE	JAKARTA is the new name of JAVA after JAVA EE adopiton from Eclipse Foundation	FCM	Firebase Cloud Messaging
Spring BOOT	A Java framework (a "competitor" of JAVA EE)	GDPR	General Data Protection Regulation

CORS	Cross-Origin-Resource-Sharing	AES-256	Advanced Encryption Standard with 256 bits key length
SSL	Secure Sockets Layer	BLE	Bluetooth Low Energy
TLS	Transport Layer Security	hPA	hectopascal unit of pressure measurement
Mosquitto	An Open Source MQTT server that implements the MQTT protocol	CLI	Command Line Input
Philips Hue Bridge	A device that controls the Philips home automation	FHIR	Fast Healthcare Interoperability Resources
Xiaomi Aqara Gateway	A device that controls the Xiaomi home automation	JPA	Java Persistence API
Raspberry PI	A small hardware device that operates as an autonomous computer for low-demanding computational labour	HR	Heart Rate
ACL	Access Control List	bpm	Beats per minute
oAuth	Open-source authentication scheme that defines explicitly the delegated authentication process	mm	mili-meters, mili equals one of thousands
MQTT topic	A topic is string (a word) that the broker users to filter messages from the client	ISO	International Standards Organization
Vuex	A state management library for Vue.js applications	HL7	Health Level Seven International (HL7) is a not-for-profit, ANSI-accredited standards developing organization

List of tables

List of figures

1 Introduction

This deliverable (D3.2) of Smart Bear aims to present the effort that was carried out during the first period of the project regarding the devices, the sensors, the software and their corresponding technology. The overall components architecture and the main terminology were presented in the deliverable of the Smart Bear architecture "D2.2 (D6) – SMART BEAR Architecture". In this deliverable it is presented the current selected infrastructure including devices, sensors, mobile application, users' dashboard that will be installed to the home and used by the elderly population. Any device that is used, installed and integrated is an "off-the-shelf" product aiming to be immediate operational for any user and furthermore allow measurements' integration with the project-specific Smart Bear software environment.

In order to facilitate the final selection of the devices preliminary testing with devices and sensors from different vendors was performed. Some devices were not appeared so much "user-friendly" from the first point of view, regarding the fact that the users will be elderly person and not "enough" familiar with the handling of mobile and portable technologies; prior to this deliverable a more specific effort was performed and included in another project-specific deliverable that describes the process and the selection of the devices.

The key point of the Smart Bear project aims to the provision of a unique mobile application - that is presented in this document - aiming to integrate and unify all "off-the-shelf" devices in order to be as much as possible user-friendly and simple to use; a key task is to minimize the user intervention to only charging and powering on and off every device. The elderly persons will not use each vendor's application that requires registration, device tuning, web portal connection, software updates from desktop applications etc.; and of course it will not be obliged to read, learn and apply the vendor specific instructions for each device. In this regard the Smart Bear technology is creating a HomeHub platform and a Mobile Application (SB@App) that both will minimize user interventions and consequently minimize any cost.

The HomeHub is hardware and software integration that unifies all home sensors and it is installed in the house of the persons. The HomeHub communications are based on Zigbee standard which is "the only complete IoT solution - from mesh network to the universal language that allows smart objects to work together"[1] that uses a low-power radio technology IEEE 802.15.4-2011[2].

The mobile application (SB@App) of Smart Bear provides:

- a unique control-center that bi-directionally communicates with the HomeHub (unified home devices & sensors integrator) and
- the integration and data handling of the devices and sensors that are capturing physiological and health related measurements.

The SB@App sends all measurements to the Smart Bear cloud (SB@Cloud) to be stored and processed but can also receive personalized information to inform, alert and notify each person.

[1] https://zigbeealliance.org/solution/zigbee/

[2] https://standards.ieee.org/standard/802_15_4-2011.html

The overall integration is also supported by the user's dashboard with local analytics and developers' monitoring and access platforms.

To efficiently handle and manipulate installations and user needs this deliverable - and its forthcoming updates - are reporting both the users' and the administrators' software tools applied at the HomeHub, the SB@App and the User's Dashboard that are presenting in the chapters of this deliverable.

To comply and satisfy the contractual requirements the Table 1 and Table 2 are presenting the fulfillment of the effort in comparison with the requested obligations.

WP3 Objectives as described in the Grant Agreement	Actions Done
To provide efficient and secure out of the box federation of smart devices present in the SMART BEAR eco-system;	Selected the openHAB software platform as a vendor agnostic integrator. Also the Zigbee2MQTT software system was applied. Both systems satisfied the needs beyond proof-of-concept expectations.
to provide monitoring capabilities for end user activities, their conditions and the interventions applied; to provide the local analytics needed to infer higher level information from the raw data monitored;	The selection of devices was made by accounting each device functionality and API in order to support monitoring capabilities and analytics from the SB@App without special vendor-specific interventions (e.x. access the vendor site to retrieve measures).
to summarize data prior to transmission to the backend cloud;	All devices were selected by the requirement to offer an SDK and an API. By this way, the SB@App is developed with application-specific data handling per device. The HomeHub (openHAB) software platform provides programmatical capabilities to handle all available device and sensor values.
to develop the SMART BEAR mobile application and associated submodules;	This deliverable presents an active working version of the SB@App; the application is under development.

Table 1: Correspondence to the objectives of the work package 3.

Deliverable Description as described in the Grant Agreement	Actual Implementation
Report on SMART BEAR @ Home Enabling Components v1: This deliverable will be a report documenting the components included in D3.1. The task responsible for the development of each of these components, as identified in the description of D3.1, will also be	In this document the components that are used in the D3.1 are reported in terms of usability, interconnection and integration. Each task contributes the deliverable equally by presenting the 'smart home automation' of task 3.1 at chapter 2, the user's dashboard to support analytics and monitoring of tasks 3.2/3.3 at chapter 5, the mobile application of task 3.4 at chapter 3. A

responsible for the provision of the documentation of the relevant component in this deliverable.	detailed description of measures including data types, data fields and metadata of each personal sensor is presented in chapter 4 that is linked with the activities, interventions and monitoring of task 3.2 and local analytics of task 3.4.
The delivered components will include: at least 2 heterogeneous devices covered by the monitoring solution (KPI-1); at least 3 parameters inferred by basic analytics (KPI-4); application for screening potential participants (KPI-12); will integrate at least 2 sensing and actuation mechanisms such as those described in Table 2 (KPI-20).	KPI-1: There are 3 heterogeneous devices integrated with the HomeHub as presented in Table 3 that are capturing measurements and monitor the environment also linked with the SB@App. KPI-4: Four (4) high-level parameters that will inferred are cardiological status (with ECG), body-related measurements, physical activity status, living environment state. Health-related measurements are presented in chapter 5. KPI-12: The application of screening started by close collaboration with the pilot related WPs. In this regard initial testing performed by pilot-related partners and such initial testing is presented in appendix 6.2 KPI-20: Currently they are included 7 sensor values from HomeHub (Table 4) and at least 5 sensing and actuation mechanisms are used from Philips, Xiaomi, Withings, Garmin, openHAB in comparison with the Table 2 of the Grant Agreement.

Table 2: Fulfillment of the expected deliverable expectations

2 Smart Bear HomeHub

2.1 *The architecture and the building blocks*

The HomeHub of the Smart Bear project defines both the hardware and the software that is installed inside a building to provide and support the home automation. More specific, the HomeHub integrates and links all devices and sensors in an environment that afterwards is called 'smart'. There are many available technologies that can be installed in a house (home, building, city, vessel) under the term gateways, bridges, or hubs; for example, the Philips Hue device is called a "Hue Bridge" while the competitor device from Xiaomi is called "Aqara Gateway". In order to control all sensors and devices from a central access unit a hub is required. Once this hub supports the transfer of data values to another computational system by applying another protocol (e.x. from WiFi to ZigBee) is also called a gateway.

The Smart Bear project must use and integrate multidisciplinary devices and sensors with the simplest and more robust way. For this purpose, the openHAB[3] software platform is applied to a Raspberry PI[4] aiming to serve the technology agnostic integration and functionality through a unified way by a single management and access interface common to all external actors (users, computers, mobiles, interfaces).

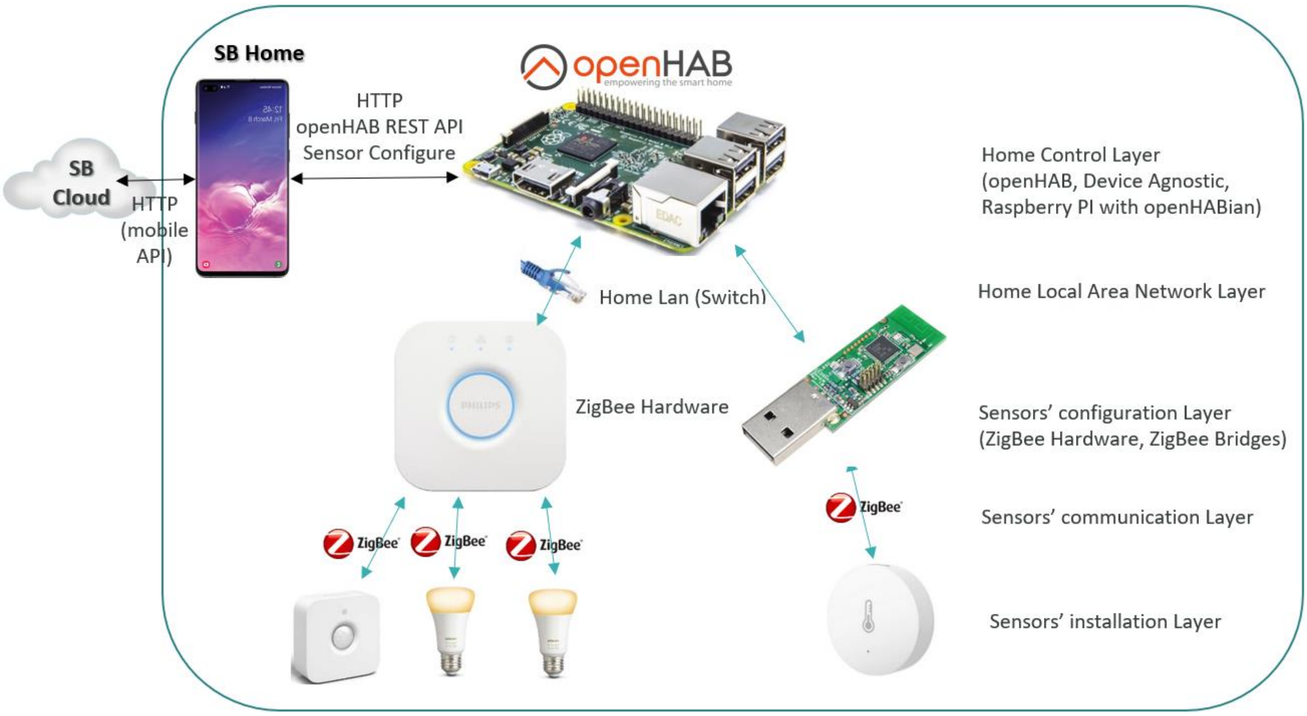


Figure 1: The Architecture of the HomeHub

Figure 1 shows the architecture of the HomeHub integration that is applied to the Smart Bear project. The architecture is based on five layers that are presented next (from bottom to top):

a) Sensors' installation Layer: The physical sensors that are installed by a technician (or the person itself) in a house. All of them are available to the stores as of-the-self sensors.

---

[3] https://www.openhab.org/

[4] https://www.raspberrypi.org/

b) Sensors' communication Layer: This is a layer hidden from the user that explicitly defines the communication protocol that a sensor is using to communicate with a hub. The communication in most cases is based on ZigBee[5] or WiFi[6] protocols. The sensors that were selected for the project are using ZigBee. In the Smart Bear it is used the Zigbee2MQTT[7] technology and platform.
c) Sensors' configuration Layer: This is the part of devices. The devices can be gateways (e.x. Philips Hue bridge, ZigBee gateways, Xiaomi Aqara Gateways). The sensors cannot operate without the appropriate hardware that fits its communication protocol. Many sensors require their corresponding bridge from the same vendor while thousands of other sensors are using the standard ZigBee technology which is characterized as the main and complete communication protocol for IoT systems.
d) The Home Local Area Network (LAN) Layer: This layer explicitly defines the communication of the Raspberry PI board with the gateways and bridges. By this way Raspberry interconnects and unifies all different devices and sensors. In the Smart Bear project this means that the home of any installation requires network communication with at least a network switch and a ZigBee usb stick that is plugged in the Raspberry PI.
e) Home Control Layer: This is the high-level layer and the actual HomeHub. The main building block is a Raspberry PI computer (any computer can host the HomeHub role) supplied with the openHAB software. The heart of the HomeHub is the openHAB technology agnostic platform that unifies all the home installation and allows access through the REST API by the Smart Bear mobile application (that in turn integrates the HomeHub with the Smart Bear cloud). This high-level layer is responsible for the following: 1) provides unified operation of all sensors, 2) exposes a unique programming interface to develop rules, actions and notifications and 3) provides a REST API[8] so any external system can communicate with any sensor to read values, invoke actions, program operations etc.

The proposed architecture provides a low-cost, single solution, technological and vendor agnostic with flexible and efficient interface for the users, the developers and any other external system. The details of the functional specifications are presented to the official openHAB documentation[9].

2.2 *Installing the Devices and the Sensors of the HomeHub automation*

There are two major parts that need to be installed, configured, and integrated to make the HomeHub functional; the hardware part and the software part are described next.

2.2.1 Hardware Components

The set of devices that were selected to compose the HomeHub automation divided in two categories as follows: a) An underlying set of devices is needed for integrating and unifying the overall automation and this is called hub, b) The set of devices that contain sensors (or standalone sensors).

[5] https://zigbeealliance.org/solution/zigbee/

[6] https://grouper.ieee.org/groups/802/11/

[7] https://www.Zigbee2MQTT.io/

[8] https://www.openhab.org/docs/configuration/restdocs.html

[9] https://www.openhab.org/docs/

2.2.1.1 HomeHub Automation

The Table 3 presents the required hardware items that need to be installed in order to provide the Smart Bear HomeHub automation. The overall infrastructure is based on a stable and well-tested solution that uses a Raspberry PI 4 computer.

No.	Device	Images (taken from vendor sites)
1	A Raspberry Pi 4 Model B computer with a micro-SD card, network interface and its own power supply.	
2	A USB connected ZigBee adapter that is connected on the Raspberry PI. The adapter allows more than 1200 sensors to be supported by the home automation hub.	
3	Philips Hue Bridge[10]	

Table 3: The hardware components of Smart Bear HomeHub.

According to Table 3 the Raspberry Pi computer will host the openHAB software thus to allow the device and vendor agnostic integration under a unified open-source umbrella that provides its own programming capabilities. This will facilitate the large-scale installation process, reduce costs, and allow interoperability for integrating with any third-party system as openHAB provides its own API.

The ZigBee adapter is connected to the Raspberry PI through the USB port and operates with the Zigbee2MQTT software platform; this adapter is required to communicate with the Aqara temperature and humidity sensor. More detailed, the adapter allows the home automation to support more than 1200 sensors related not only with automation but also with smart city and environmental interventions and planning; by this way Smart Bear is fully compatible with any IoT platform and installation. The Philips Hue bridge integrates the Hue automation and support the management interface (e.x. update firmware) of all hue sensors; in a big installation more than one bridges may be added to cover the range of the sensors (bulbs etc.) as the ZigBee radio-frequency chipset of the hue covers a range of 30 meters.

2.2.1.2 Home Devices & Sensors

A selected collection of devices and sensors that will provide measurements from the home environment is presented in Table 4 to provide the first set of stable operational installation.

[10] https://www.philips-hue.com/en-us/explore-hue/apps/bridge

No.	Device	Sensors	Images (taken from vendor sites)
1	Philips Motion Sensor	Three sensors for motion detection, temperature, and level of lighting.	
2	Philips Smart Bulbs	A sensor for lighting adjustments including color, brightness, and color temperature.	
3	Xiaomi Aqara Temperature Device	Three sensors for temperature, humidity, and pressure.	

Table 4: The automation devices and sensors

Currently there are 7 sensors totally available to the Smart Bear home automation that are providing measures and will allow personalized intervention through the mobile application of the Smart Bear. The details of all measurements are presented in chapter 4.

2.2.2 Software Components

In the heart of the home automation is the software, as also prementioned. All the software components that were applied are open source with their own documentation for both users and developers; we ‘ll avoid repetition of any information that is presented in the documentation.

No.	Software Components	Description
1	openHABian	openHABian is a self-configured Linux system setup ready to be installed in a Raspberry PI computer. It is pre-configured with openHAB, MQTT[11] server, NodeJS and many other frameworks that reduces lengthy and challenging installation tasks.
2	openHAB	The main software component that operates the HomeHub is the openHAB software platform that uses its own Java EE framework to build a modular software architecture that can manipulate any smart automation from a central center being independent from individual vendor requirements.
3	Zigbee2MQTT.io	Zigbee2MQTT is a software platform that can integrates more than 1200 sensors under a single framework. It is based on major software frameworks as NodeJS and MQTT.

Table 5: The major software components

[11] https://mqtt.org/

Table 3 presents the main components that are installed and used in the Raspberry PI computer to build the HomeHub. More specific, there are some important aspects that need to be presented:

a) The Zigbee2MQTT is a "competitor" to openHAB but openHAB is a layer above and can integrate it; in our case openHAB integrates the Zigbee2MQTT through the MQTT binding. Furthermore, openHAB provides the same access API for all ZigBee connected sensors.
b) Zigbee2MQTT requires the MQTT protocol and an MQTT broker; actually, there is no smart automation without MQTT, the standard for IoT applications.
c) To exchange data between the ZigBee adapters and the underlying software platform an MQTT broker is required and the Mosquitto framework[12] is used and installed with the openHABian.
d) Any future sensor requirement can be added though the Zigbee2MQTT framework that supports "***1217*** *devices are supported from* ***189*** *different vendors*"[13].

The Smart Bear home automation with the selected software technology can successfully linked with major market available AI technologies including Amazon Alexa[14], Google Assistant[15] and Apple Homekit[16].

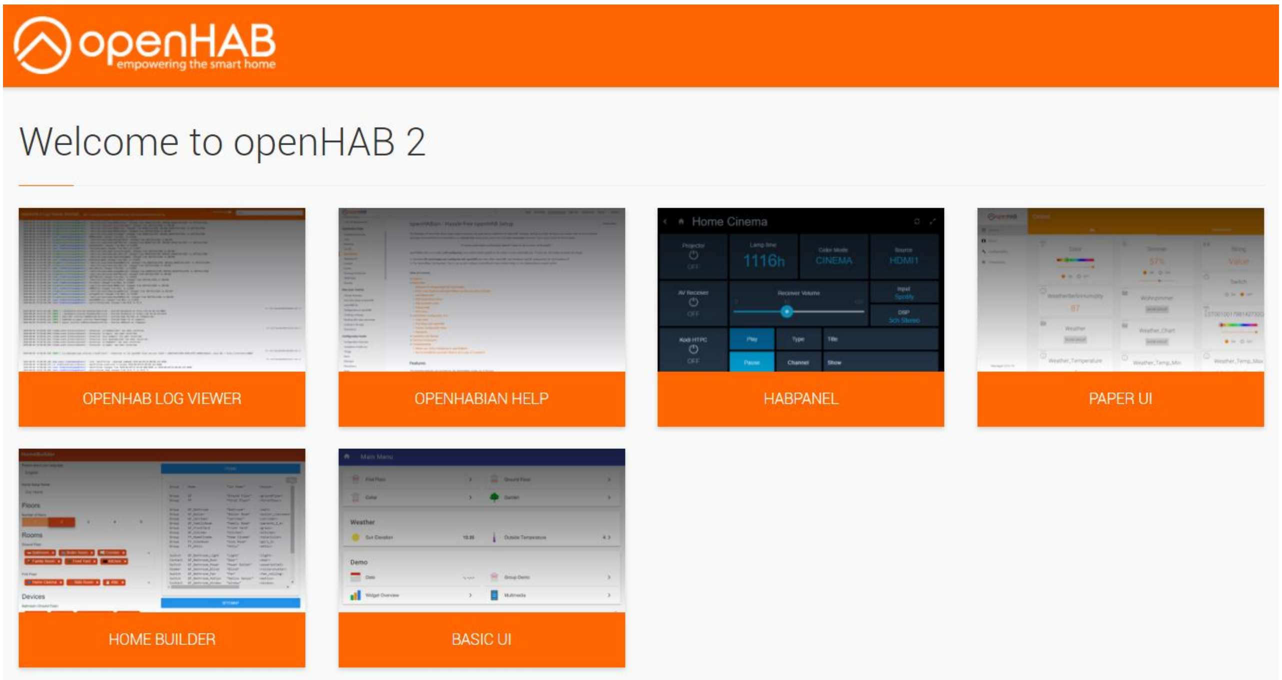


Figure 2: The openHAB main interface for configurations and management.

2.2.2.1 Interfacing the home sensors to the HomeHub

The process of interfacing the home sensors with the HomeHub starts just after a full working installation of the software components on the Raspberry PI hardware; it is assumed that the

[12] https://mosquitto.org/

[13] https://www.Zigbee2MQTT.io/information/supported_devices.html

[14] https://developer.amazon.com/en-US/alexa

[15] https://assistant.google.com/

[16] https://www.apple.com/shop/accessories/all/homekit

home environment is supported by a Local Area Network that will allow access for viewing and instructing the sensors.

The openHABian system allows accessing the platform through a zero-conf scheme using Avahi[17] which enables automatic service discovery. This means that access to the openHAB platform is made without the need of knowing the IP address or any other information. By this way, the platform is accessible with any web browser in the home LAN by the following address as also displayed in Figure 2: http://openHABianDevice.local:8080

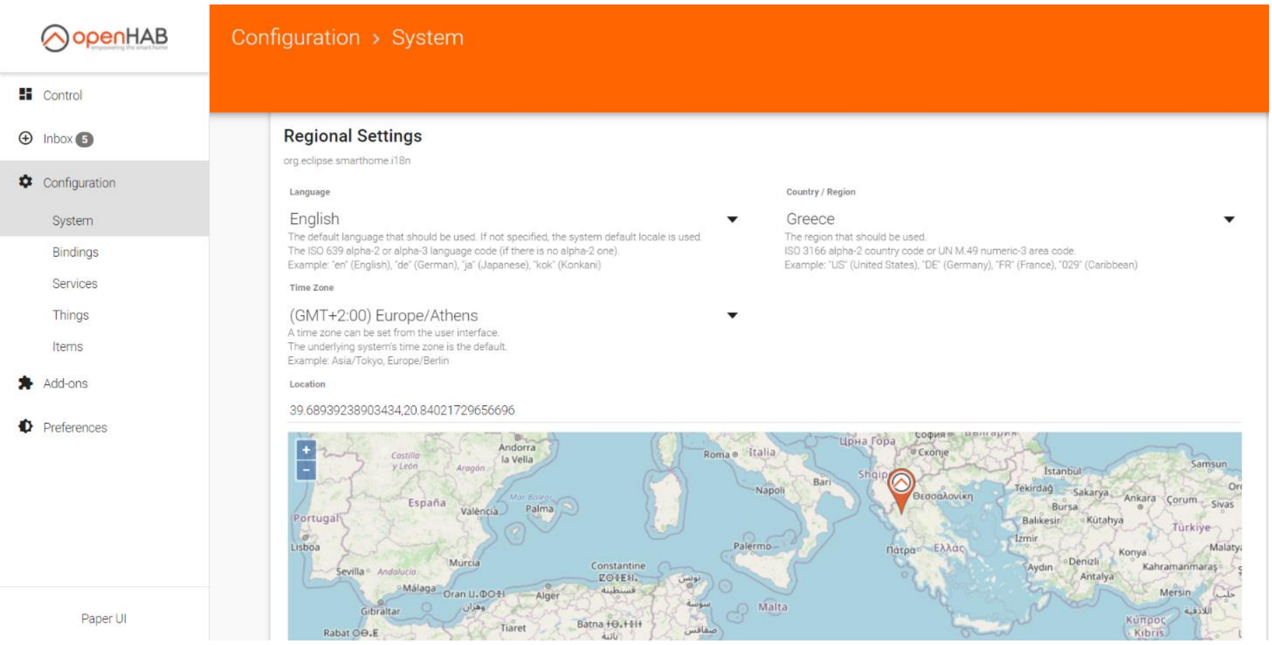


Figure 3: The system configuration interface.

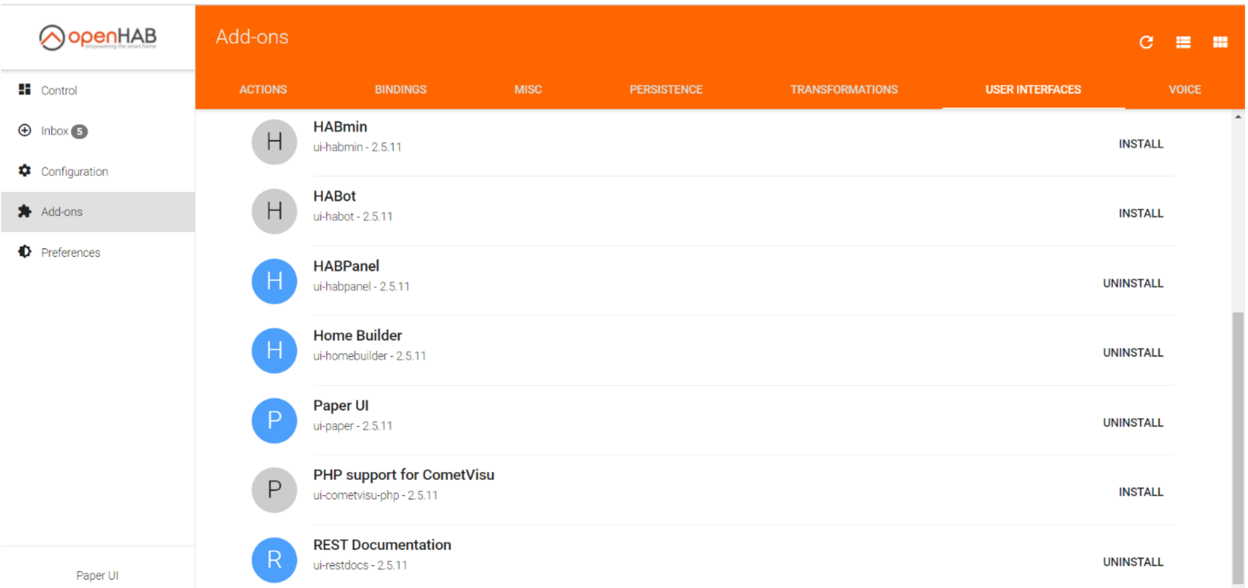


Figure 4: The openHAB add-ons.

[17] http://avahi.org/

The system configuration of openHAB (once at the beginning) is done just after the initial power-on of the platform as presented in Figure 3. The main installation step consists in adding the appropriate add-ons that are provided by openHAB community as presented in Figure 4.

The add-ons terminology of openHAB is categorized as: "actions", "bindings", "misc"," persistence", "transformations", "user interfaces" and "voice control". The applied add-ons at this phase of the Smart Bear are bindings, transformations, and user interfaces[18].

Bindings are the "drivers" that allow a sensor (hardware like Philips Hue) or service (software like a Cloud Platform or a Weather Portal) to be linked with the software platform; without it there is no way of interaction and communication.

Transformation are specific add-ons that are automatically applied during the data exchange. They are used to interpret (or translate) data from technical raw values to human readable representations[19]. In the current installation a transformation is applied to extract values from JSON structures that are send to openHAB from the Zigbee2MQTT platform.

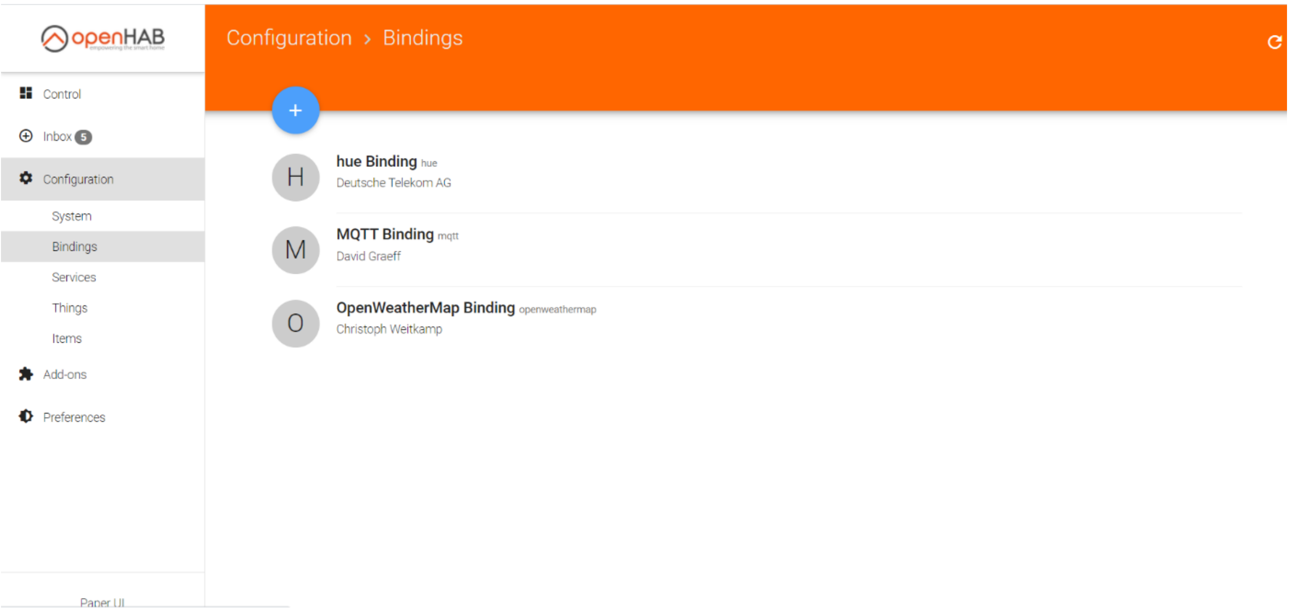


Figure 5: The installed and enabled bindings

The User Interfaces, as the name implies, are the software components that allow users and technicians (e.x. administrators) to operate and customize the HomeHub installation. Figure 2 of the initial welcome screen displays 6 different User Interfaces while Figure 4 is the screenshot of a single user interface called "paper ui".

The installed and enabled bindings are presented in Figure 5. They consist in, the Philips Hue bridge (to handle hue sensors), the MQTT broker (to handle all Zigbee2MQTT sensors) and a software weather related component that provides weather information from the OpenWeatherMap[20] service regarding the area of the installation. The list of available sensors

[18] Definition at https://www.openhab.org/docs/concepts/#things-channels-bindings-items-and-links
[19] Definition of Transformation: https://www.openhab.org/docs/configuration/transformations.html
[20] https://openweathermap.org/

and devices that can be interfaced and linked are presented in Figure 6. The available technologies provided by the MQTT binding are displayed in Figure 7.

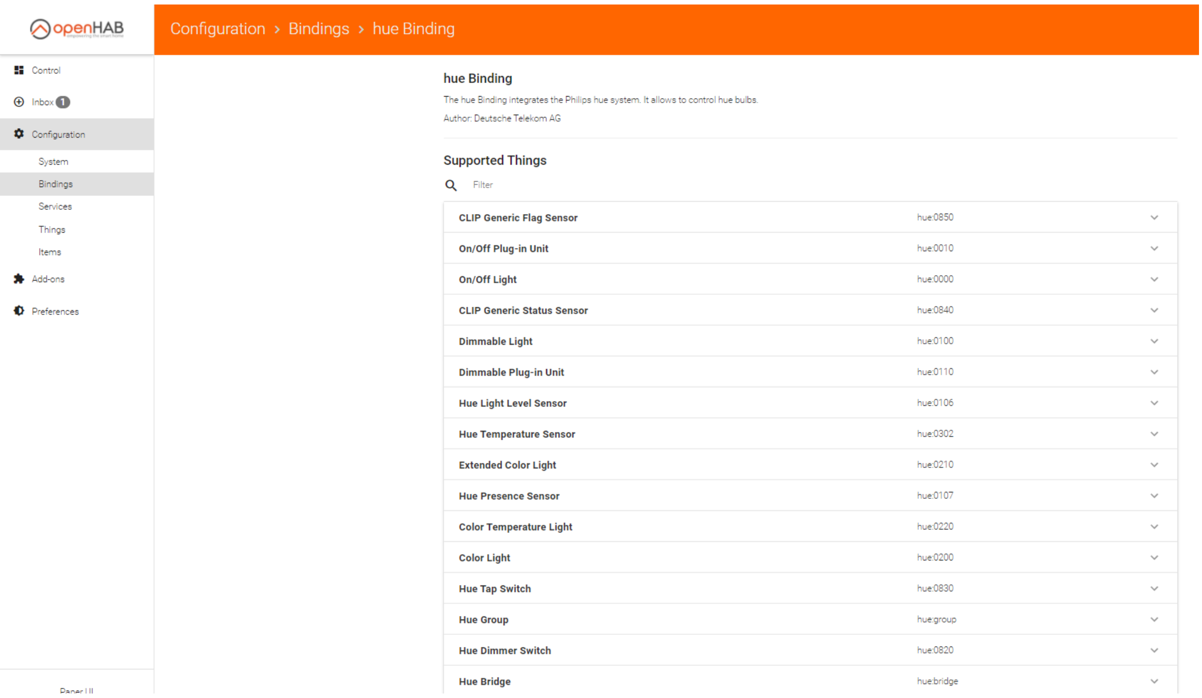


Figure 6: The supported sensors (or Things) of the Philips Hue binding.

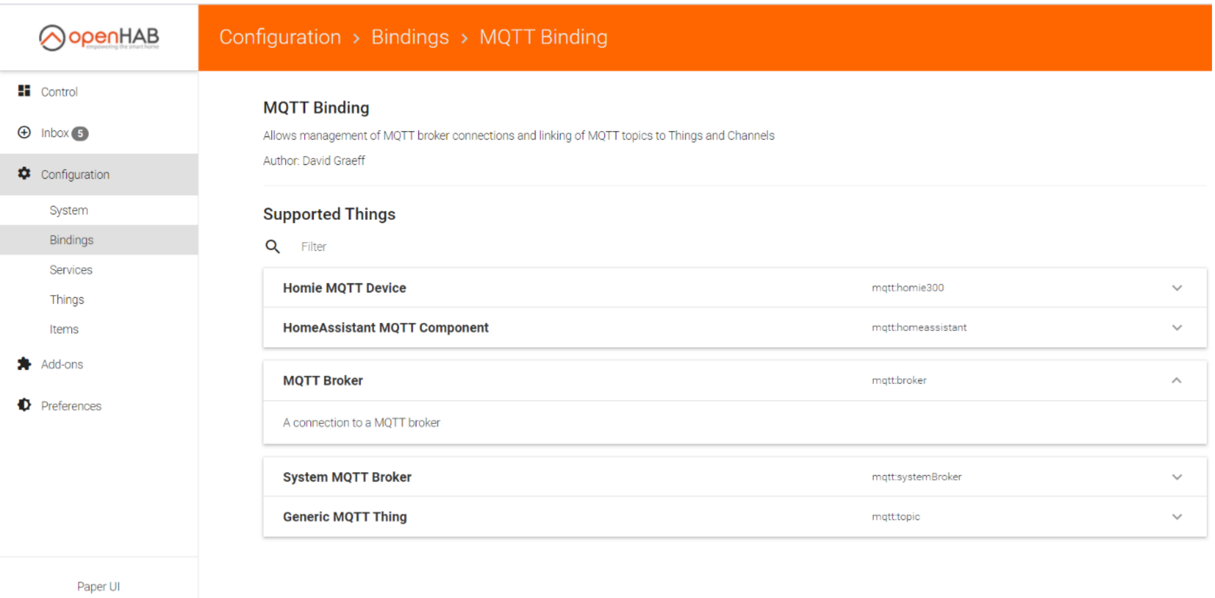


Figure 7: The installed MQTT technologies and services

The MQTT broker is the service that is used to link the topics of the Mosquitto Broker with the HomeHub.

```
openhabian@openHABianDevice: ~
[12:52:06] root@openHABianDevice:/home/openhabian# mosquitto -v
1611489130: mosquitto version 1.5.7 starting
1611489130: Using default config.
1611489130: Opening ipv4 listen socket on port 1883.
1611489130: Opening ipv6 listen socket on port 1883.
1611489131: New connection from 127.0.0.1 on port 1883.
1611489131: New client connected from 127.0.0.1 as mqttjs_732ce76e (c1, k60).
1611489131: Will message specified (7 bytes) (r1, q0).
1611489131:     zigbee2mqtt/bridge/state
1611489131: Sending CONNACK to mqttjs_732ce76e (0, 0)
1611489131: Received SUBSCRIBE from mqttjs_732ce76e
1611489131:     zigbee2mqtt/# (QoS 0)
1611489131: mqttjs_732ce76e 0 zigbee2mqtt/#
1611489131: Sending SUBACK to mqttjs_732ce76e
1611489131: Received PUBLISH from mqttjs_732ce76e (d0, q0, r0, m0, 'zigbee2mqtt/bridge/logging', ... (53 bytes))
1611489131: Sending PUBLISH to mqttjs_732ce76e (d0, q0, r0, m0, 'zigbee2mqtt/bridge/logging', ... (53 bytes))
1611489131: Received PUBLISH from mqttjs_732ce76e (d0, q0, r0, m0, 'zigbee2mqtt/bridge/logging', ... (93 bytes))
1611489131: Sending PUBLISH to mqttjs_732ce76e (d0, q0, r0, m0, 'zigbee2mqtt/bridge/logging', ... (93 bytes))
1611489131: Received PUBLISH from mqttjs_732ce76e (d0, q0, r1, m0, 'zigbee2mqtt/bridge/state', ... (6 bytes))
1611489131: Sending PUBLISH to mqttjs_732ce76e (d0, q0, r0, m0, 'zigbee2mqtt/bridge/state', ... (6 bytes))
1611489133: New connection from 127.0.0.1 on port 1883.
1611489133: New client connected from 127.0.0.1 as 94fd3af5-523c-420d-848d-07e608b02ca6 (c1, k60).
1611489133: No will message specified.
1611489133: Sending CONNACK to 94fd3af5-523c-420d-848d-07e608b02ca6 (0, 0)
1611489133: Received SUBSCRIBE from 94fd3af5-523c-420d-848d-07e608b02ca6
1611489133:     homeassistant/# (QoS 1)
1611489133: 94fd3af5-523c-420d-848d-07e608b02ca6 1 homeassistant/#
1611489133: Sending SUBACK to 94fd3af5-523c-420d-848d-07e608b02ca6
1611489133: Received SUBSCRIBE from 94fd3af5-523c-420d-848d-07e608b02ca6
1611489133:     zigbee2mqtt/aqaraht (QoS 1)
1611489133: 94fd3af5-523c-420d-848d-07e608b02ca6 1 zigbee2mqtt/aqaraht
1611489133: Sending SUBACK to 94fd3af5-523c-420d-848d-07e608b02ca6
1611489133: Received SUBSCRIBE from 94fd3af5-523c-420d-848d-07e608b02ca6
1611489133:     +/+/$homie (QoS 1)
1611489133: 94fd3af5-523c-420d-848d-07e608b02ca6 1 +/+/$homie
1611489133: Sending SUBACK to 94fd3af5-523c-420d-848d-07e608b02ca6
```

Figure 8: The Mosquitto broker in action while receiving Zigbee2MQTT messages.

Managing the MQTT broker through the binding of openHAB avoid exposing public internet address and port of the MQTT broker, this increases the access security and provides a wall of protection in front of home sensors. The MQTT broker is implemented by the Mosquitto service on the Raspberry PI computer and is explicitly used by the Zigbee2MQTT platform. The logging of the broker's execution is displayed in Figure 8.

The Zigbee adapter that is responsible of interconnecting ZigBee enabled sensors (almost all IoT sensors nowadays) is plugged in the USB port of the Raspberry PI and it is auto recognized as a serial device by the Linux OS as displayed in Figure 9.

```
[21:46:21] root@openHABianDevice:/home/openhabian# ls -la /dev/ttyA
ttyACM0  ttyAMA0
[21:46:21] root@openHABianDevice:/home/openhabian# ls -la /dev/ttyACM0
crw-rw---- 1 root dialout 166, 0 Jan 22 19:44 /dev/ttyACM0
[21:47:02] root@openHABianDevice:/home/openhabian# ls -la /dev/serial/by-
by-id/   by-path/
[21:47:02] root@openHABianDevice:/home/openhabian# ls -la /dev/serial/by-id/usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0018E2AB1B-if00
lrwxrwxrwx 1 root root 13 Jan 22 19:44 /dev/serial/by-id/usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0018E2AB1B-if00 -> ../../ttyACM0
[21:47:44] root@openHABianDevice:/home/openhabian#
```

Figure 9: The ZigBee adapter that is needed to support

The ZigBee adapter needs to be paired with the ZigBee sensors to allow the bi-directional exchange of sensors' values by following each sensor's setup instructions; this process is described in Zigbee2MQTT pairing relevant instructions[21]. The process of interfacing and

[21] https://www.zigbee2mqtt.io/getting_started/pairing_devices.html

integrating the sensors with Zigbee2MQTT is based on parameterization of the configuration file as presented in Figure 10.

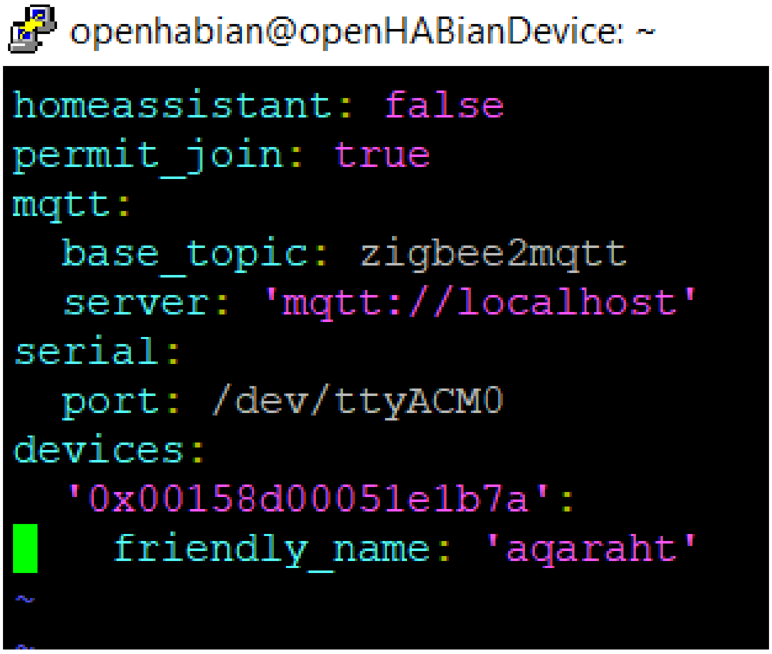


Figure 10: The parameters for interfacing ZigBee adapter with Zigbee2MQTT platform.

Through the configuration phase the topic of the MQTT broker is also defines and this is used by openHAB to access the messages (the JSON structure of the sensor values).

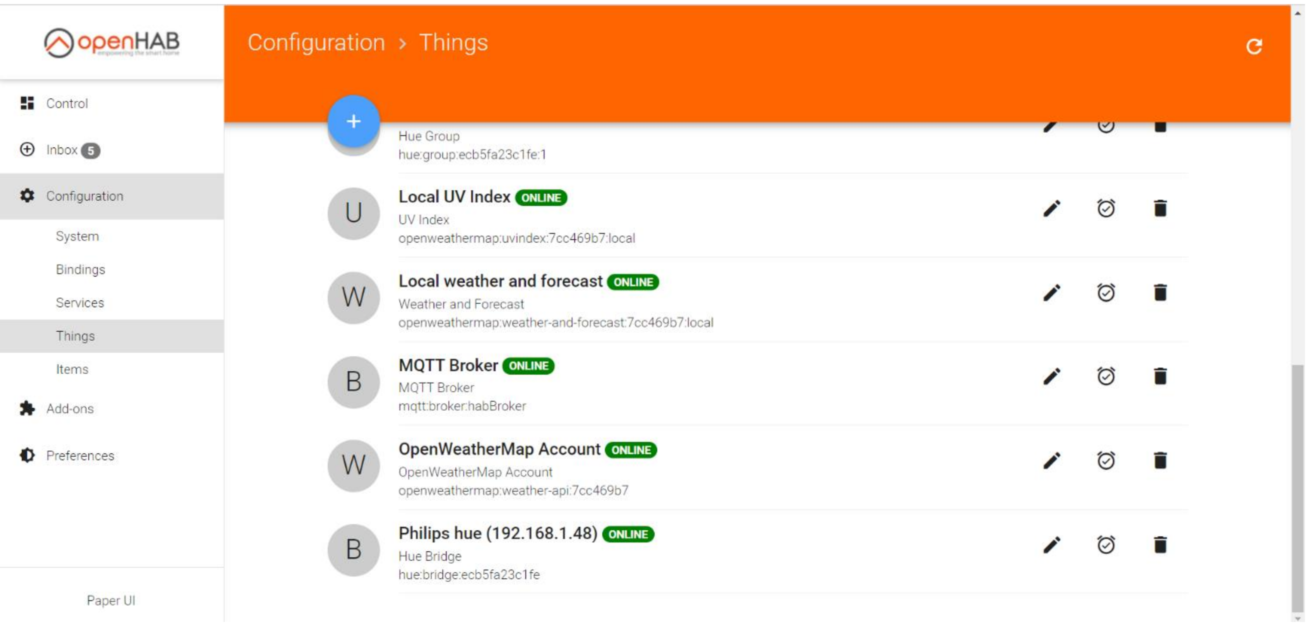


Figure 11: The configuration and the list of active things.

2.2.2.2 Automation and access schemes

In this section are presented the procedures that are providing the automation and various access schemes that means different ways of handling the HomeHub of the Smart Bear; the user specific mobile app is detailed presented in chapter 3.

The automation and smart application of HomeHub is supported by a number of programmable entities that are included in openHAB and more specific the “things” and “items”. The things are entities that are added to the platform to allow the devices and sensors to expose their values; there is no way to capture and handle any sensor without a “thing”. Things can be anything between hardware and software that needs to expose items, data values, internet services etc.

Figure 11 presents some of the things that are already available in the prototype installation. Things can be added by two ways:

a) from the “Paper UI” of openHAB as displayed in Figure 11 (e.x. Philips Hue) or
b) from the command line as displayed in Figure 12 where the Aqara temperature, pressure and humidity are added to openHAB by just adding content to a “things” file with a specific syntax provided by openHAB. In this figure it is also enabled the JSONPATH transformation service that automatically extracts the values (e.x. parses the JSON path) to feed the corresponding channel.

openhabian@openHABianDevice: ~

```
Bridge mqtt:broker:habBroker [ host="localhost", secure=false ] {
Thing topic homeaqara "HomeSensors" {
        Channels:
            Type number : temperature [ stateTopic = "zigbee2mqtt/aqaraht", transformationPattern="JSONPATH:$.temperature" ]
            Type number : humidity [ stateTopic = "zigbee2mqtt/aqaraht", transformationPattern="JSONPATH:$.humidity" ]
            Type number : pressure [ stateTopic = "zigbee2mqtt/aqaraht", transformationPattern="JSONPATH:$.pressure" ]
    }
}
```

Figure 12: The automation of any “thing” can be added by just adding prescribed openHAB content to a “things” file.

To allow things to interact with the users of the HomeHub we need to define the items of interaction and intervention with the user interfaces. This functionality is created through the creation of “items”. In the OpenHAB terminology “items” are representing the functionality that is used by any application (and the Smart Bear mobile application). The communication of an “item” with a “thing” is implemented with “channels” as presented in Figure 13.

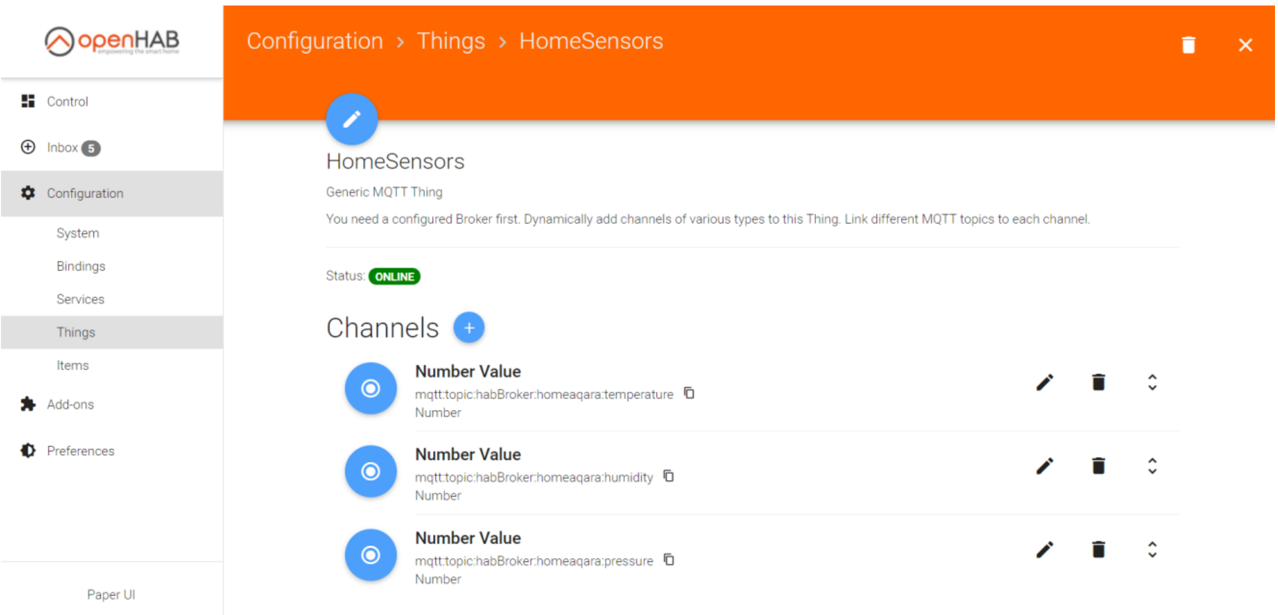


Figure 13: The channels of the Aqara sensors.

Each channel maps to an item with specific details regarding the data types, the data values and potential transformations. Figure 14 shows a subset of the user interface where the items are managed.

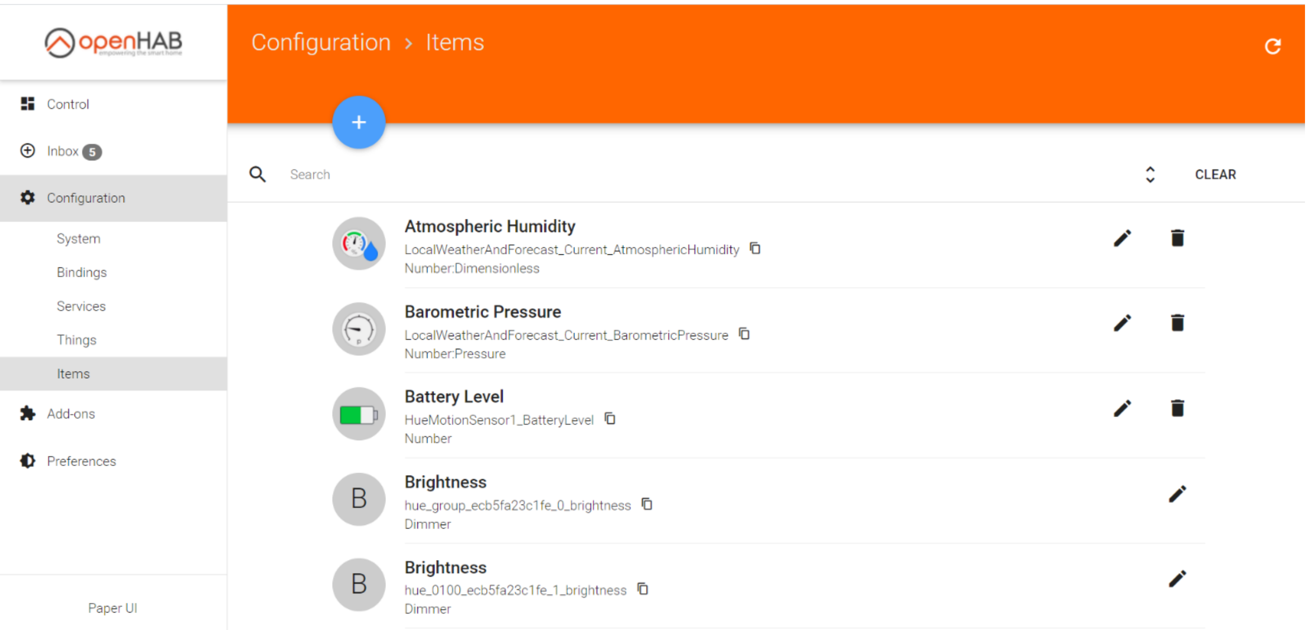


Figure 14:A subset of the current available items.

The most common way of handling the items of openHAB is through the "Paper UI" but items can be also manipulated through the command line; this is a powerful feature that allows developers and administrators to add/remove/edit items programmatically. Figure 15 presents the content of an item's file that was created to allow provision of the Aqara sensor values through the MQTT topics[22] that are fed by the Zigbee2MQTT platform that is linked by the corresponding openHAB's MQTT binding with the Mosquitto.

openhabian@openHABianDevice: ~

```
Number AQ_TEMPERATURE "Sensor Temperature [%.1f C]" {channel="mqtt:topic:habBroker:homeaqara:temperature"}
Number AQ_HUMIDITY "Sensor Humidity [%.1f %%]" {channel="mqtt:topic:habBroker:homeaqara:humidity"}
Number AQ_PRESSURE "Sensor Pressure [%.1f hPa]" {channel="mqtt:topic:habBroker:homeaqara:pressure"}
```

Figure 15: The content of the corresponding "items" file as defined according to openHAB formalism.

As presented in Figure 15 the items AQ_TEMPERATURE, AQ_PRESSURE, AQ_HUMIDITY are available to the interfaces (both user views and REST responses) of the openHAB platform. The items are described in a file with a defined structure by openHAB. Their format[23] as a data type includes additional details as the display label and the format string.

Regarding a secure access scheme as a major factor of all components of the Smart Bear home automation we are enabling an internal feature of openHAB that is called "openHAB cloud" as presented in Figure 16. This service permits accessing the openHAB platform (including all functions and services) with the connected sensors from an external network (a network that resides outside the home location). The service allows remote development and remote

[22] https://mosquitto.org/man/mqtt-7.html: "Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough."

[23] https://www.openhab.org/docs/configuration/items.html

integration from the partners implementing different parts of the project; it is considered a secure way to expose the sensor values of the home automation to the outer world.

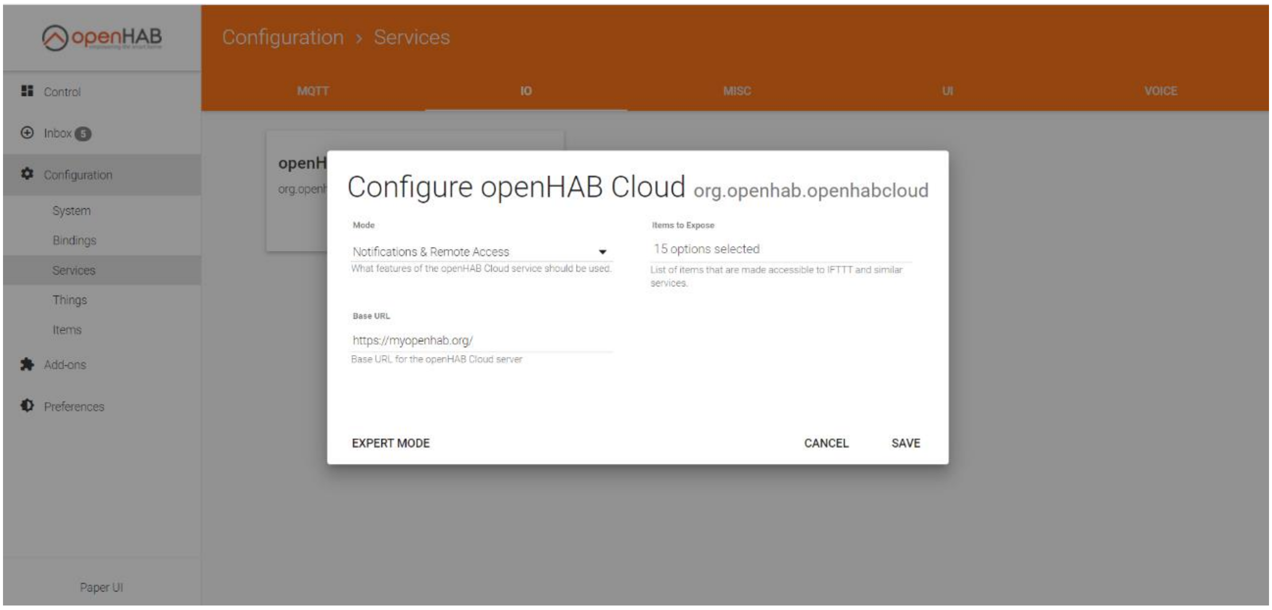


Figure 16: The openHAB cloud service for remote developer access and testing

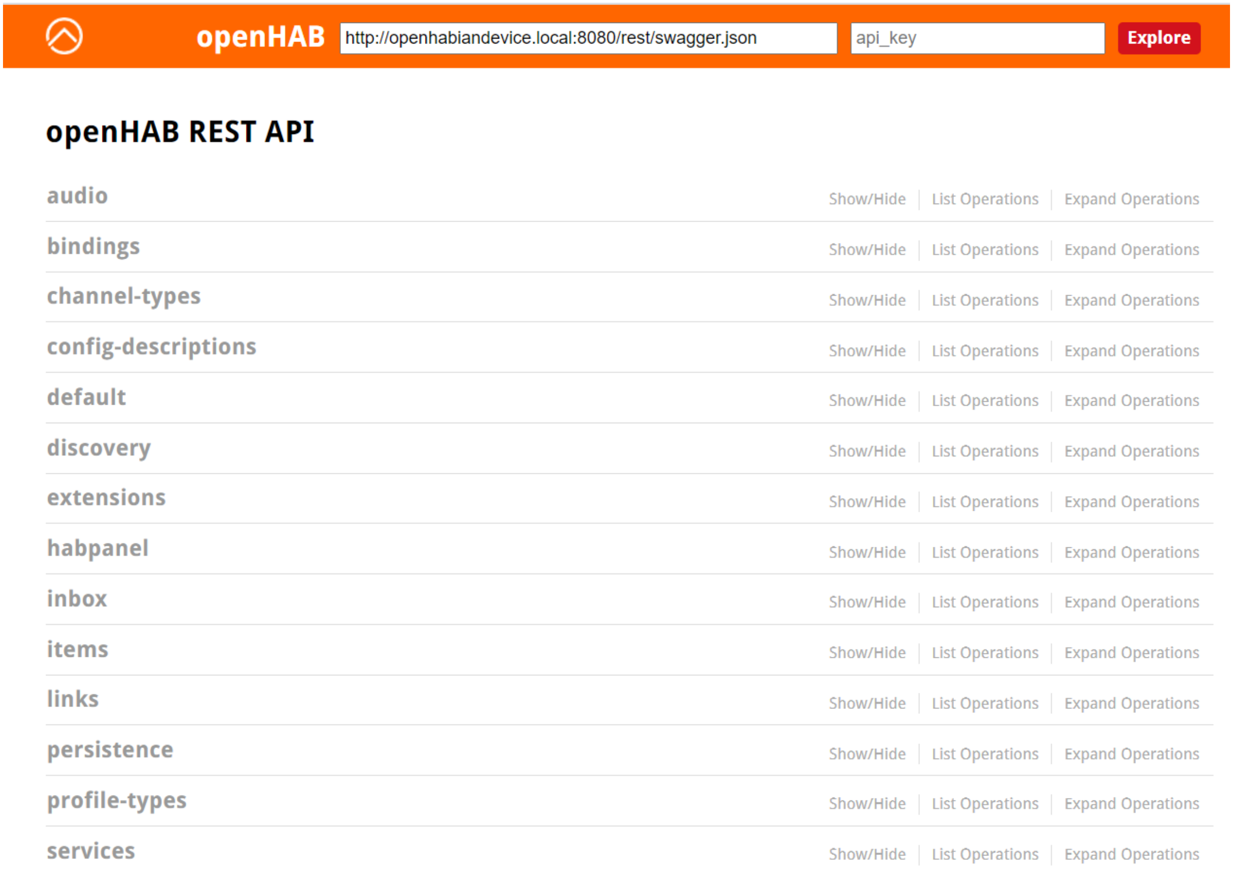


Figure 17: The REST API of the openHAB is provided at any installation.

2.2.3 The HomeHub REST API

One of the most important features provided by the openHAB is a secure (supports SSL/TSL, CORS[24], BASIC AUTH) and documented REST API. The REST API is used to allow any external system and specifically the Smart Bear mobile application to integrate with the HomeHub. The openHAB REST API can be enabled on all openHAB installation and can be reached through any web browser or a developers-oriented REST tool (e.x. Postman[25]). The detailed documentation with examples allows efficient and robust integration.

The REST API supports the GET, PUT, POST, DELETE methods of the HTTP protocol and that means that the system can create and build a new home automation through the REST API. Figure 17 presents a view of the web browser that shows the REST operations for any entity of the openHAB system.

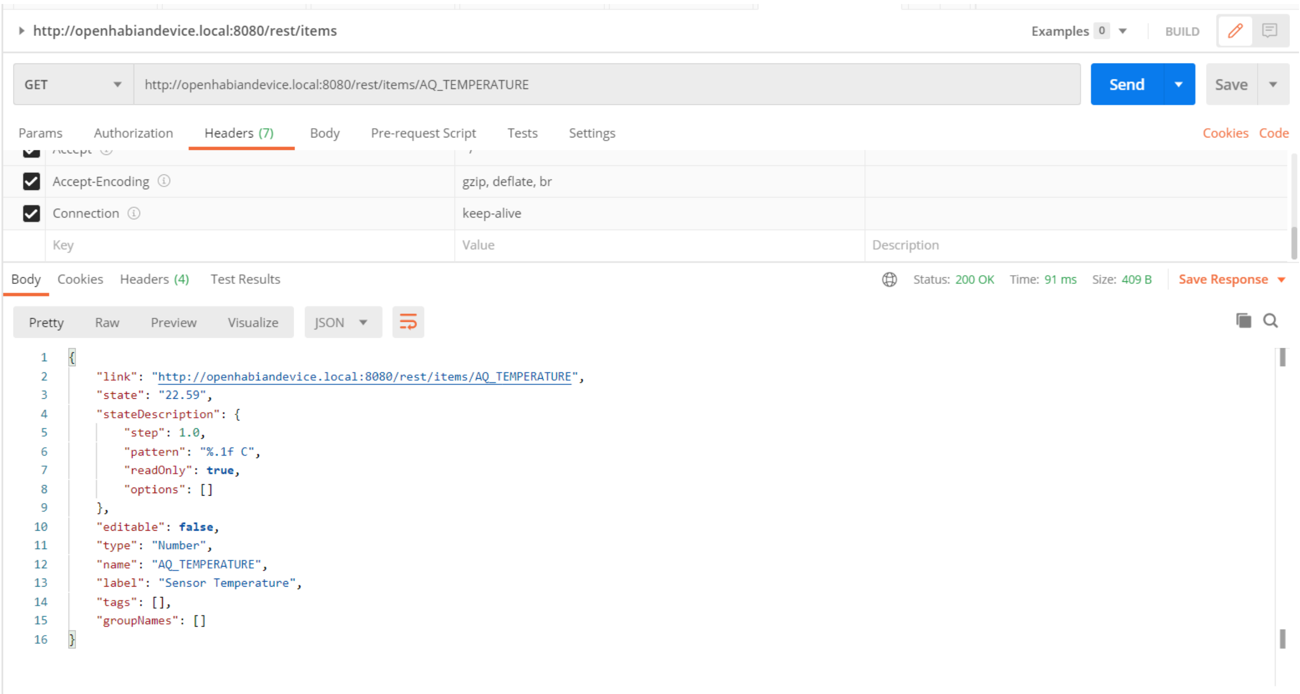


Figure 18: Accessing the Aqara AQ_TEMPERATURE item from the REST API

By applying the REST API, the administrator or an external application can create automation rules, add new items, add notifications, read persistent items. Using the Postman tool as presented in Figure 18 the REST API provides a JSON structure with the details of the corresponding AQ_TEMPERATURE item.

2.3 *Personal Devices Linked (not integrated to the HomeHub)*

The WP3 is also responsible for providing the technology and devices for the overall Smart Bear project; in this aspect the list of devices that will be used by the elderly persons are part of this chapter although they are related directly with the home automation. The detailed measurements, the data types and the fields of all devices are presented in chapter 4. Table 6 presents the selected personal "health" tracking devices. The communication with these devices

---

[24] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[25] https://www.postman.com/

will handled by their corresponding APIs' by the Smart Bear mobile application as described in chapter 3.

No.	Personal Device Selected	Description
1	Garmin Vivosport Smart Watch	Track activity and location
2	Withings Body+	Measure body related characteristics (weight, bmi, etc)
3	Withings BPM Core	Make a quick Cardio Check-up including ECG
4	Withing Smart Thermometer	Measure Body Temperature
5	IHealth Smart Air Plus	Measure SPO_2
6	Phonak Audio Marvel-50	Hearing Aids

Table 6: The list of personal "health" related devices.

3 Smart Bear Mobile App (SB@App)

The Smart Bear mobile application (SB@App) is a component aimed at providing a user-friendly graphical user interface for the participant. The application will handle the communication between the wearable devices and the SB platform while accessing the Platform utilities and functionalities. The application also provides useful data to the participant.

3.1 *Mobile App Development & Deployment Process*

3.1.1 Technology Stack

The SB@App is developed with Android Studio IDE version 4.0.1, Google's official integrated development environment. The minimum supported Android SDK version is v29.0 and it is the recommended one by the Android IDE for new applications development. Android SDK is a collection of software tools and libraries that are helpful to developers to build Android applications. In the development of the SB@App the programming language used is Kotlin. In comparison to Java, Kotlin is found to be more concise and interoperability is in its core purpose, allowing the developers writing modules in Kotlin that work flawlessly within existing Java code. Moreover, Kotlin's type system has inbuilt null safety, eliminating possible NullPointerExceptions at runtime, which is largely responsible for Android development mistakes. The SB mobile application is developed with the Model View ViewModel architecture (MVVM). The MVVM architecture model is preferred compared to other models (i.e. MVC) since it completely separates the application's UI from its logic.

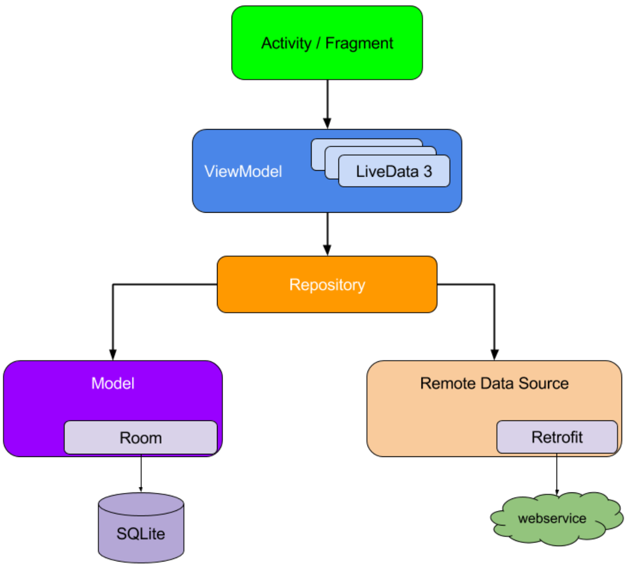


Figure 19: Smart Bear app internal architecture (https://developer.android.com/jetpack/guide)

The architecture diagram shown in figure 19consists of the following components:

- **Activity**: a screen of the Android app's user interface
- **Fragment**: a reusable portion of your app's UI
- **ViewModel**: store and manages UI-related data
- **Repository**: handle data operations
- **Room**: an object-mapping library that provides local data persistence. It provides an abstraction layer to the SQLite.
- **Retrofit**: library to access the backend

- **WebService**: the SB backend

The Activities and Fragments in the Smart Bear application display the user interface and start the requests that are passed to the ViewModel. The ViewModel then sends the requests to the Repository that determines to process it either by retrieving the requested data from the SQLite or from the WebService, based on user scenarios. When the data is retrieved the ViewModel gets updated and through subscription the Fragment gets renewed and displays the changes to the user. Frameworks and libraries used on SB@App:

- Android X, Material, Jet Brains. Core packages for navigation, design and asynchronous tasks.
- FireBase which is responsible for receiving and managing notifications
- Retrofit that builds sends and manages REST API calls
- ExoPlayer for playing and controlling videos
- MpAndroidChart that displays the charts needed

The mobile application is developed to run on mobile devices equipped with Android 10 or a more recent version.

The SB@App in order to operate relies on some backend components, hosted on the SB@Cloud. The notifications server manages the processing of notifications and/or alerts generated by the SB backend components. Also, the Garmin intermediate server is required to support the OAuth 1.0a workflows for the extraction of collected data from the corresponding vendor cloud.

The Garmin Intermediate Access Server is developed with Spring Boot 2.2.7.RELEASE and the Notifications server on 2.1.2.RELEASE. The coding language used is JAVA. Their architecture is based on the Spring Boot Flow Architecture, as it is shown in Figure 20:

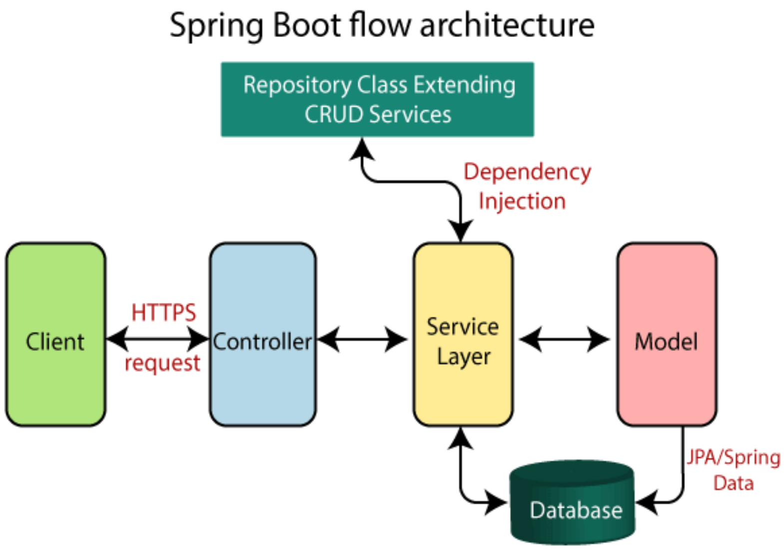


Figure 20: Intermediate and Notifications Servers Architecture

The components involved in the architecture shown are:

- **Client**: Users device or another backend service that requests and sends data to this backend
- **Controller**: A web request handler that is in the presentation layer
- **Service:** Are used to write business logic in a different layer
- **Model**: The format of the data
- **Repository**: mechanism for encapsulating storage, retrieval, and search behavior.

The requests in the SB mobile application are handled by the Controller that passes the request to the Service. The service then processes the request and if there is a need for storage, retrieval or search from the Database the Repository executes it.

3.1.2 Communication between the SB@App and the backend components

The communication between the SB mobile application and both the Garmin intermediate and the notification server is implemented using the HTTPS protocol and RESTful web services, which is a way of providing interoperability between computer systems on the Internet.

- **Hypertext Transfer Protocol (HTTPS):** This is an application protocol for distributed, collaborative, and hypermedia information systems. HTTPS is the foundation of secure data communication for the World Wide Web.
- **Representational State Transfer (REST):** It is a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

3.1.3 Automatic Releases for the SB Mobile App

The release of the SB mobile application file is in the standard APK format. The APK stands for Android Package Kit. It is the file format used by the Android OS to distribute and install mobile apps. In order to release the apk file a keystore is needed to secure the APK. Android OS requires that apps must be signed in order to be installed. Signing an app first requires creating keystores. A keystore is a storage mechanism for security certificates. A public key certificate is used to sign an APK before deployment to services like the Google Play Store. In order to secure the SB mobile application the keystore was generated through the Android Studio, as it is shown in figure 21.

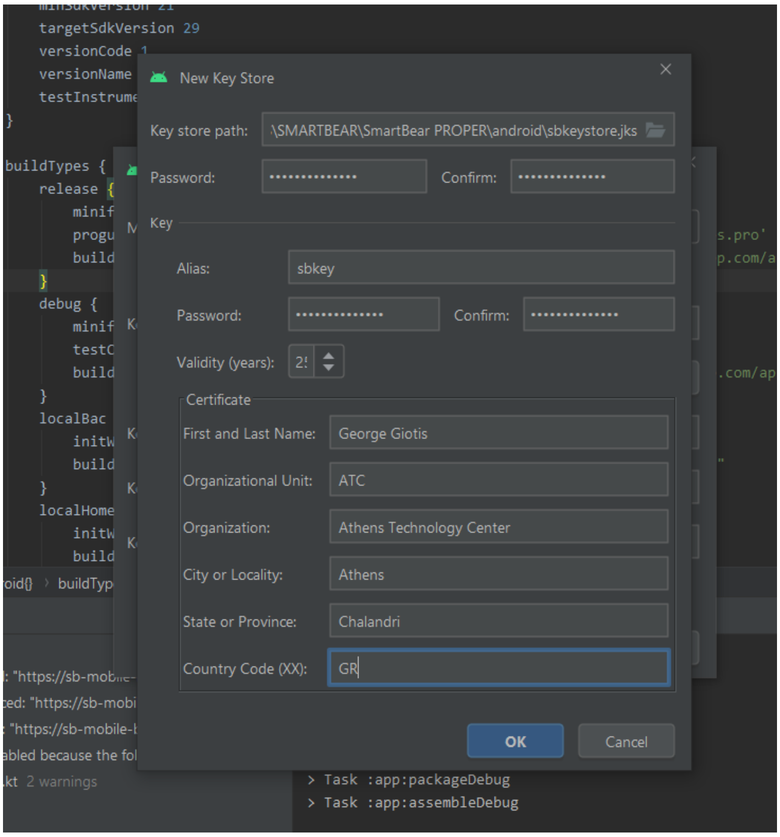


Figure 21: Smart Bears Keystore generation

Then a keystore.properties file is placed inside the development application folder, that is used on the building process of a release file.
The release of the signed apk can be done either by Android Studio or from the command line

- Android Studio IDE release

Inside the Android studio from its main toolbar menu the signed apk was released inside the projects app folder

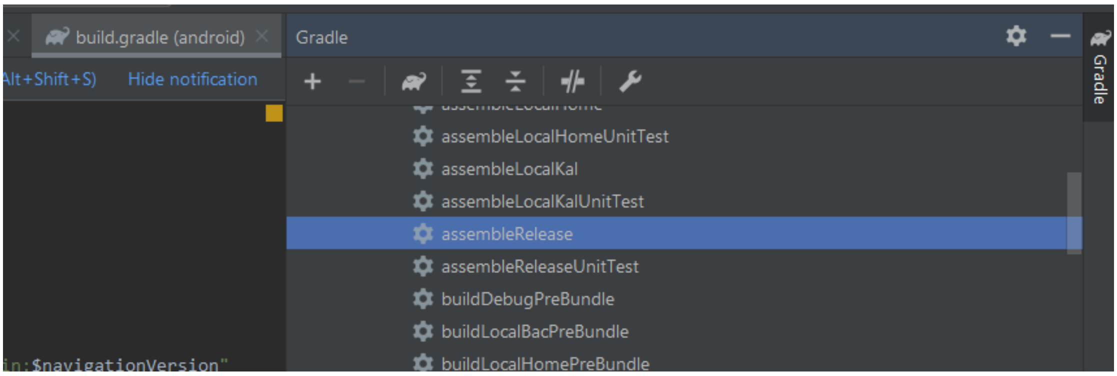


Figure 22: Release an SB APK file in Android IDE

- Command Line

From the command prompt of Windows OS, by executing the start process of the release, the signed apk was released inside the projects app folder.

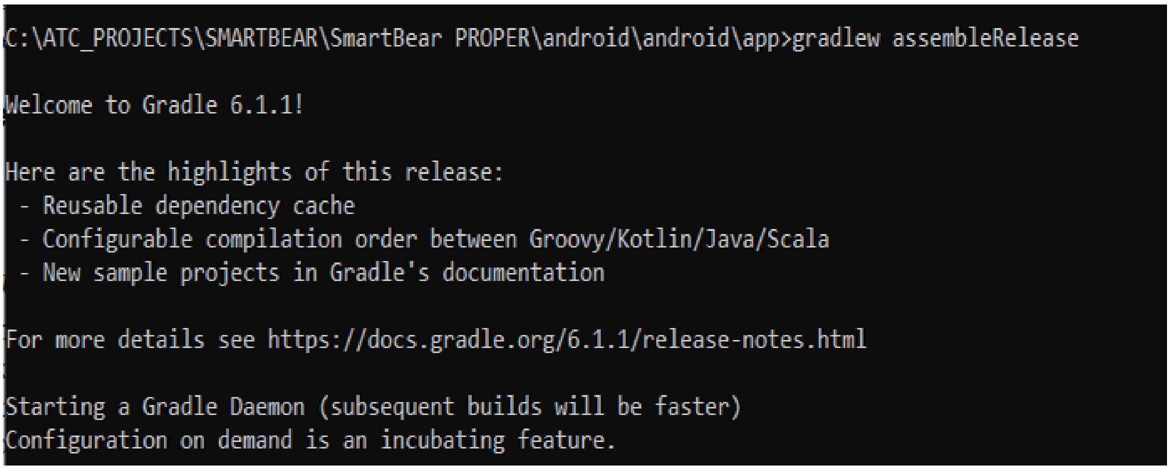


Figure 23: APK release from Android Studio

3.1.4 Mobile App Delivery

Google Play Store[26] is the official store and portal for Android apps, games and other content for Android OS devices.

Registration and Release

Through the Google Play developer console, Smart Bear mobile application is registered and created. Several setup steps where required for the registration.

[26] https://play.google.com/

Main store listings

This process required Language, App name, URL, and Graphics that will be displayed on the app page. Language declared is en-US, name of the app “Smart Bear”, and url given is http://smart-bear.eu, and graphics was given from the smart bear app project.

Content Rating

Questionnaire for content rating was submitted and reviewed by google. Target audience and content was declared. Google reported the app as PEGI 3, target audience declared is above 18.

Store Settings

The category of the Smart Bear Application was declared along with contact details. Category chosen is Health & fitness app.

After these steps, the Smart Bear mobile application is under review by Google to be released. The current version released with private access. Access is granted to test users by inserting them in a predefined emailing list.

Updates

When a new updated version of the Smart Bear mobile application is released on Google Play Store, the devices will be pre-configured with the Play Store application to auto update the Smart Bear application. Google Play Store application on the user device will handle the update procedure along with the android OS of the devices.

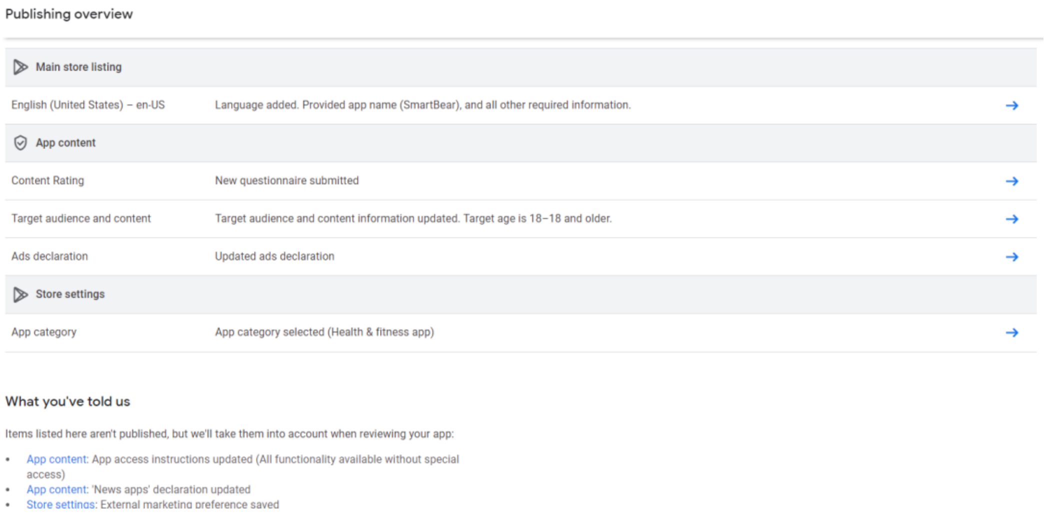


Figure 24: Google Play Store registration process for Smart Bear

3.2 *Notifications mechanism*

The notifications mechanism that is used to deliver notifications and alerts to the SB@App is based on Firebase[27] Cloud Messaging (FCM), which is a cross-platform messaging solution that allows to reliably send messages. From an architectural point of view, the components involved

[27] https://firebase.google.com/

are the SB backend component (part of the SB@Cloud) that consumes the Firebase Admin SDK, the FCM cloud backend, the Android transport layer and the SB@App with the SDK integrated, as it is shown in Figure 25.

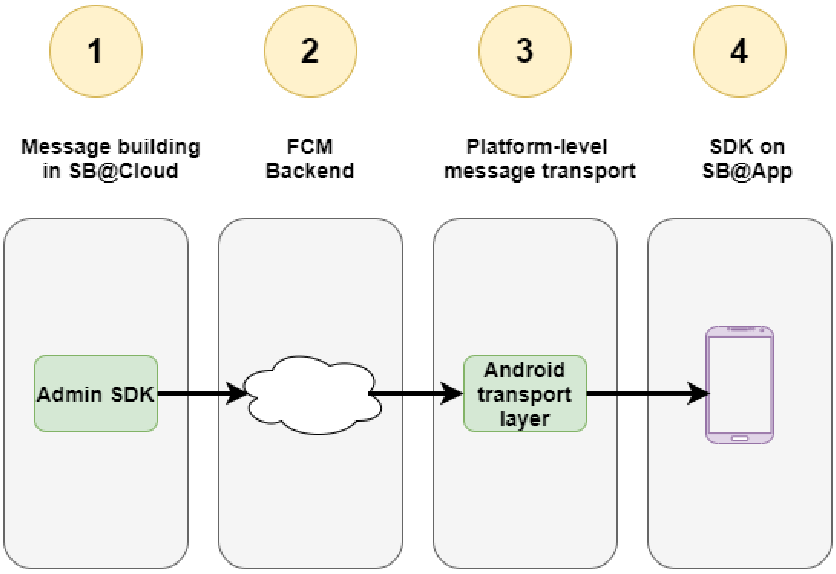


Figure 25: Cloud messaging components

The FCM solution is GDPR compliant. The Firebase users typically act as the data controller whereas Google operates as a data processor or service provider and thus the data is under the user's control. In any case, all data contained in a notification and/or alert are pseudo anonymized.

The model of message delivery used in the SB context is based on the FCM topic messaging, which is based on the publish/subscribe model. This way, messages generated in the SB backend can be sent to mobile devices that have opted into particular topics. There are two types of topic used, one that refers to the SB user's profile reflecting the medical groups the user belongs to (i.e. hearing loss group or heart disease group) and one that uniquely identifies the SB user for personal notifications/alerts. In the former case the topic name is extracted by the getProfileInfo API exposed by the SB backend, while in the latter case the topic name refers to the user's pseudo ID.

A backend component, the notifications server/component, has been implemented to support the workflow described. The notifications backend server integrates the Firebase Admin SDK (com.google.firebase.firebase-admin_6.13.0) and exposes a REST API for publishing messages to particular topics.

The API to access the notifications endpoint is shown:

Notifications Endpoint Trigger a notification in the context of the SB@App.	
Method	POST
Header	userId: User's pseudo ID.
Path	/notifications

Request body	{ "topic": "asdf1234", "message": "this is a notification", "title": "some notification title", "type": "INFO/ALERT" }

Then FCM handles routing and delivering the message reliably to the registered mobile devices. A visual representation of the notification/alerts messages is shown in figure 26:

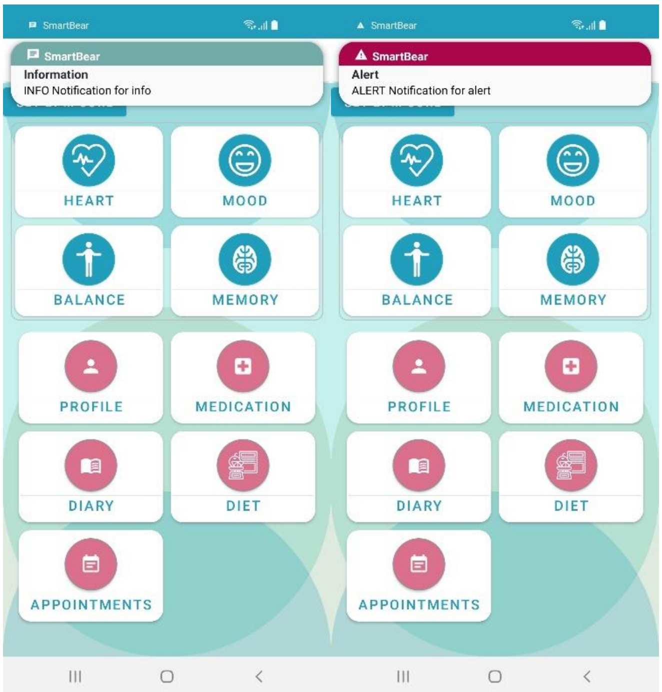


Figure 26: SB@App Incoming notifications/alerts

3.3 *Data Encryption*

The SB mobile application applies data encryption on the data collected not only while data is transmitted from the SB@App to the SB data repository and vice versa (in-transit) but also while data stays temporarily in the mobile database (at-rest). The encryption function implemented uses AES-256, a symmetric key data cipher well suited to mobile internal encryption. The SB@App creates a 256-bit cryptographic key when it is accessed for the first time and the generated key is stored in the Android Keystore, which is a storage facility for cryptographic keys. The secret key is stored in a protected format to prevent unauthorized access. Moreover, since the key is created from the SB@App and stored in the internal keyStore, the access to the key is restricted to the SB@App itself.

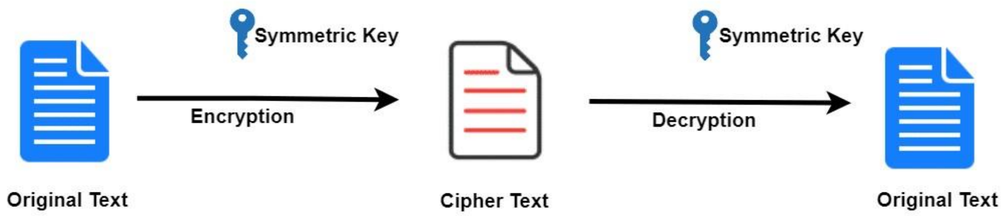


Figure 27: SB@App internal database encryption & decryption flow

The data collected from the connected devices are first encrypted using the secret key and then are stored in the internal database. Periodically the SB@App checks the internal database for pending data packets, it decrypts them using the secret key and sends them to the SB data repository.

3.4 *Integration with SB Devices*

In the current release the SB@App provides integration with the following devices/components: the home hub, the smart watch and the blood pressure recorder.

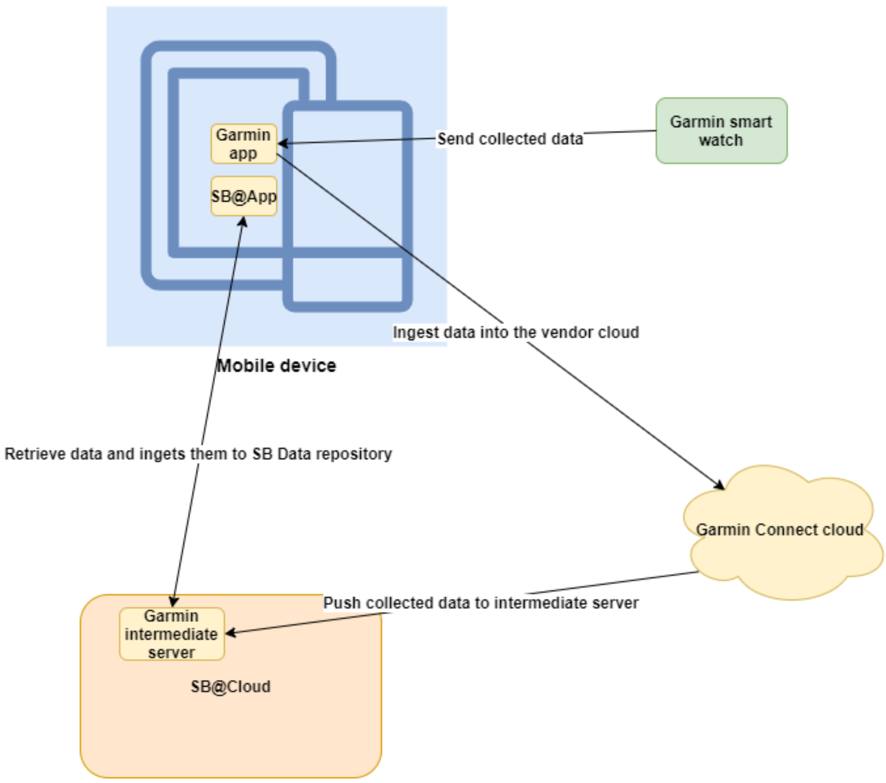


Figure 28: Data ingestion workflow for smart watch device

The solution tested as the SB home hub component is based on OpenHAB while the one tested as the SB smart watch is based on Garmin Vivosport and the one tested as the SB blood pressure recorder is based on Withings BPM Core.

REST API integration is used for the communication with the home hub component. Web API integration is used to retrieve the data from the corresponding vendor clouds of Garmin and Withings. The wearable devices transmit data (via Bluetooth connections) to the vendor applications running in the mobile device. The vendor apps act as a gateway for the collected data so that they eventually reach the vendor cloud. In the case of the Garmin smart watch device an intermediate server is required where the collected data are synced at preconfigured intervals by the vendor cloud service. The Health API offered by the Garmin grants access to the data of interest. The Web API offered by the Withings is the Data API and allows to directly retrieve data from the Withings vendor cloud. The SB@App ingests the collected data to the SB data repository if an internet connection is available. The data ingestion workflow is shown in figure 28.

Local Network communication with HomeHub: SB app uses the Local Network of the user's residence, to start the communication with the HomeHub.

3.4.1 SB@HomeHub

SB@App communicates with the home hub component that acts as a gateway the various sensors connected to it. The SB@App retrieves the home sensor values and ingests them to the SB data repository, encrypts them, temporarily stores them and displays them to their respective screens. All this process is proceeded through the openHAB REST API presented in Figure 17.

3.4.1.1 HomeHub communication

The SB@App communicates with the home hub component through the REST API that it exposes. A preconfigured REST endpoint is scanned to detect whether the home hub connection is available. The predefined home hub endpoint URL is http://openhabiandevice.local:8080/.

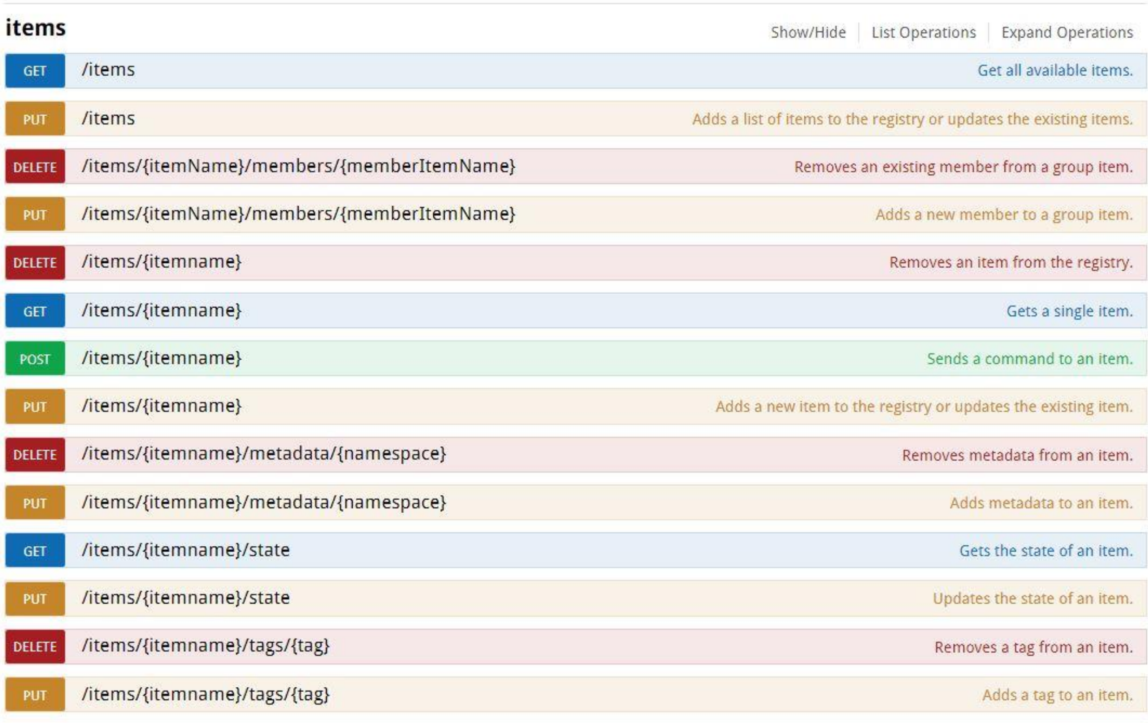


Figure 29: HomeHub REST API List for items, in the current version Smart Bear uses the first GET /items request

The SB@App sends a GET request to the items' endpoint of the HomeHub to verify its presence. Upon successful retrieving of the sensors installed, along with their names and values the active status of the HomeHub is considered verified.

3.4.1.2 Integration to SB@HomeSensors - REST API integration

After verifying the connection to the home hub component the The SB@App communicates periodically with the home hub controller to retrieve all the home sensor data by sending REST requests per home sensor item.

HomeHub Items Request:
curl --location --request GET 'http://cloud.kalatzis.eu:8080/rest/items' \

Response:


```
[
  {
    "link": "http://cloud.kalatzis.eu:8080/rest/items/HueMotionSensor1_LastUpdated",
    "state": "2021-01-22T10:48:36.000+0100",
    "stateDescription": {
      "pattern": "%1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS",
      "readOnly": true,
      "options": []
    },
    "editable": true,
    "type": "DateTime",
    "name": "HueMotionSensor1_LastUpdated",
    "label": "Last Updated",
    "category": "Time",
    "tags": [],
    "groupNames": []
  },
  ……………………………….
}
]
```


3.4.1.3 Usage of HomeSensors data

Each sensor in the HomeHub interface is named as “items”. Every item has a name, a type and a state that the Smart Bear application displays on its respective screens. Moreover, these values are encrypted with AES-256 and stored temporarily in the application database until they are sent with a REST call to the SB data repository and then deleted from the local database.

3.4.1.4 Screens of HomeSensors data

The data are displayed in the screens of light levels, comfort and movement. The Light Levels screen displays the light levels of each room inside the participants housing. In the Comfort screen, the temperature and the moisture from sensor data is displayed. While in the Movement displays the latest change of the door sensor in the user residence.

Figure 30: Light levels, comfort and movement screens

3.4.2 SB@SmartWatch (Garmin)

The Garmin vívosport wearable tracks the user's daily summaries and sends the data to the corresponding vendor app that is installed and runs in the mobile device through Bluetooth connection. Then the vendor app syncs those data to the vendor cloud (Garmin Connect Platform) whenever an internet connection is available. The SB@App with the usage of the HEALTH API[28] provided by Garmin retrieves this data.

3.4.2.1 HEALTH API integration

In order to integrate the Garmin Connect HEALTH API with the Smart Bear mobile application the following steps are required.

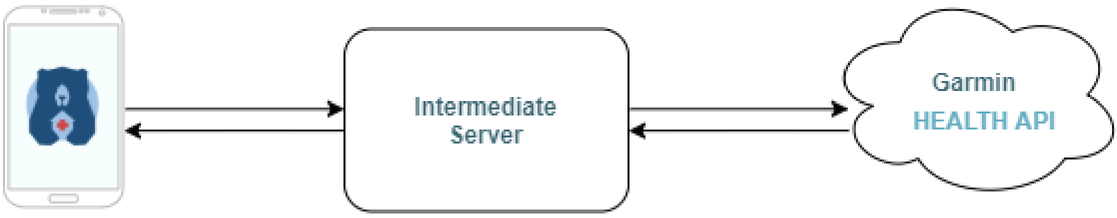


Figure 31:Process of retrieving Garmins wearable data

1. Install and Configure intermediate server

Set up a server application that functions as the intermediate connection between the Garmin HEALTH API and the Smart Bear mobile application. The server is able to authorize, store

---

[28] https://developer.garmin.com/gc-developer-program/health-api/

temporarily and receive and send requests to the Garmin HEALTH API and respond to Smart Bear application requests.

2. Installation of Garmin Connect Mobile Application:
On the mobile device Garmin Connect[29] application was installed via playstore.

3. Set up Garmin Connect Mobile Application:
Garmin connects display an initial setup with forms that require user sex, age, height and weight. As well as a screen that pairs the wearable Garmin Vivosport with the mobile device through the Garmin Connect application. After the initial setup of the Garmin Connect Application a sign in process is required into the Garmin Connect with an email and a password for storing user data into the Garmin Connect Platform.

4. Create and register an Application inside Garmin Connect:
Authorization to receive user's data is granted through a registered application that is created inside the Garmin website, named "Smart Bear". Smart Bear Garmin application is a security key to request from Garmin Connect Platform.

5. Authorization Process for each user
The Authorization process is based on oAuth 1.0a.

Step 5.1: Acquire Unauthorized Request Token
At the initial process of setting up of Smart Bear mobile application, Smart Bear mobile application sends a request for the Garmin Request Token. When the Request Token is retrieved it is saved on the storage of the mobile device.

Request:
```
curl --location --request POST 'https://connectapi.garmin.com/oauth-service/oauth/request_token' \
--header 'Authorization: OAuth oauth_nonce="xyzxyzxyzxyz",
oauth_signature="xyzxyzxyzxyzxxx",
oauth_consumer_key="xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz",
oauth_timestamp="xyzxyzxyzxyz", oauth_signature_method="xyzxyzxyzxyz",
oauth_version="1.0"' \
--header 'Cookie: __cfduid=xyzxyzxyzxyz'
```

Response:
```
oauth_token=xyzxyzxyzxyzxyz&oauth_token_secret=xyzxyzxyzxyzxyz
```

Step 5.2: Authorization of the Request Token

[29] https://play.google.com/store/apps/details?id=com.garmin.android.apps.connectmobile&hl=el&gl=US

With the request token retrieved in step 1 the SB@App loads a pop-up web browser that redirects the user to the Garmin sign-in URL. The URL contains an authorization form with email and password fields that the user must fill in. On the next steps of this procedure, there is a choosing of which category of data each user accepts to share with the SB@App and an agreement of terms that is required to be agreed.

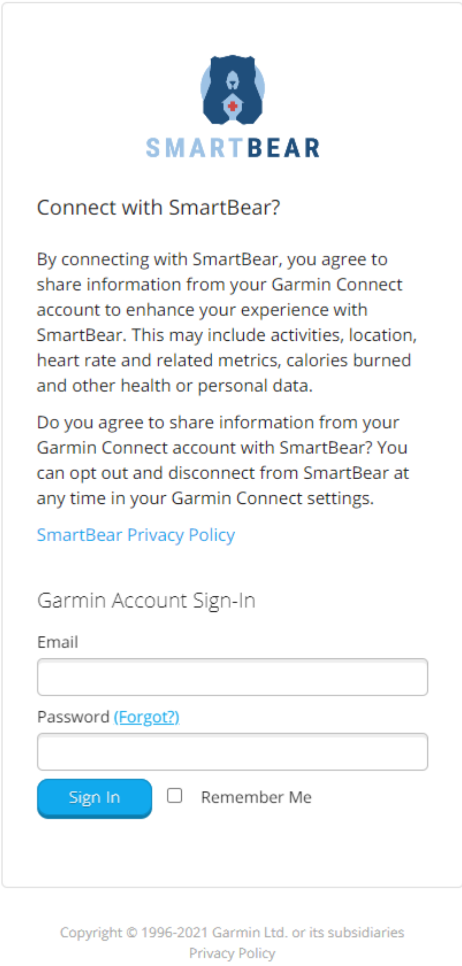


Figure 32: Sign in process for granting access to user data

Step 5.3: Acquire User Access Token

After the user authorized the data access from the Smart Bear Garmin application, the user is redirected back to the partner's site with a verification code. A new request is auto generated and sent that receives the User Access Token and User Access Token Secret.

Request:

curl --location --request POST 'https://connectapi.garmin.com/xxyz/xxxyz' \
--header 'Authorization: OAuth oauth_nonce="xyzxyzxyzxyzxyz",
oauth_signature="xyzxyzxyzxyzxyzxyzxyzxyzxyz",
oauth_consumer_key="xxyzxyzxyzxyzxxxxxxxxxxxx", oauth_token="xxxxxxyzxyzxxxxxx",

oauth_timestamp="xxyzxyzxxx", oauth_verifier="xxyzxyzxxx",
oauth_signature_method=xxxyzxyzxyzxxxx, oauth_version=xxxyzx \
--header 'Cookie: __cfduid=xxxxyzxyzxxx'
--header 'Cookie: __cfduid=xxyzxxxxyz'

Response:
oauth_token=xyzxyzxyzxyzxyz&oauth_token_secret=xyzxyzxyzxyzxyz

The User Access Token and User Access Token Secret for this procedure is stored in the Smart Bear Mobile Application shared preferences storage. If no Access Token is saved in the storage this procedure will be re attempted from step 5.1.

6. Retrieving data for each user:
The Garmin wearable device syncs data with the vendor application and then the vendor application transmits these data to the Garmin cloud. The SB@App must trigger the intermediate server to retrieve these data. The Garmin cloud API transmits at fixed intervals (~ 8 hours) the collected data to the SB intermediate server.

3.4.2.2 Data collected

The Health API provides a series of data from the user measurements that refer to daily and the sleep summary data. The Sb@App stores the following well-being properties:

- Summary Id
- Summary starting date
- Summary last hour
- Average heart rate

Regarding the summary sleep data the SB@APP stores the following properties:

- Summary Id
- duration in seconds
- starting time in seconds

Daily summary request:
curl --location --request GET 'https://garmin-backend.herokuapp.com/push/dailyPushedSummary'

Response:
{
  "epochs": **null**,
  "dailies": [
    {
      "userAccessToken": "**xxxxxxxyyyyyyxxxxxx**",
      "summaryId": "x37b622c-6009f960-a140-6",

"activityType": "WALKING",
"activeKilocalories": 839,
"bmrKilocalories": 1122,
"steps": 4606,
"distanceInMeters": 3814.0,
"durationInSeconds": 41280,
"activeTimeInSeconds": 6325,
"startTimeInSeconds": 1611266400,
"startTimeOffsetInSeconds": 7200,
"moderateIntensityDurationInSeconds": 120,
"vigorousIntensityDurationInSeconds": 1440,
"floorsClimbed": 0,
"minHeartRateInBeatsPerMinute": 55,
"maxHeartRateInBeatsPerMinute": 159,
"averageHeartRateInBeatsPerMinute": 87,
"restingHeartRateInBeatsPerMinute": 72,
"timeOffsetHeartRateSamples": {
………………
}
}
…………
]
}

From the daily summary data request, Smart Bear mobile application displays the average heart rate beats per minute.

Daily sleep request:
curl --location --request GET 'https://garmin-backend.herokuapp.com/push/sleepPushedSummary'

Response:

{
"epochs": null,
"dailies": null,
"sleeps": [
{
"userAccessToken": "xxxxxxxyyyyyyxxxxxx",
"summaryId": "x37b622c-600a1634-50dc",
"durationInSeconds": 20700,
"startTimeInSeconds": 1611273780,

"startTimeOffsetInSeconds": 7200,
"deepSleepDurationInSeconds": 2220,
"lightSleepDurationInSeconds": 13260,
"awakeDurationInSeconds": 840,
"sleepLevelsMap": {
…………………………………..
}
………………………………………
}
]
}

While from the sleep summary data request, Smart Bear mobile application displays the sleep duration of the previous day.

3.4.2.3 Screens of SmartWatch data measurements

Heart Rate: The Heart Rate screen provides the latest sync measurement screen from the daily summary data of the heart rate and the history of measurements. The user can see his latest heart rate measurement along with an info message that indicates a conclusion of the result based on generic health rules. Respectively on the history screen there are all the measurements.

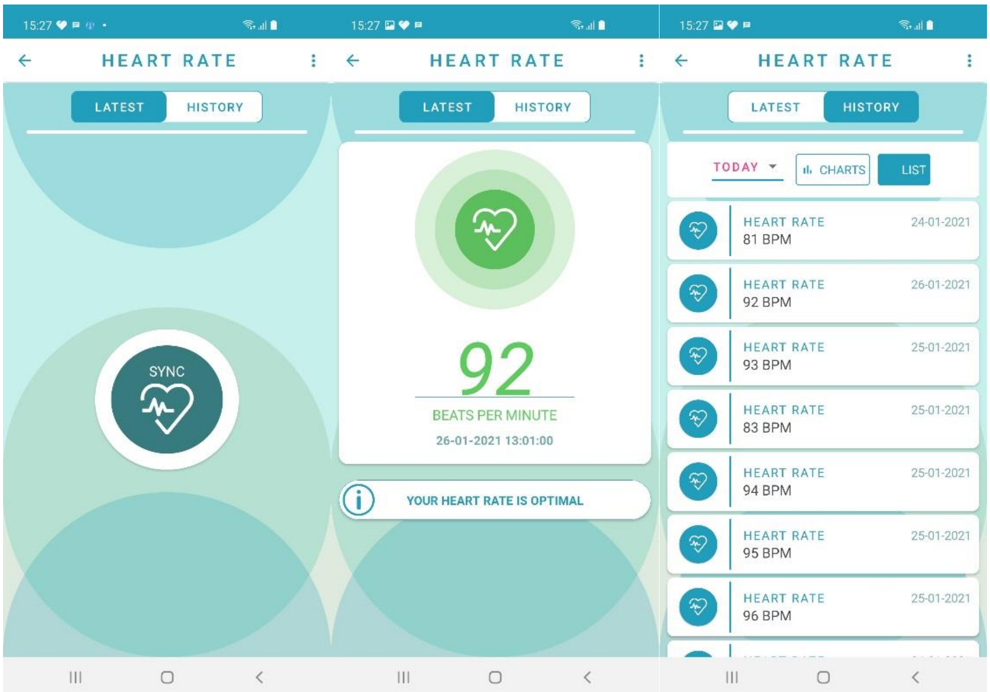


Figure 33: Heart Rate data display

Sleep Statistics: The Sleep statistics screen provides the latest sync measurement screen from the daily sleep summary data. The user can see his latest sleep duration period of time along with an info message that indicates a conclusion of the result based on generic health rules.

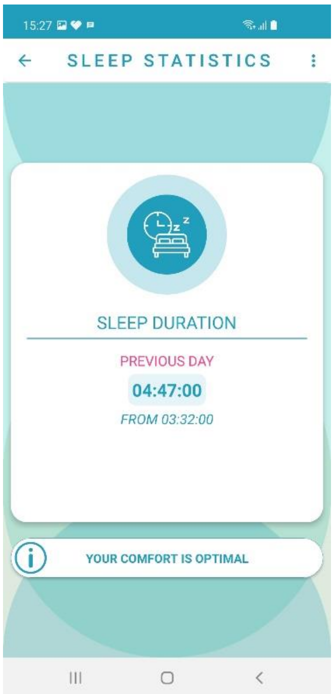


Figure 34: Sleep Statistics

3.4.3 SB@BloodPressure (Withings)

The blood pressure recorder retrieves the user's vital signs about systolic and diastolic pressure and heart rate data, and syncs them to vendor application. The application sends the data to the vendor cloud where they are accessible and stored. The SB@App is able to retrieve those data by using the vendor data API.

3.4.3.1 WEB Api integration

In order to integrate the Withings DATA API[30] into the SB@App the following steps are required.

1. Installation of Vendor Application

On the mobile device Withings Health Mate application is installed from Google Play Store.

2. Setup Withings Health Mate Application

Withings Health Mate Application displays an initial setup with forms that require user sex, age, height and weight then a screen that pairs the wearable Withings BPM Core with the mobile device.

[30] https://developer.withings.com/oauth2/#section/Data-API

After the initial setup of the Withings Health Mate Application a sign in process is required into the Withings Health Mate application[31] with an email and a password for storing user data into the Withings platform.

4. Create and register an Application inside Withings:
Authorization to receive user's data is granted through a registered application that is created inside the Withings website, named "Smart Bear". Smart Bear Withings application is an authorized gateway to retrieve data from Withings Platform.

5. Authorization Process for each user
The Authorization process is based on oAuth 2

Step 5.1: Acquire Authorization Code
At the initial process of setting up of Smart Bear mobile application, Smart Bear mobile application will redirect the user to a specific url of Withings where he must sign in and grant required permissions for the Withings Smart Bear application. After this step an authorization code will be generated from Withings.

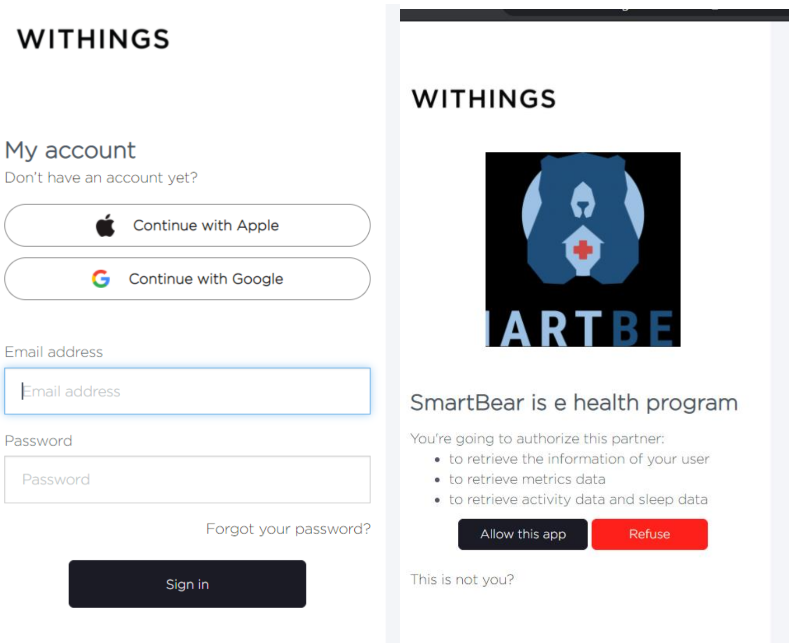


Figure 35:Withings user data access procedure

Step 5.2: Acquire Access Token and Refresh Token of each user
Smart Bear mobile application will retrieve the authorization code from the previous step and request an Access Token and Refresh Token from Withings API.

[31] https://play.google.com/store/apps/details?id=com.withings.wiscale2&hl=el&gl=US

Access and Refresh token request:

```
curl --location --request POST 'https://wbsapi.withings.net/v2/oauth2' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'action=requesttoken' \
--data-urlencode 'client_id=xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz' \
--data-urlencode 'client_secretxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz' \
--data-urlencode 'grant_type=authorization_code' \
--data-urlencode 'code=xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz' \
--data-urlencode 'redirect_uri=https://app.smart-bear.eu/callback'
```

Response:


```
{
  "status": 0,
  "body": {
    "userid": "xyzxyzxzyxzyxyzxyzyxz",
    "access_token": "xyzxyzxzyxzyxyzxyzyxz",
    "refresh_token": "xyzxyzxzyxzyxyzxyzyxz",
    "scope": "user.info,user.metrics,user.activity",
    "expires_in": 10800,
    "token_type": "Bearer"
  }
}
```


6. Retrieving data for each user:

The SB@App with the usage of the access and refresh token can retrieve the data from the Withings cloud, by accessing the Data API. Whenever the token is expired a new one is generated by using the refresh token.

3.4.3.2 Data Collected

DATA API, provides a series of data from the user like blood pressure, heart rate etc. From these data, Smart Bear mobile application displays the user's blood pressure, systolic and diastolic from a daily basis along with the date collected. The Sb@App stores the following properties:

- Measurement Id
- measurement date
- measurement systolic
- measurement diastolic

Measurement Request:

```
curl --location --request POST 'https://wbsapi.withings.net/measure' \
--header 'Authorization: Bearer asdf' \
--header 'Content-Type: application/x-www-form-urlencoded' \
```

```
--data-urlencode 'action=getmeas' \
--data-urlencode 'Authorization=Authorization' \
--data-urlencode 'meastype=meastype' \
--data-urlencode 'meastypes=meastypes' \
--data-urlencode 'category=category' \
--data-urlencode 'startdate=startdate' \
--data-urlencode 'enddate=enddate' \
--data-urlencode 'offset=offset' \
--data-urlencode 'lastupdate=integer'
```

Response:


```
{
  "status": 0,
  "body": {
    "updatetime": 1611308688,
    "timezone": "Europe/Athens",
    "measuregrps": [
      {
        "grpid": xxxxxxxyyyyyyxxxxx,
        "attrib": 2,
        "date": 1608121898,
        "created": 1608121898,
        "category": 1,
        "deviceid": null,
        "hash_deviceid": null,
        "measures": [
          {
            "value": 1820,
            "type": 4,
            "unit": -3,
            "algo": 0,
            "fm": 131
          }
        ],
        "comment": null
      },
      {
        "grpid": xxxxxxxyyyyyyxxxxx,
        "attrib": 2,
        "date": 1608121898,
        "created": 1608121898,
        "category": 1,
```

```
            "deviceid": null,
            "hash_deviceid": null,
            "measures": [
              {
                "value": 9900,
                "type": 1,
                "unit": -2,
                "algo": 0,
                "fm": 131
              }
            ],
            "comment": null
          }
        ]
      }
    }
```


3.4.3.3 Screens of BloodPressure Measurements

The data are displayed in the Blood Pressure Screen. The blood pressure screen provides the latest sync measurement screen and the history of the measurements. The user can see his latest systolic and diastolic pressure with an info message that indicates a conclusion of the result based on generic health rules. Respectively on the history screen there are all the measurements.

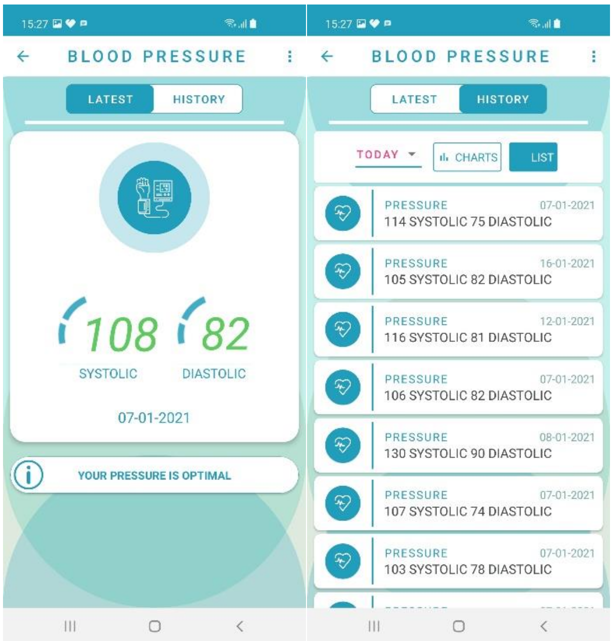


Figure 36: Blood Pressure data display on Smart Bear Mobile Application

4 Measurements, Data Types, Data Values and Metadata

4.1 *Withings Body+*

The measurements that can be retrieved by the API are presented in Table 7.

Description	Value type	Frequency of measurement (minimum)
Body weight	kg, floating point	Weekly (x1)
Body muscle mass	kg, floating point	Weekly (x1)
Body bone mass	kg, floating point	Weekly (x1)
Body fat mass	kg, floating point	Weekly (x1)
Body fat free mass	kg, floating point	Weekly (x1)
Body fat ratio	%, integer	Weekly (x1)

Table 7: List of measurements from the smart body scale device.

Weight units supported by the device:

- kilogram (kg)
- pound (lb)
- Imperial stone:pound (StLb)

Height units supported by the device:

- meter (m)
- inch (in)

Specifications:

1. Model: Withings Body+.
2. Length: 325 mm (12.8’’).
3. Width: 325 mm (12.8’’).
4. Depth: 23 mm (0.90’’).
5. Weight: 2.1 kg (4.6 lbs).
6. Weight range: 5 kg - 180 kg (9 lbs - 396 lbs).
7. Precision: 0.1 kg (0.2 lbs).
8. Display: 6.1 cm x 4.1 cm (2.4’’ x 1.6’’), 128 x 64 pixels, Backlighting.
9. Connectivity: Wi-Fi 802.11 b/g/n (2.4 GHz only), Bluetooth / Bluetooth Low Energy (Bluetooth Smart Ready).
10. 4 weight sensors.
11. Metrics Weight, Fat mass (%) - standard and athlete mode, Muscle mass (kg), Bone mass (kg), Water mass (%).
12. Batteries: 18 months battery life, 4 AAA batteries.
13. Other features: Position Control™ technology, Automatic user recognition.

Data request parameters:

- Meastypes, integer
  List of measure types.

- Category, integer
  1 for real measures, 2 for user objectives.
- Startdate, integer <timestamp>
  Measures' start date.
- Enddate, integer <timestamp>
  Measures' end date.
- Offset, integer
  When a first call returns more:1 and offset:XX, set value XX in this parameter to retrieve next available rows.
- Lastupdate, integer
  Timestamp for requesting data that were updated or created after this date. Can be used instead of 'startdate' & 'enddate'.

Example of data request:

"status": 0,
"body":
    "updatetime": "string",
    "timezone": "string",
    "measuregrps":
        "grpid": 12,
        "attrib": 1,
        "date": 1594245600,
        "created": 1594246600,
        "category": 1594257200,
        "deviceid": "892359876fd8805ac45bab078c4828692f0276b1",
        "measures":
            "value": 65750,
            "type": 1,
            "unit": -3,
            "algo": 3425,
            "fm": 1,
            "fw": 1000
        "comment": "A measurement comment"

4.2 *Withings BPM core*

The measurements that can be retrieved by the API are presented in Table 8.

Description	Value type	Frequency of measurement (minimum)
Diastolic Blood Pressure	mmHg, integer	Daily (x2)

Systolic Blood Pressure	mmHg, integer	Daily (x2)
Heart Pulse	BPM, integer	Daily (x2)
ECG signal	μV, time series (20 sec, Fs=500Hz)	Daily (x2)
Afib classification	string	Daily (x2)
Digital stethoscope signal	time series (20 sec)	Daily (x2)
Digital stethoscope classification	string	Daily (x2)

Table 8: List of measurements from the smart blood pressure device.

Specifications:
1. Model: BPM Core.
2. Length: 560 mm (22 in.).
3. Width: 165 mm (6.4 in.).
4. Height: 450 mm (unfolded) (17.7 in.).
5. Weight: 430 g (0.947 lb).
6. Cuff circumference: fits arm circumferences from 22 to 42 cm (9 to 17 in)
7. 3-lead ECG: two stainless-steel electrodes located inside the cuff, and one on the tube.
8. Digital stethoscope: silicone membrane and stainless-steel support.
9. Display: LED matrix screen.
10. Pressure sensor: gauge sensor.
11. Sensor: semiconductor pressure sensor.
12. Connectivity: Bluetooth and Wi-Fi.
13. Storage and memory: Up to 6 measurements can be stored on the BPM Core without synchronization in Bluetooth or Wi-Fi.
14. Technology: Cuff oscillo metric method.
15. Measurement range: 0 to 285 mmHg, SYS 60 to 230 mmHg.
16. Accuracy: ±3 mmHg or 2% of reading (pressure) / ±5% of reading (pulse).
17. Cuff inflation: automatic inflation with air pump at 15 mmHg/s.
18. Certifications: Medical EC approved (under review to be FDA cleared), 2008 ANSI/AAMI/ISO 81060-2:2013, EN ISO 81060-2:2014, FCC part B 15B: 2013.
19. International standards: European Society of Hypertension (ESH) for all countries except USA and Canada, American Heart Association (AHA) for USA and Canada.
20. Metrics: Heart rate measurement range: from 40 to 180 beats per minute.
21. Battery life: Up to 6 months; rechargeable via micro-USB cable.
22. Operation temperatures: 10°C - 40°C (50°F and 104°F).
23. Storage conditions: stored in a clean, dry location between -25°C and 55°C (-13°F and 131°F).

The device offers a 3-lead ECG recording along with blood pressure monitoring. The ECG signal is recorded in micro-volt (μV). The ECG functionality in this device is clinically validated and it comes with medical CE marking. The raw ECG signals recording can be retrieved using the 'Heart v2 – Get' service from the API, which provides access to the full data set of all ECG recordings. The basic characteristics of each EEG recording include:

- a duration of 20 seconds ECG recording,

- signal acquisition with a sampling frequency of 500 Hz,
- a total number of 10,000 samples per raw ECG recording, and
- an ECG plot diagram with scaling of 25 mm/s, 10 mm/mV.

Example of ECG data request:

"status": 0,
"body":
    "signal": "[1,2,3,4,5,6,7,8,9]",
    "sampling_frequency": 500,
    "wearposition": 1 *

*Wear position:
Unknown = -999999
Right wrist = 0
Left wrist = 1
Right arm = 2
Left arm = 3
Right foot = 4
Left foot = 5

The ‘Heart – List’ service of the API can be used to get a list of all ECG recordings performed in the chosen period of time, along with the Afib classification for each one of them, the blood pressure and the heart rate measurements.

Data request parameters:

- Startdate, integer <timestamp>
  Measures' start date.
- Enddate, integer <timestamp>
  Measures' end date.
- Offset, integer
  When a first call returns more:true and offset:XX, set value XX in this parameter to retrieve next available rows.

Example of data request:

"status": 0,
"body":
    "series":
        "deviceid": "892359876fd8805ac45bab078c4828692f0276b1",
        "model": 44, *
        "ecg":
            "signalid": 48,
            "afib": 1 **
        "bloodpressure":
            "diastole": 100,

"systole": 101
"heart_rate": 82,
"timestamp": 1594159644

*Device model:
Unknown = -999999
BPM CORE = 44
**Atrial fibrillation classification:
Unknown = -999999
Negative = 0
Positive = 1
Inconclusive = 2

4.3 *Withings Thermo*

The measurements that can be retrieved by the API are presented in Table 9.

Description	Value type	Frequency of measurement (minimum)
Temperature	°C, integer	Daily (x2)
Body temperature	°C, integer	Daily (x2)
Skin temperature	°C, integer	Daily (x2)

Table 9: List of measurements from the smart thermometer device.

Temperature units supported by the device:
- degree Celsius (°C)
- Fahrenheit (°F)

Specifications:
1. Model: Withings Thermo.
2. Length: 116 mm (4.57 in.).
3. Diameter: 33.2 mm (1.31 in.).
4. Weight: 75 g (0.165 lbs); including batteries.
5. Made of ABS and thermoplastic elastomer.
6. Connectivity: Wi-Fi 2.4 GHz 802.11 b/g/n, Bluetooth Smart Ready, MFI compatibility, Apple Wireless Accessory Configuration Support (WAC).
7. 16 infrared temperature sensors, 2,000 temperature measurements per second.
8. Numeric LED temperature display.
9. Fever status indication.
10. Accuracy: 0.2°C (0.4°F) for 35.5°C - 42°C (95.9°F - 107.6°F) range, 0.3°C (0.5°F) accuracy outside 35.5°C - 42°C (95.9°F - 107.6°F) range.
11. Battery life: Two years with two AAA (LR03) batteries.

The data request parameters and an example of the format of a data request are presented in Withings Body+ device section above.

4.4 *iHealth Air Smart Pulse Oximeter*

The measurements that can be retrieved by the API are presented in Table 10.

Description	Value type	Frequency of measurement (minimum)
Blood oxygen saturation	%, integer	Daily (x2)
Pulse rate	BPM, integer	Daily (x2)

Table 10: List of measurements from the smart pulse and oximeter device.

Specifications:

1. Model: PO3M.
2. Internally powered, type BF.
3. Enclosure degree of ingress protection: IPX1.
4. LED display.
5. Powered by battery; 3.7V Lithium-ion, 330mAh.
6. Peak wavelength: 660nm/880nm.
7. Maximum optical output power: 1mW.
8. SpO2 Measuring Range: 70-99%.
9. Average Root Mean Square (ARMS) of SpO2 Accuracy: 80%-99%:±2%, 70%-79%: ±3%, <70%: no definition.
10. Pulse Rate Measuring Range: 30/min-250/min.
11. Pulse Rate Accuracy: 30/min ~ 99/min: ±2, 100/min ~ 250/min: ±2%.
12. Data update period: 15 seconds.
13. Automatic shut-off after 8 seconds of no indication on the sensors.
14. Operation environment conditions: Temperature 5°C-40°C; Humidity <80%; Atmospheric pressure: 700hPa-1060hPa.
15. Storage conditions: Temperature -20°C-55°C; Humidity <95%; Atmospheric pressure: 700hPa-1060hPa.

Data request parameters:

- "Oxygen value", public,
  double BO { get; set; }
- "Heart rate", public,
  double, HR { get; set; }
- "Measure time", public,
  Long, MDate { get; set; }
- "note", public,
  string, Note { get; set; }
- "Longitude", public,
  double, Lon { get; set; }

- "Latitude", public,
  double, Lat { get; set; }
- "Data only Serial number", public
  string, DataID { get; set; }
- "Last change time", public,
  long, LastChangeTime { get; set; }
- "Data Source the value is 'Manual' or 'FromDevice'", public,
  string, DataSource { get; set; }
- "UserId", public,
  string, userid { get; set; }
- "Time zone of measurement location", public,
  string, TimeZone { get; set; }

4.5 *GARMIN VIVOSPORT Smart watch*

The measurements that can be retrieved by the API are presented in Table 11.

Description	Value type	Frequency of measurement (minimum)
Number of steps	number of steps, integer	Continuous
User's goal for number of steps	number of steps, integer	-
Distance traveled	meters, floating point	Continuous
Active kilocalories (burned through movement)	kCal, integer	Continuous
BMR Kilocalories (burned by existing Basal Metabolic Rate)	kCal, integer	-
Intensity Minutes	minutes, integer	Continuous
Total duration of vigorous intensity activities (>600 seconds)	seconds, integer	Continuous
Total duration of moderate intensity activities (>600 seconds)	seconds, integer	Continuous
User's goal for duration of moderate to vigorous intensity activity	seconds, integer	-
Floors Climbed	Number of floors, integer	Continuous
User's goal for floors climbed	Number of floors, integer	-
Average heart rate (last 7 days)	BPM, integer	-
Average heart rate at rest	BPM, integer	Continuous

Minimum heart rate value	BPM, integer	Continuous
Maximum heart rate value	BPM, integer	Continuous
Entries of mappings between offset from start time in seconds to a heart rate value in previous 15 seconds	BPM, map	-
Average stress level	number, integer	-
Duration in stress (stress levels 26-100)	seconds, integer	Continuous
Duration in no stress (stress levels 1-25)	seconds, integer	Continuous
Duration in low stress (stress levels 26-50)	seconds, integer	Continuous
Duration in medium stress (stress levels 51-75)	seconds, integer	Continuous
Duration in high stress (stress levels 76-100).	seconds, integer	Continuous
Stress duration while engaging in physical activity (non-reliable)	seconds, integer	Continuous
Maximum stress level	number, integer	-
Stress quality	label, string	-
Type of activity	label, string	-
Duration of activity	seconds, integer	During activity (continuous)
Distance covered	meters, floating point	During activity (continuous)
Steps	number of steps, integer	During activity (continuous)
Average heart rate	BPM, integer	During activity (continuous)
Maximum heart rate	BPM, integer	During activity (continuous)
Average speed	meters per second, floating point	During activity (continuous)
Maximum speed	meters per second, floating point	During activity (continuous)
Average pace	minutes per kilometer, floating point	During activity (continuous)
Maximum pace	minutes per kilometer, floating point	During activity (continuous)

Average cadence while biking	rounds per minute, floating point	During activity (continuous)
Maximum cadence while biking	rounds per minute, floating point	During activity (continuous)
Average cadence while running	steps per minute, floating point	During activity (continuous)
Maximum cadence while running	steps per minute, floating point	During activity (continuous)
Average cadence while swimming	strokes per minute, floating point	During activity (continuous)
Kilocalories burned during activity	kCal, integer	During activity (continuous)
Number of active lengths	number, integer	-
Starting point latitude	degree, floating point	Once
Starting point longitude	degree, floating point	Once
Total elevation gain	meters, floating point	Continuous
Total elevation loss	meters, floating point	Continuous
Move IQ Events activity duration	seconds, integer	Continuous
Move IQ Events activity type	label, string	Continuous
Move IQ Events activity subtype	label, string	Continuous
Total sleep duration	seconds, integer	Continuous
Time spent in deep sleep	seconds, integer	Continuous
Time spent in light sleep	seconds, integer	Continuous
Time spent in REM sleep	seconds, integer	Continuous
Time spent awake	seconds, integer	Continuous
Time that sleep could not be measured	seconds, integer	Continuous
Entries of sleep level in deep, light, and awake	unix timestamps in seconds, map	-
Sleep quality	label, string	-
Maximum volume of oxygen consumed per minute per kilogram of body weight at maximum performance (Vo2Max)	milliliters, floating point	During activity
Fitness age	age, integer	During activity
Temperature (outside)	°C, integer	-
Weather forecast	label, string	-

Table 11: List of measurements from the smart watch device.

- The device learns and proposes a new step goal for each day.
- The total number of floors climbed so far and goal for the day.
- Current stress level: The device measures your heart rate variability while you are inactive to estimate your stress level. A lower number indicates a lower stress level. The stress level range is from 0 to 100, where 0 to 25 is a resting state (i.e not stressful), 26 to 50 is low stress, 51 to 75 is medium stress, and 76 to 100 is a high stress state.
- Stress quality: A qualitative label summarizing all stress measurements (e.g. 'unknown', 'calm', 'balanced', 'stressful', 'very_stressful', 'calm_awake', 'balanced_awake', 'stressful_awake', 'very_stressful_awake').
- Move IQ Event activities are automatically detected by the device.
- Sleep quality: The data could be auto-confirmed, but the sleep window could have been manually adjusted, or the sleep data itself is entirely manually entered. Possible values: 'MANUAL', 'DEVICE', 'AUTO_TENTATIVE', 'AUTO_FINAL', 'AUTO_MANUAL', 'ENHANCED_TENTATIVE', 'ENHANCED_FINAL'. Naps are not considered in sleep statistics.
- Fitness age: An estimation of user's fitness level, calculated by comparing internal fitness metrics with the average readings of biometrically similar users by age and gender.
- The current temperature and weather forecast can only be obtained from a paired smartphone.

Specifications:

1. Model: GARMIN VIVOSPORT.
2. Lens Material: Chemically strengthened glass.
3. Case material: Polymer.
4. Strap material: Silicone.
5. Width: 21.0 mm.
6. Thickness: 10.9 mm.
7. Small/medium fits wrists with a circumference of 122-188 mm.
8. Large fits wrists with a circumference of 148-215 mm circumference.
9. Touchscreen color display.
10. Display size: 9.7 mm x 19.3 mm (0.38" x 0.76").
11. Display resolution: 72 x 144 pixels
12. Display type: sunlight-visible, transflective memory-in-pixel (MIP)
13. Weight: Small/medium: 24.1 g, Large: 27.0 g.
14. Battery life: Up to 8 hours in GPS mode, up to 7 days in smartwatch mode.
15. Water rating: Swim.
16. Memory/History: 7 timed activities, 14 days of activity tracking data.
17. Clock Features: Time/date, GPS Time Sync, Automatic daylight saving time, Alarm clock, Timer, Stopwatch.
18. Health Monitoring: Wrist-based Heart Rate (constant, every second), Daily Resting Heart Rate, Fitness Age, All-day Stress Tracking, Relaxation breathing timer, Sleep, Hydration, Woman's Health Tracking.
19. Sensors: GPS, Garmin Elevate™ wrist heart rate monitor, Barometric altimeter, Accelerometer.
20. Connectivity: Bluetooth® Smart and ANT+®
21. Feature: Smart notifications, Weather, Controls smartphone music, Find My Phone, Find My Watch, VIRB® Remote.

22. Activity tracking features: Step counter, Move bar, Auto goal (i.e. device learns activity level and assigns a daily step goal), Calories burned, Floors climbed, Distance traveled, Intensity minutes, TrueUp™, Move IQ™.
23. Training features: HR zones, HR alerts, HR calories, % HR max, HR Broadcast (i.e. over ANT+™ to paired devices), GPS-based speed and distance, Customizable data pages, Auto Pause®, Auto Lap®, VO2 Max (Run), Custom alerts, Touch and/or button lock, Activity history on watch.

Data request parameters:

- summaryId, string: Unique identifier for the summary.
- calendarDate, string: The calendar date this summary would be displayed. The date format is 'yyyy-mm-dd'.
- startTimeInSeconds, integer: Start time of the activity in seconds using Unix timestamp.
- startTimeOffsetInSeconds, integer: Offset in seconds to add to startTimeInSeconds to derive internal time of the device.
- activityType, string: Included in daily summaries for backwards compatibility purposes. It will always default to 'WALKING'.
- durationInSeconds, integer: Length of monitoring period in seconds; 86400 once a full day is complete, but less if a user syncs mid-day.
- activeTimeInSeconds, integer: Portion of the monitoring period in seconds considered as 'Active'. This relies on heuristics internal to the device.
- isParent, Boolean: If present and set to true, this is the parent activity of one or more child activities that should also be made available in the data feed (e.g. 'MULTI_SPORT' type of activity).
- parentSummaryId, String: If present, this is the summaryId of the related parent activity.
- Manual, Boolean: Indicates that an activity was manually entered.

4.6 *Home sensors*

The measurements that can be retrieved by openHAB are presented in the table below.

Description	Value type	Frequency of measurement (minimum)
Room light intensity	Illuminance, integer	Continuous
Room temperature	°C, floating point	Continuous
Presence Sensor	detection event, date	Continuous
Humidity	%, floating point	Continuous
Atmospheric pressure	hPa, integer	Continuous
Battery percentage	%, integer	Continuous
Whether condition	label, string	-
Outdoor temperature	°C, floating point	-
Outdoor barometric pressure	hPa, floating point	-

Outdoor atmospheric pressure	%, integer	-
Wind speed	kilometers per hour, floating point	-
Wind direction	degrees, integer	-
Rain volume (last hour)	millimeters , floating point	-
Snow volume (last hour)	millimeters, floating point	-
Visibility	kilometers, floating point	-
Current UV index	integer	-

Table 12: List of measurements from the smart home devices.

Note: All whether and outdoor readings are software-based inputs through open weather API.

Specifications for Hue sensor:
1) Model: Hue Motion sensor.
2) Color: White.
3) Material: Plastic.
4) Height: 5.5 cm (2.16 inch).
5) Length: 3 cm (1.18 inch).
6) Net weight: 0.065 kg (0.14 lb).
7) Width: 5.5 cm (2.16 inch).
8) IP rating: IP42.
9) Batteries: 2xAAA (included)
10) Minimum battery life: 2 years.
11) Mounting options: Freestanding, wall.
12) Operational humidity: Suitable for use only indoors.
13) Operational temperature: 0°C - 40°C.

Specifications for Aqara sensor:
1) Model: WSDCGQ11LM.
2) Wireless Protocol: Zigbee.
3) Battery: 1xCR2032.
4) Minimum battery life: 2 years.
5) Dimensions: 36 × 36 × 9 mm (1.42 × 1.42 × 0.35 in.)
6) Temperature range: -20°C – +50°C (-4°F – +122°F).
7) Temperature precision: ±0.3°C (±0.5°F).
8) Humidity range: 0 – 100% RH (non-condensing).
9) Humidity precision: ±3%.
10) Atmospheric pressure range and Precision: 30 kPa – 110 kPa.
11) Atmospheric pressure precision: ±0.12 kPa.

The Philips Hue motion sensor can be used in various use case scenarios such as to:
- Trigger the Philips Hue smart lights when motion is detected, helping the user navigate safely.
- Customize the Philips Hue smart lights to either turn on or adjust their intensity, depending on the time of day or other conditions.

- Customize the Philips Hue smart lights using the integrated light sensor, to detect if there is still enough daylight or the lights should turn on.

Note: This product requires a connection to the Hue Bridge.

Data request parameters:

- Temperature, Integer: Temperature reading from Hue sensor.
- Illuminance, Number: Illuminance reading from Hue sensor.
- Light Level, Number: Light level reading from Hue sensor.
- Last Updated, string: Date and time of motion Hue sensor activation.
- Battery Level, Number: Battery percentage left.
- Temperature precision, Number: Controls the precision of temperature values of Aqara sensor, e.g. 0, 1 or 2; default 2.
- Temperature calibration, Number: Allows manual calibration of temperature values of Aqara sensor, e.g. 1 would add 1 degree to the temperature reported by the device; default 0.
- Humidity precision, Number: Controls the precision of humidity values of Aqara sensor; default 2.
- Pressure precision, Number: Controls the precision of pressure values of Aqara sensor; default 1.
- Pressure calibration, Number: Allows manual calibration of pressure values of Aqara sensor; default 0.

4.7 *Hearing aids*

The integration of the hearing aids devices has not be fully documented since more information is required from the manufacturer's side, and thus Table 13 below presents an initial list of measurements that have been identified as available at this stage, but the implementation-readiness for each one will be defined in the following versions of the mobile-sensor integration platform.

Description	Value type	Frequency of measurement (minimum)
Duration of active use	hours/minutes, floating point	Continuous
Duration of active use at specific time intervals	hours/minutes, floating point	Continuous
Average duration of active use per day	hours/minutes, floating point	Continuous
Total days of usage	number of days, integer	Continuous
Duration of active use in various environments/conditions per day.	minutes, floating point	Continuous

Duration of exposure at environmental noise levels (dB SL) >85dB per day	minutes, floating point	Continuous
Duration of exposure at environmental noise levels (dB SL) >85dB per week	minutes, floating point	-
Percentage of active use in soft intensity sounds	%, floating point	Continuous
Percentage of active use in medium intensity sounds	%, floating point	Continuous
Percentage of active use in high intensity sounds	%, floating point	Continuous
Manual program adjustments by user per day	number of adjustments, integer	Continuous
Manual program adjustments by user per week	number of adjustments, integer	-
Manual volume adjustments by user per day	number of adjustments, integer	Continuous
Manual volume adjustments by user per week	number of adjustments, integer	-
Log of remote fine tuning	string	-
Number of times the user turned on the device	times per day, integer	Continuous

Table 13: List of measurements from smart body scale.

Specifications:
1) Model: PHONAK AUDEO MARVEL-50.
2) Power: Batteries (non-rechargeable version).
3) Battery life: ~5-7 days (depending on usage).
4) Antenna type: Resonant loop antenna.
5) Operation frequency: 2.4 GHz – 2.48 GHz.
6) Modulation: GFSK, Pi/4 DPSK, GMSK.
7) Radiated power: < 2.5mW.
8) Bluetooth 4.2 Dual-Mode.
9) Bluetooth® range: ~1m.
10) Profiles supported: HFP (Hands-free profile), A2DP.
11) Temperature during transportation and storage: -20° to +60° Celsius (-4° to +140° Fahrenheit).
12) Humidity during transportation: Up to 90% (non-condensing).
13) Humidity during storage: 0% to 70%, if not in use.
14) Atmospheric pressure: 200 hPA to 1500 hPa.
15) Sonova AG declares that this product meets the requirements of the Medical Devices Directive 93/42/EEC as well as the Radio Equipment Directive 2014/53/EU.

5 The Smart Bear Users' Dashboard

5.1 *Introduction*

The Smart Bear dashboard is a component aimed at providing a user-friendly graphical user interface for the clinicians of the platform. The application is responsible for managing the interactions between the cloud platform and the users who must use it to enter patients into the system, enter clinical data, launch analytics on aggregated data and check that patients are actually using the devices in their possession to save the data in the platform.

5.2 *Technology*

5.2.1 Technology stack

The dashboard was developed using JavaScript with the Vue.js framework as the main tool. In addition to the framework in question, extensions such as Vuex, vue-router, Vuetify and Axios were used. The main development tool was Visual Studio Code along with a series of specific plugins to support the syntax recognition of the chosen framework.

Vue.js is the backbone of the application because in its simplest form it only takes care of the visualization part focusing on the ViewModel layer of the MVVM pattern. It connects the View and the Model via two-way data bindings, but through the use of the other plugins it allows to extend its functionality. The framework developers themselves made these external tools available as plugins or external libraries like in the case of Vuex. Normally a Vue.js application is structured using View components but using the extension called Vuex we have added the state management pattern to the application, so we added a new component called Store to handle all the data in the application acting as a single point of truth for all the components and defining rules to ensure that the state is mutated in a predictable fashion.

Our Vue.js application is therefore composed of:

- Views (or pages): vue instances that contain different components
- Components: reusable view instances with a name and a proper lifecycle
- Store: a component that contains the logic that handle the data in the application

The main libraries used to extend the functionalities are:

- Vuex: used to implement the Flux pattern
- Vuetifyjs: an UI component library implementing the material design
- Axios: a promise-based HTTP request library that allows you to interface with REST API
- Apache ECharts: used to display charts when needed

An application of this type can be seen as a tree of nested components, this leads to a certain difficulty in managing the states of the individual components. To overcome this problem of separation between the view and its state, a tool like Vuex is used. Thanks to this tool, the data of all the components of a view are in a singleton and therefore all data readings and actions that modify this data have only one access point.

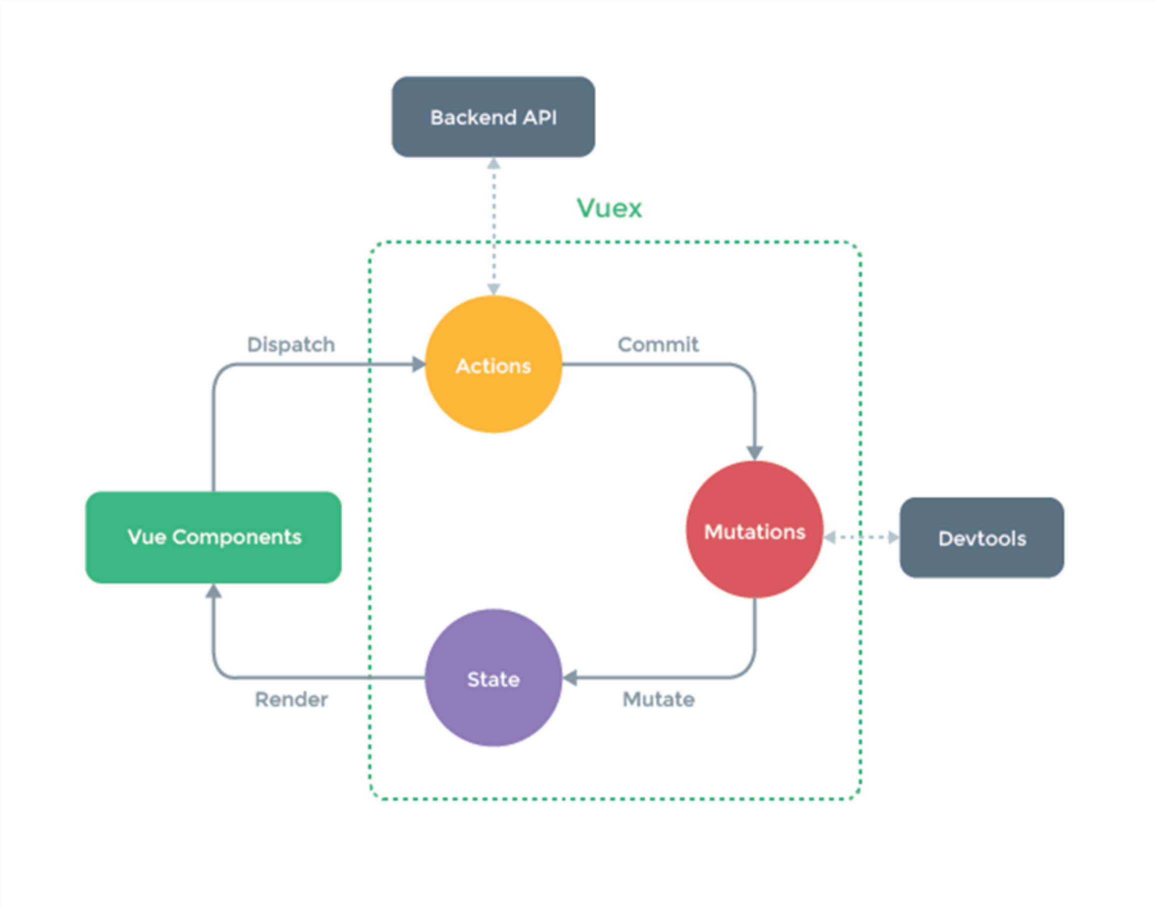


Figure 37: Vuex architecture

Figure 37 shows how the application life cycle is structured using this pattern:

- **State**: this single object contains all your application level state and serves as the single source of truth
- **Mutations**: the only way to actually change state in a Vuex store is by committing a mutation. Vuex mutations are very similar to events: each mutation has a string **type** and a **handler**
- **Actions**: are similar to mutations but instead of mutating the state, actions commit mutations and they can contain arbitrary asynchronous operations (like REST API calls)

In order to provide a satisfying user experience the UI was built using *Vuetifyjs* framework. This library was created on the basis of Vue.js to help the developer in creating SPAs suing a mobile-first approach and exploiting the guidelines of Material Design. The library provides a rich set of highly customizable components that can also be extended to provide highly specialized versions and the ability to integrate the internationalization that will be used for the different project pilots.

Axios is the library used to make calls to the REST APIs of the cloud platform services, the choice to use this library is motivated by the best support provided in error handling and by better support for different browsers and their older versions.

5.3 *Development Methodology*

As explained previously, the Vue.js framework provides some extensions that can be added to the project as libraries, but it is not limited to just that. In fact it deals with providing development tools such as:

- Vue CLI: a full system for rapid prototyping Vue.js application
- Vue Devtools: a browser devtools extension for debugging Vue.js applications

Vue Cli was installed using Yarn (a package manager for javascript) from the command line, which is a tool that provides:

- Interactive project scaffolding
- Zero config rapid prototyping
- A collection of official plugins
- A graphical user interface to create and manage different projects like in Figure 38

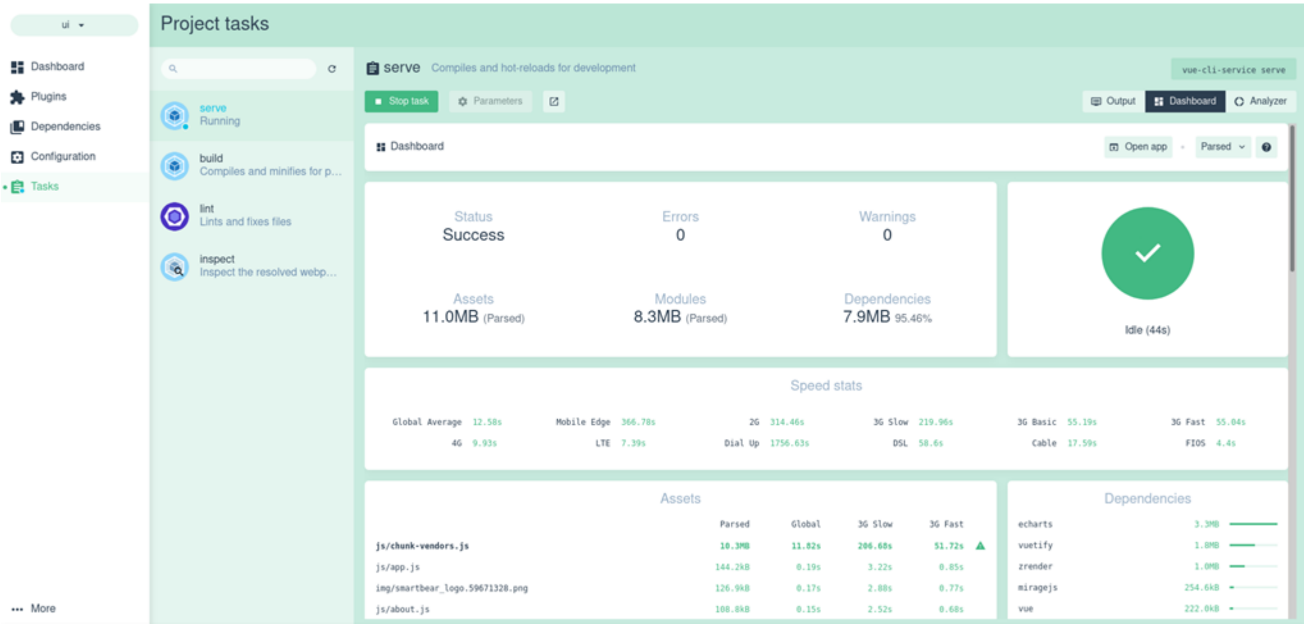


Figure 38: *Vue UI*

It is the main development tool as it provides the possibility of providing an opinionated and pre-configured template for webpack while also ensuring that all the various plugins and configuration tools work together in a functional way without having to reconfigure everything every time. Vue Cli also monitors the project folder via hot-reloading and serves a local version of the application that is being developed in order to have a live preview and be able to monitor progress at each save. Finally, the tool is also used to build the project in a minified version to use it in a production environment.

Vue Devtools is a browser extension that could be installed both on Chrome and Firefox and is used to debug a Vue.js application. The tool allows inspecting component trees, reviewing events, and time-travel debugging of Vuex states. Having these features makes debug the live application fairly easy like in Figure 39.

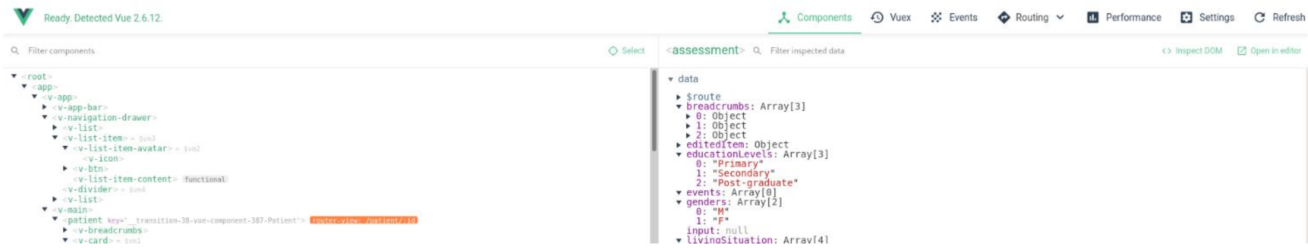


Figure 39: Vue Devtools

The extension is activated by opening the Development Console of the browser when the page in the tab contains a Vue.js app. The tool provides a DOM-style tree view of all the components of the app with information about every component, data, computed properties and binding in the page. The most useful feature is the possibility to do a Time-Travel debug, for example revert to a previous state, replay events and so on, using the Vuex pane. The Events panel provides a way to see the log of all the events emitted by the application with their names, sources and payloads.

5.4 *The Dashboard of the Smart Bear Users*

5.4.1 Home

The dashboard of the Smart Bear platform is built as an SPA application. The main page, the one that welcomes the user after he/she logs in, is mainly composed of a series of charts that show the status of the platform, or a series of aggregated data useful to understand if there are inconsistent or worrying data. Two types of alerts are also displayed: a tab regarding users who are not sending data or who seem to have some problems in the platform, the other shows the alerts regarding the status of the latest analytics launched. The Home page is shown in Figure 40.

The aggregated data presented on the main dashboard page is presented through the use of the Apache ECharts library. This library allows you to view even very complex graphs with a lot of data in a very simple way, it is sufficient to receive a JSON input. This JSON contains both the data and the actual configuration of the chart to be created. An example of a configuration can be seen below:

```
{
    tooltip: {
      trigger: 'axis'
    },
    legend: {
      data: ['Cardio', 'HighP', 'Deaf', 'Diab', 'OverW']
    },
    grid: {
      left: '3%',
      right: '4%',
      bottom: '3%',
      containLabel: true
    },
    toolbox: {
      feature: {
```

```
        saveAsImage: {}
      }
    },
    xAxis: {
      type: 'category',
      boundaryGap: false,
      data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
    },
    yAxis: {
      type: 'value'
    },
    series: [
      {
        name: 'Cardio',
        type: 'line',
        stack: 'Total',
        data: [120, 132, 101, 134, 90, 230, 210]
      },
      {
        name: 'HighP',
        type: 'line',
        stack: 'Total',
        data: [220, 182, 191, 234, 290, 330, 310]
      },
      {
        name: 'Deaf',
        type: 'line',
        stack: 'Total',
        data: [150, 232, 201, 154, 190, 330, 410]
      },
      {
        name: 'Diab',
        type: 'line',
        stack: 'Total',
        data: [320, 332, 301, 334, 390, 330, 320]
      },
      {
        name: 'OverW',
        type: 'line',
        stack: 'Total',
        data: [820, 932, 901, 934, 1290, 1330, 1320]
      }
    ]
  };
```

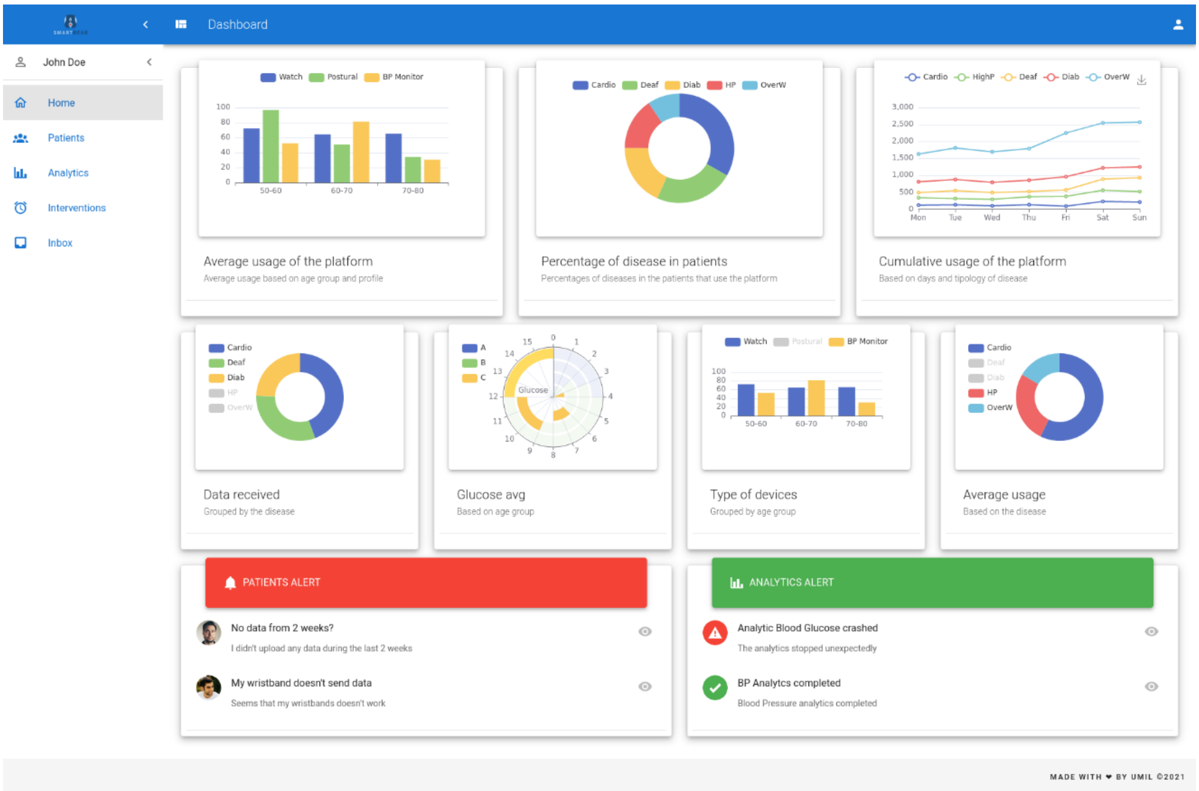


Figure 40: Home page of the dashboard

5.4.2 Patients page

This page shows as a first view a table with the users of the platform. Inside the page there is the possibility to create new users, delete and open the edit page where there is the possibility to insert a new assessment or view those previously carried out. The patients page is shown in Figure 41.

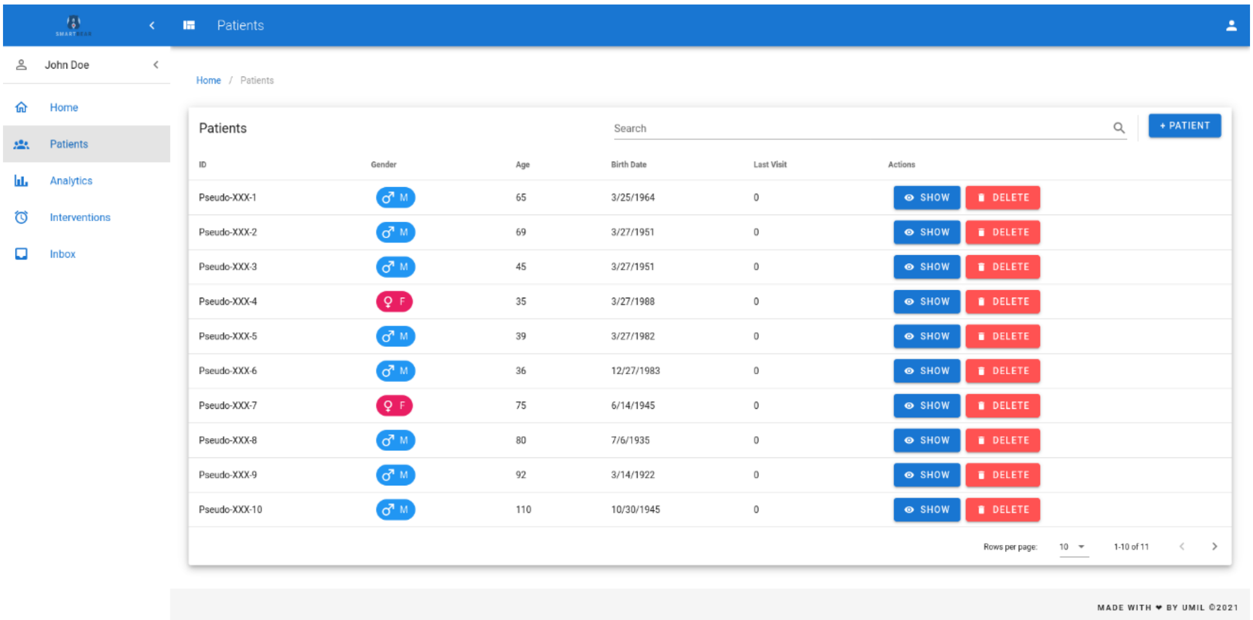


Figure 41: Patients page

5.4.3 Patient page

The patient page is the main point of aggregation of data concerning the single person entered in the system. Within the page it is possible to view and modify the general parameters of the individual patient, open a previous assessment or create a new one and save the data in the FHIR repository. It is also possible to assign and see which devices have been assigned to a single user or to assign new ones. In the clinical data part, you can read aggregate data on the individual patient to check if there are any problems with sending data or using the platform. The patient page is shown in Figure 42.

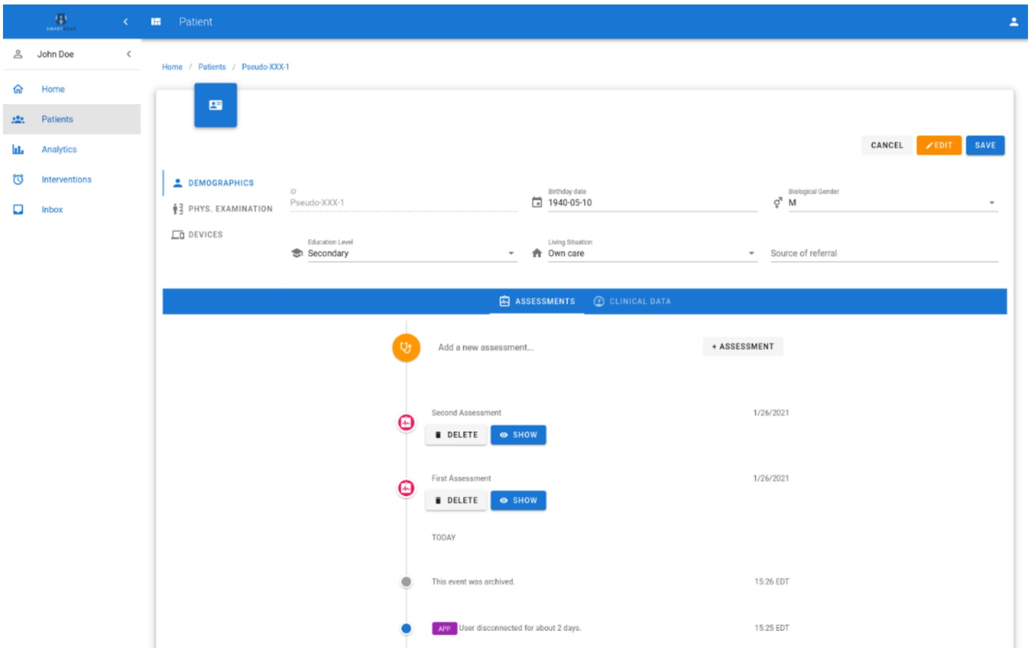


Figure 42: Patient page

5.4.4 Analytics

The analytics page shows the table of analytics that have been launched, terminated and failed. The user can create a new one by using a guided path that allows the configuration of pre-existing analytics in the system by filling in some parameters as it is shown in Figure 43.

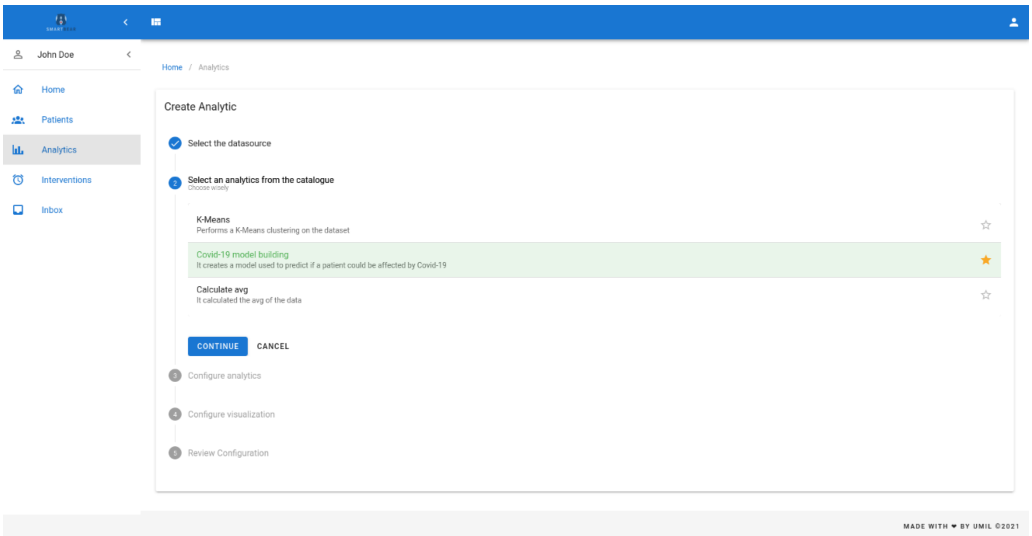


Figure 43: Create Analytic page

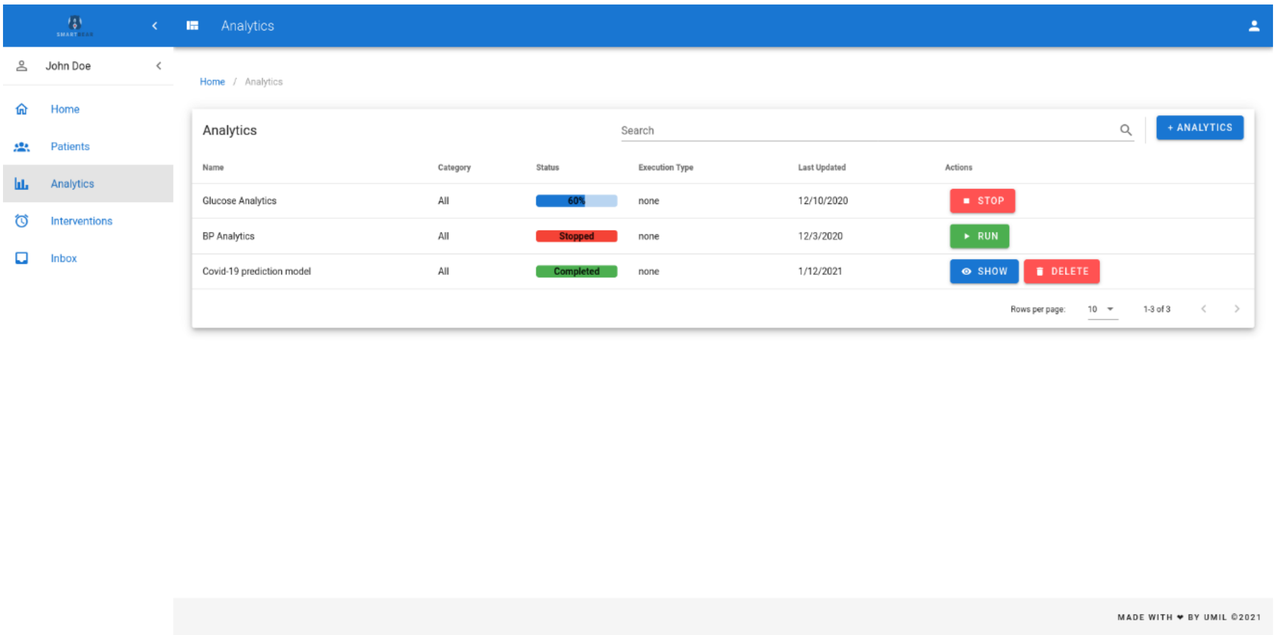


Figure 44: Analytics page

If the analytic has successfully completed its execution, you can view the results on its recap page. The analytics page is shown in Figure 44.

5.4.5 Inbox

The inbox page is the one where the alerts flow of which the user has a peek preview in the initial dashboard. On this page, the table shows the alerts that are generated by the system, such as those that warn that a patient has not been sending his wearable data for a few days. The user can then send an alert to the patient to notify him/her of the problem. The Inbox page is shown in Figure 45.

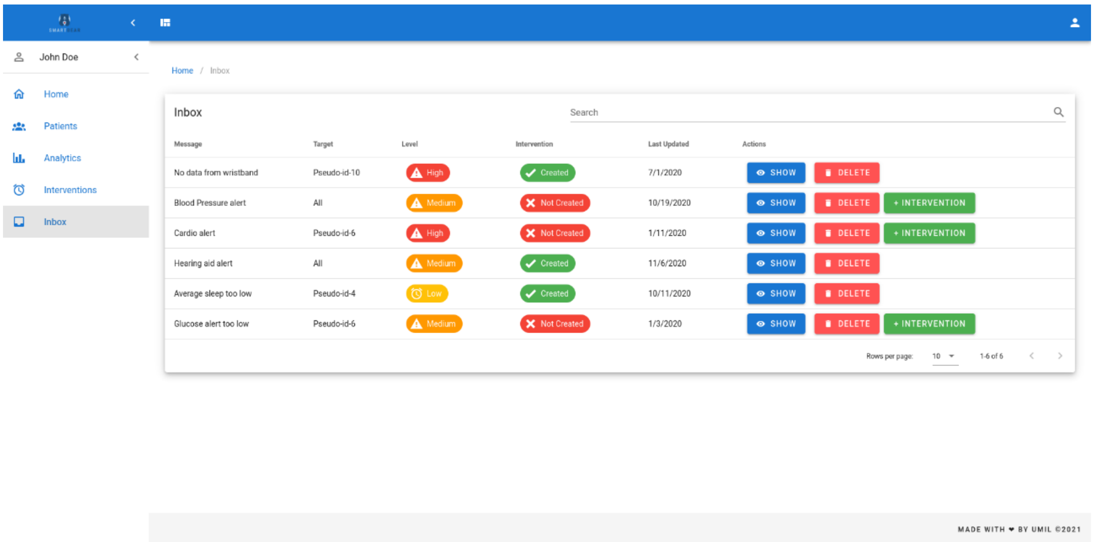


Figure 45: Inbox page

5.4.6 Interventions

The interventions page shows the list of interventions already entered into the system, providing the user with the possibility of eliminating some interventions or creating new ones based on criteria defined by the user. The Interventions page is shown in Figure 46.

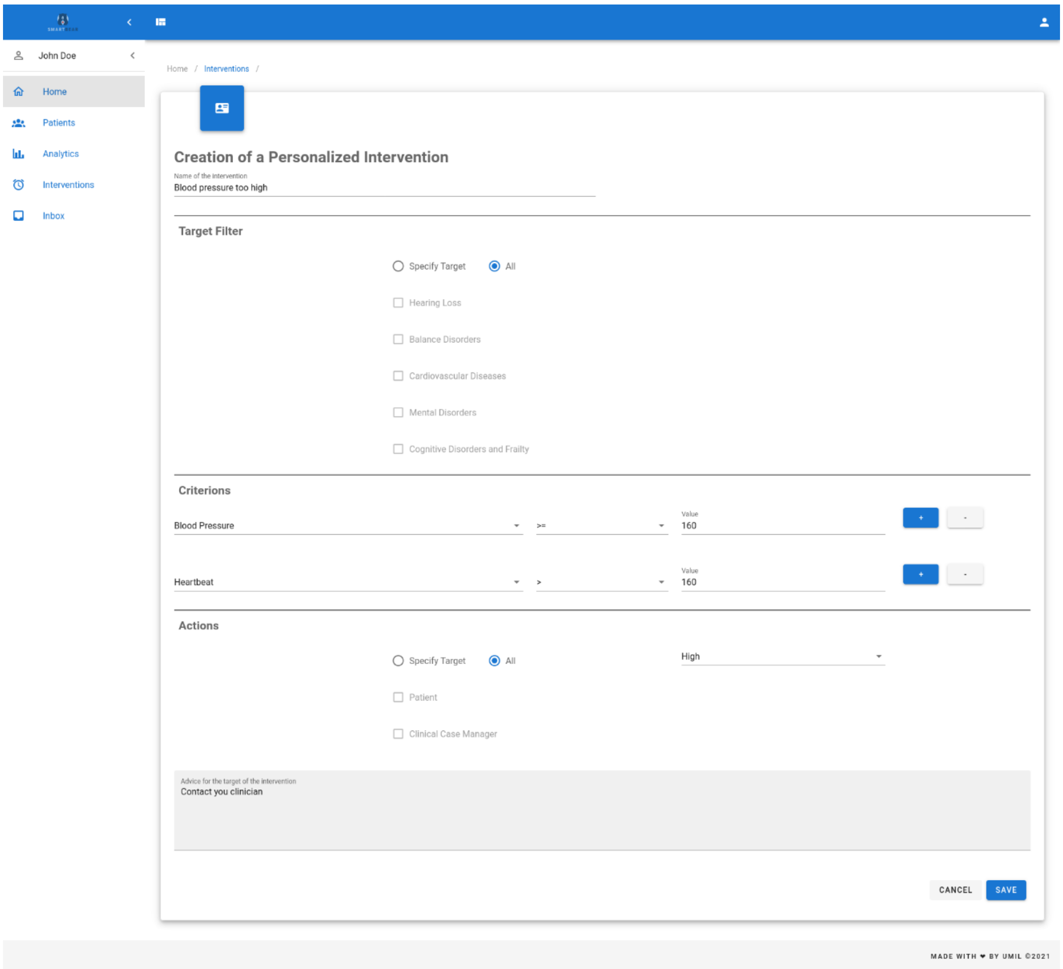


Figure 46: Intervention page

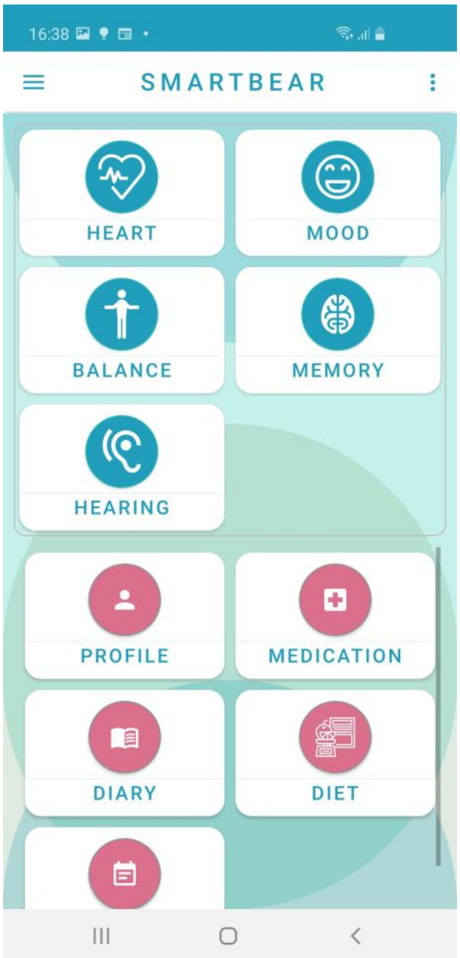
6 Appendix

6.1 *Smart Bear Mobile App Mockups*

This section includes the design of the user interface of the SB@App, based on the initial requirements of the project; the figures of the mockups that follows are not numbered.

6.1.1 Main Menu Screen

Main menu screen provides the user to navigate through the Smart Bear application. On the Main menu screen the corresponding medical groups are displayed based on the user profile along with the default sub menu apps. The Medical Groups are Heart, Balance, Mood, Memory and Hearing.
For each user, access is granted for at least 2 medical groups. While the sub menu apps for appointments, Smart Bear profile, weight, my diet and diary will be displayed by default for all the participants.



Main Menu Screen

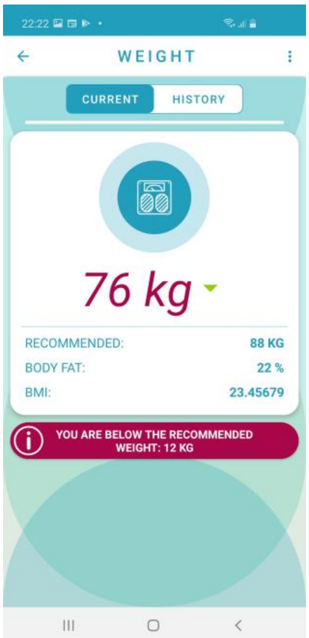
6.1.2 My Heart Menu

My Heart submenu app screen provides the navigation to the screens, Heart Rate, Pressure, Weight, Weather, Activity and Oximeter.

Heart Menu Screen

6.1.3 Weight Screen

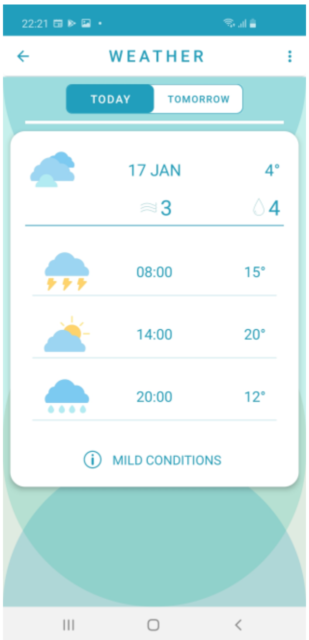
The Weight screen displays the latest measurements from the Scale device, as well the optimal measurements based on general health rules and the history of user measurements.



Weight Screen

6.1.4 Weather Screen

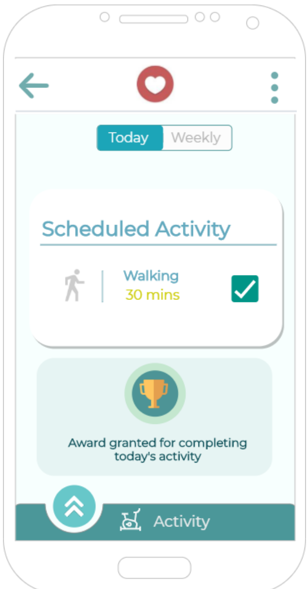
The Weather screen provides the latest weather predictions for today and days ahead provided by the abilities of the API, to inform the user regarding outdoor activity.



Weather Screen

6.1.5 Activity Screen

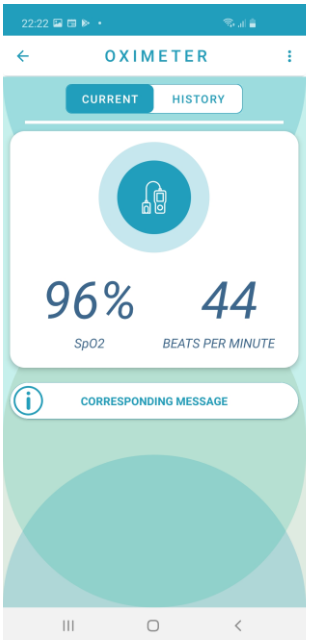
The Activity screen displays the recommended activity for the user. Rewards will be displayed for the users success upon completion.



Activity Screen

6.1.6 Oximeter Screen

The Oximeter screen displays the latest measurements from the Oximeter device, as well the history of those measurements.



Oximeter Screen

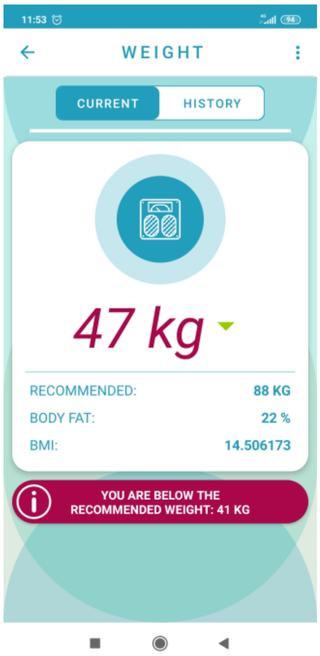
6.1.7 My Balance Menu

My Balance submenu app screen provides the navigation to the screens, Blood Pressure, Heart Rate, Weight, Weather, Activity, Rehabilitation and Benefits.

My Balance Menu Screen

6.1.8 Weight screen

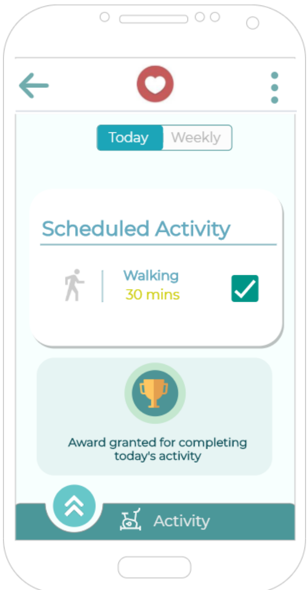
The Weight screen is the same as in the submenu of My Heart.



Weight Screen

6.1.9 Activity screen

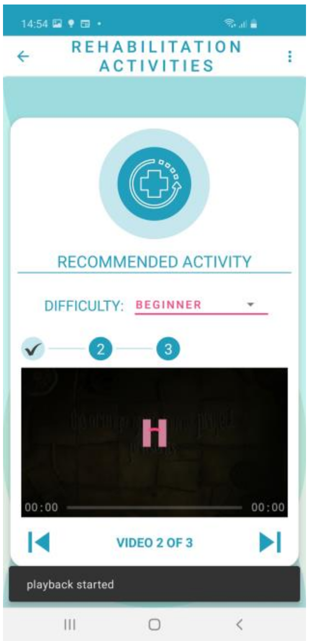
The Activity screen is the same as in the submenu of My Heart.



Activity Screen

6.1.10 Rehabilitation Activity screen

The Rehabilitation Activity offer a series of videos with activities for rehabilitation. These videos are distinguished by the user's participation and self-confirmation of fulfilling these tasks. Also there is a choice to navigate to each video.



Rehabilitation Screen

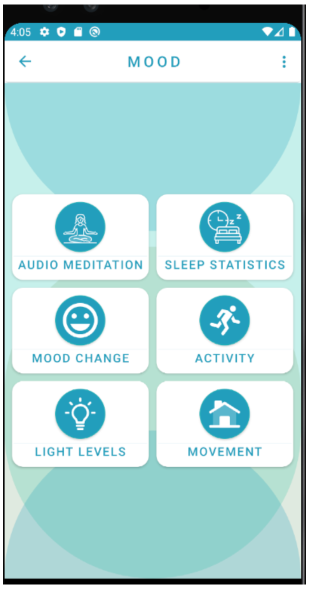
6.1.11 Benefits screen

The Benefits screen provides the user the reasons that will benefit him/her upon the completion of this program.

Benefits Screen

6.1.12 My Mood Menu

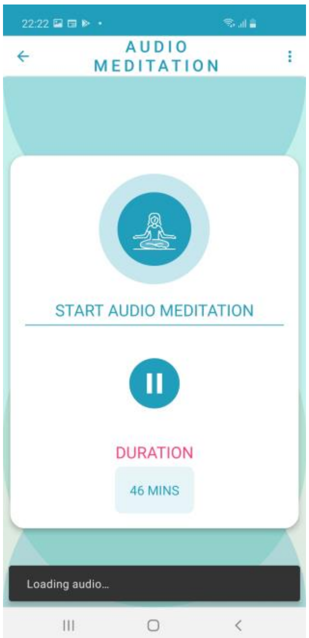
My Mood submenu app screen provides the navigation to the screens, Audio Meditation, Sleep Statistics, Mood Change, Activity, Light Levels and Movement.



Mood Menu Screen

6.1.13 Audio Meditation Screen

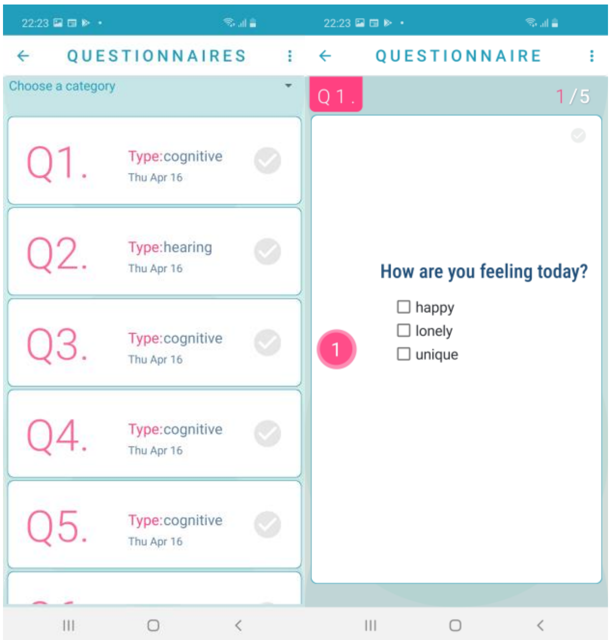
The Audio Meditation screen plays audio files for meditation usage to the user.



Audio Mediation Screen

6.1.14 Mood Change Screen

The Mood Change screen displays questionnaires that the user is required to fill.
User will be able to submit the questionnaires.



Mood Change Screen

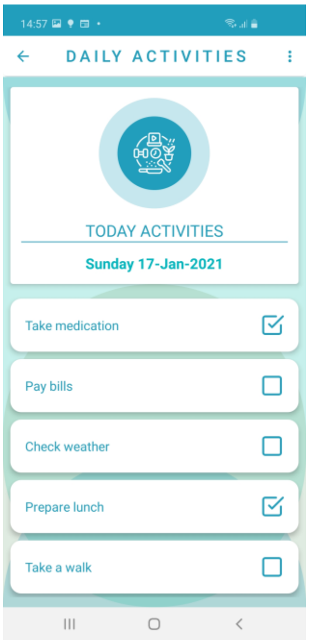
6.1.15 My Memory Menu

My Memory submenu app screen provides the navigation to the screens, Daily Activity, Light Levels and Comfort.

My Memory Screen

6.1.16 Daily Activity

The Daily Activity screen displays a self-reported questionnaire that reminds the user his daily routine.



Daily Activities Screen

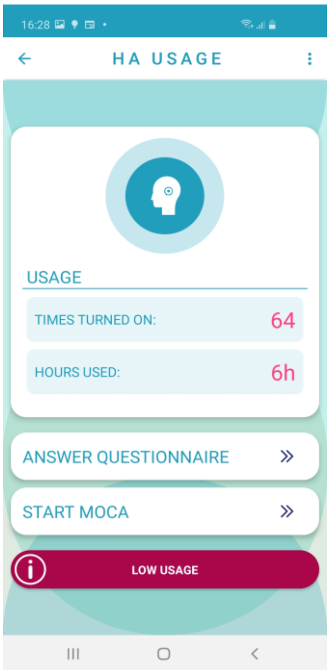
6.1.17 My Hearing Menu

My Hearing submenu app screen provides the navigation to the screens, Hearing Aids usage, Hearing Aids benefits and Changes of Hearing Aids.

My Hearing Menu Screen

6.1.18 Hearing Aids Usage Screen

The Hearing aids usage screen displays the usage of the Hearing Aids wearable. As well it provides the navigation to answer questionnaires derived from the clinicians. A corresponding message will be displayed based on general rules.



Hearings Aids Usage Screen

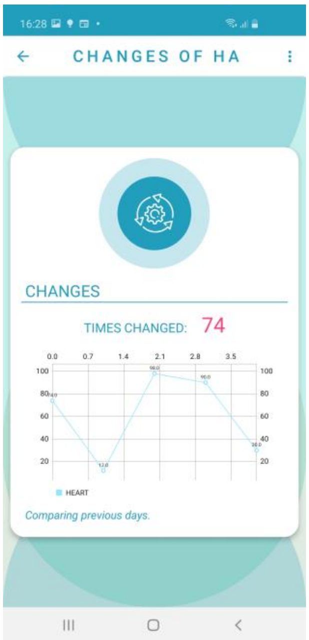
6.1.19 Hearing Aids Benefits

The Hearing Aids Benefits screen displays the benefits derived from the participation in the program. In a progress chart.

Hearing Aids Benefits Screen

6.1.20 Changes of Hearing Aids Screen

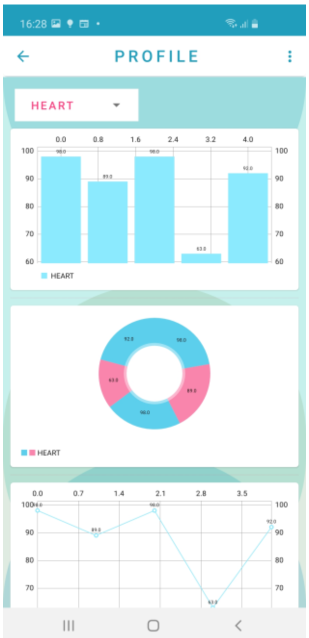
The Changes of Hearing Aids screen displays the changes that the user did on the levels of his hearing aids wearable.



Changes of Hearing Aids Screen

6.1.21 Profile Screen

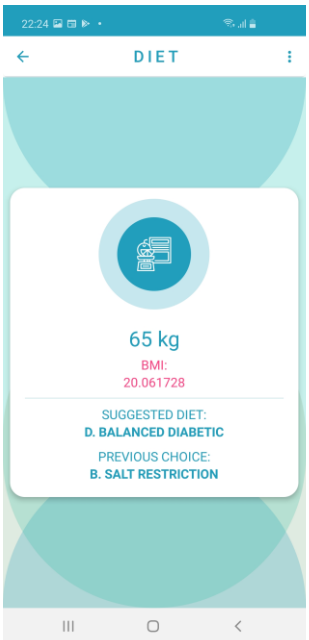
The Profile Screen will display his data in charts and some useful information.



Profile Screen

6.1.22 Diet Screen

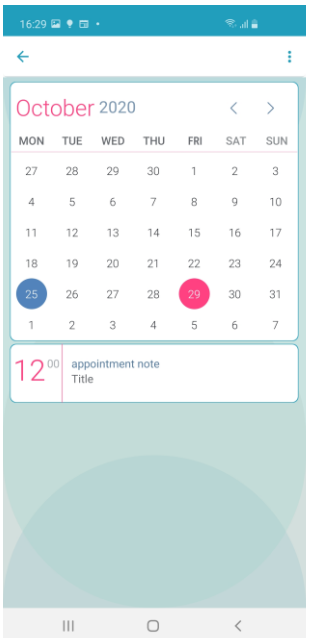
The Diet Screen will display the current weight of the user and the suggested diet. There will be a navigation in inner screens where the users must fill a questionnaire for diet determination. A pyramid with types of food that need to be consumed by the user on daily weekly and monthly basis will be displayed. A self-reported questionnaire will determine the compliance.



Diet Screen

6.1.23 Appointments Screen

The Appointments Screen displays a calendar that the user is able to arrange appointments on a specific time and date with the corresponding clinician.



Appointments Screen

6.2 *Device (of-the-self) testing with vendor application and instructions*

6.2.1 Testing the selection of the devices

6.2.1.1 Tests peformed over 2 days

We tested all devices except the smart phone which we have not received yet. We replace it by a tablet.

1/FITBIT CHARGE 4

• Charging Magnet and the watch are hard to nest each other.

• Quick and easy first setup.

• Values are quickly transmitted.

• Wristband might not be suitable to elderly persons.

• Not ergonomic enough as the display switches itself off automatically too quickly.

2/WITHINGS BPM CORE

• Not instinctive at the first use.

• Documentation not user-friendly enough for an easy use.

• Not all functions are available if the BPM CORE is not connected via WIFI. When on Bluetooth, there is an “error” message.

• Data synchronization between tablet and the BPM CORE is very long.

• We cannot be sure about the results if the BPM CORE is not well-installed on the patient’s arm.

3/WITHINGS THERMO

• Withings Thermo app is needed and must be on.

• An account between MATE and THERMO app is available.

• Fast and easy setup.

• On the app, the display of the values is not obvious (too small and at the bottom of the page).

• The sliding touch action to assign user profile to a temperature on the Thermometer might be inadequate for elderly persons.

4/IHEALTH OXIMETER

• Fast and easy to setup.

• The battery loading access is not easy for an elderly person.

• The oximeter device can store the data until we open the IHEALTH MY VITALS app to synchronize them.

• Results are correct but if you move the oximeter to reposition it better, values can become unstable.

• It works even if the lady has some nail polish.

5/WITHINGS BODY +

• Many functions available.

• Not all functions are available if the device is not connected by WIFI.

• Screen might be too small or too dark for partially sighted person.

The Fitbit Charge 4 and the BPM Core could be miss-understood by elderly persons and thus become useless for them.

6.2.1.2 Tests done on an elderly patient.

Patient's profile

- A 69-year-old woman.
- Handicapped hips, osteoarthritis of the hands and knees.
- Average knowledge of how to use connected devices.
- Already equipped with a PC, a kindle and a smartphone.

After 5 min of training on each device, she tested them by herself 3 times. We used a tablet with Bluetooth to replace the smartphone which we had not received yet.

Smart watch (Fitbit)

- Not convenient at all as the screen is too small: too dark.
- The wristband is in silicon and she's allergic, so not a good solution.
- Not easy to set around the wrist with arthritis.
- the sliding of screens is too sensible and she had real difficulties reaching the walk activity screen.
- She did not find easily the right screen to start the "walking" function as you can slide the screens up & down and from left to right and you can press on the sides of the watch; this is very confusing.
- She much prefers the smartphone to measure the number of daily steps as there is nothing to do. It starts automatically whereas with the smart watch, you have to start the walking function and then stop it every day.

Oximeter

- She likes it very much. Easy to set and convenient, easy to understand.
- The results were accurate.

Thermometer

- She likes it very much. Easy to measure and have in hands.
- She felt the vibration but could not hear it. One must not be deaf.
- The results were accurate.
- The Bluetooth connection did not work.

Blood pressure sensor

- She had real difficulties installing it as she found it too heavy and too rigid especially as she had arthritis.
- She found it too long for getting the results.

- She tried it three times and each time, there was "error" written on the tube but the results of the blood pressure were displayed. So she did not know if it has worked or not. Nothing was displayed on the tablet although everything was Bluetooth connected.
- She did not like it.
- The cuff is long and can cover the stethoscope as there is no explanations on it.

Weighing sensor

- She likes it immediately as she found it elegant, easy and very stable.
- The results were accurate.