



D6.4 NESTORE Components Integration



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769643

DELIVERABLE ID:	WP6/D6.4/TASK NUMBER 6.5
DELIVERABLE TITLE:	NESTORE Components Integration
RESPONSIBLE PARTNER:	ROPARDO
CONTRIBUTORS:	Ciprian Candea, Marius Staicu, Gabriela Candea, Claudiu Zgripcea (ROPARDO)
NATURE	Report
DISSEMINATION LEVEL:	OTH
FILE:	NESTORE Components Integration
REVISION:	V1
DUE DATE OF DELIVERABLE:	2019.08.30
ACTUAL SUBMISSION DATE:	2019.08.30
CALL	European Union's Horizon 2020 Grant agreement: No 769643
TOPIC	SC1-PM-15-2017 Personalized coaching for well-being and care of people as they age

Document History

REVISION	DATE	MODIFICATION	AUTHOR
0.1	2019.08.16	First version	Ciprian Candea, Marius Staicu, Gabriela Candea, Claudiu Zgripcea (ROPARDO)
0.2	2019.09.10	Review and update	Gabriela Candea (ROPARDO)
0.3	2019.09.16	Review and update	Silvia Orte (EURECAT)
0.3	2019.09.16	Review and update	Emanuele Lettieri (POLIMI)
0.4	2019.09.20	Review and update	Ciprian Candea (ROPARDO)

Approvals

DATE	NAME	ORGANIZATION	ROLE
2019.08.09	Paolo Perego	POLIMI	Reviewer
2019.08.12	Ciprian Candea	ROPARDO	WP Leader



2019.08.27

Giuseppe Andreoni

POLIMI

Scientific Coordinator

Short Abstract

One of the NESTORE project's main goals is to create an ecosystem to give elderly people a coaching platform / system for guiding and optimizing their lifestyle. Since this ecosystem is mostly composed of ICT technologies, such as smartphone applications, social networks and games, the system integration plays a strategic role in the entire project.

The presented report will list the components integrated, the methods used for integration and the communication between the NESTORE components.

Key Words

Social Platform, Monitoring system, DSS, virtual coach, Shared Component, end-user, API, Sensors, IOT, Platform, Cloud, Applications



Table of Contents

1. Executive summary	6
2. Relation with other work packages	7
3. Integration layers	8
4. Shared modules.....	9
5. Device Management.....	13
5.1. Categories	14
5.2. Assign Devices to User	14
5.3. API operations.....	15
6. Other’s API’s.....	15
7. Server-Side applications	16
7.1. NESTORE Processing Unit.....	16
8. Annexes	18
8.1. Annex 1 - Ionic Proof-of-Concept	18
8.2. Annex 2 - Data Model definition for the Environmental IoT	21
8.3. ANNEX 3 - Sleep, social and weight Processing Unit.....	22
8.3.1. Smart scale	22
8.3.2. Sleep monitoring: the SCA11H Murata sensor	23
8.3.3. Sleep quality unit	24



Table of Figures

Figure 1 Graphical representation of the relationships between D 6.4 and tasks of NESTORE work packages..... 7

Figure 2 High Level NESTORE Cloud..... 9

Figure 3 IAM Main components 10

Figure 4 NESTORE Developer 10

Figure 5 NESTORE Developer UI 11

Figure 6 Module Settings 12

Figure 7 The process of gathering user data from third-party systems..... 12

Figure 8 Integration of smart scale 13

Figure 13 Example of defined categories..... 14

Figure 14 Manage User' Assets 15

Figure 15 Main blocks of Process Unit..... 17

Figure 16 Architecture of the smart scale data process 22

Figure 17 Environmental sensors: architecture of the data process 24

Glossary

API	Application Program Interface
DEU	Digital End User
GDPR	General Data Protection Regulation
IAM	Identity and Access Management
IoT	Internet of Things
JWT	JSON Web Token
MAC address	Media Access Control address
MQTT	Message Queuing Telemetry Transport



1. Executive summary

The D6.4 is represented by the NESTORE integrated system, meaning the Monitoring System, Decision Support System and Virtual Coach components. The NESTORE system can be tested using the link: <https://my.nestore-coach.eu/> and by installing on the smartphone the application. The website application from the link <https://my.nestore-coach.eu/> will be described in the deliverable D5.4 Social Platform. The access to the application is through login and password.

The presented report will list the components integrated, the methods used for integration and the communication between the NESTORE components.



2. Relation with other work packages

The integration activities started based on the Description of Action in M12 until M24 of the project. The input necessary for this process is represented by the tasks from WP2, WP3, WP4, WP5 and WP6 (see Fig 1) and contain details about protocols for the system validation in pilots, technical specifications for tangible interface, technical specification for serious games, technical specifications for social platform, details about coach interaction, details for NESTORE pathways, technical specifications for wearable sensors and for environmental sensors and technical specifications for DSS. The result of this task is represented by the NESTORE system and will be used in the task 6.7 – NESTORE Integrated Tests.

The main points needed to be achieved for the integration between components were:

- Management of users (API);
- Management of Assets (API);
- Data entry (data model, API for communication).

This report is organised in chapters as follows: chapter 3 contains the procedure and methods used to manage in the same way the end-users of the system; chapter 4 details the procedure for the management of the assets (devices); chapter 5 outline the methods of data integration (data model) inside the NESTORE system.

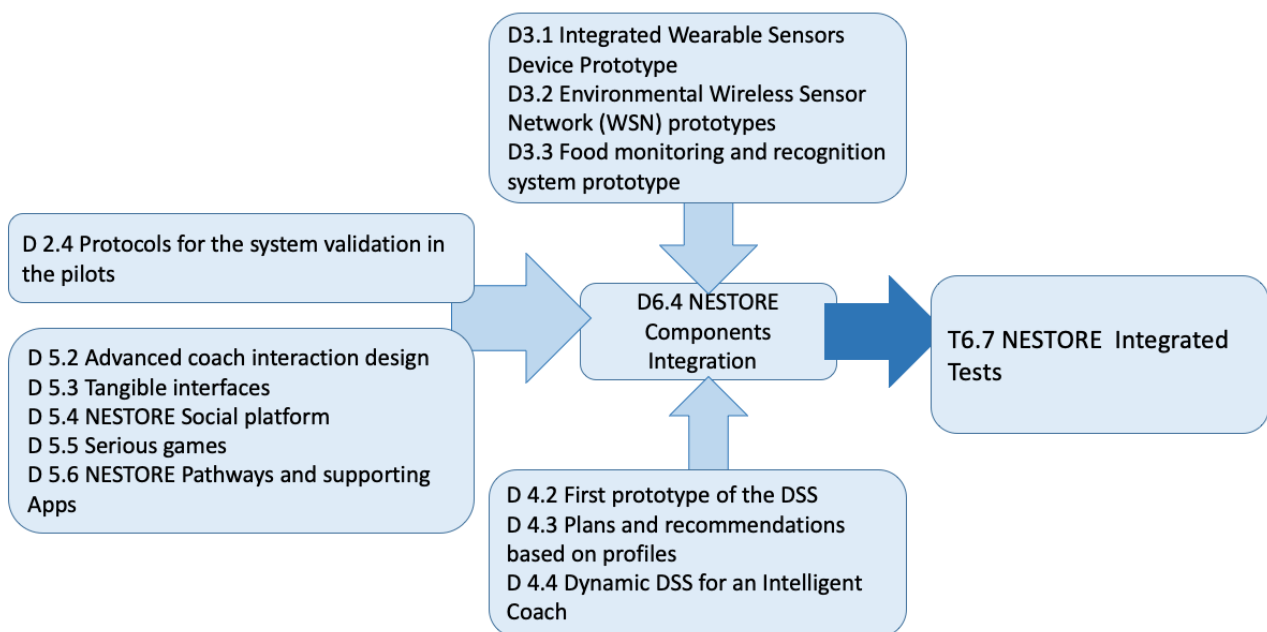


Figure 1 Graphical representation of the relationships between D 6.4 and tasks of NESTORE work packages



3. Integration layers

NESTORE platform integrate software modules that are created by different teams using appropriate databases and programming languages and techniques suitable to achieve their scope.

Then the inter-application communication / integration is the NESTORE platform cornerstone that is approached at different integration layers.

Data layer

Integration consists on moving data between different data sources: extracting data from its source, processing it conveniently and moving it to another data source. The main characteristics of this level are that it does not require many changes on applications and exchanging and transforming data is relatively less expensive than the other three levels. However, business logic is still enclosed in the primary application (the one that holds the initial data source) thus reducing real-time transactions. On the NESTORE platform modules that were developed tried to respect the Ontology guidelines defined on the WP2 and the UAAL models published at <https://git.nestore-coach.eu/uaal/ontology/>

Application interface layer

This integration level focuses on application interoperability through sharing of common business logic which is exposed by means of a predefined programming interface. It is usually implemented to expose the interface from packaged, or custom, applications to consume their services or retrieve their data.

Method layer

This level is called business integration level [3], this level comprises the business logic within the organization that applications may share, such as: services to access shared data, security, and underlying rules.

User interface layer

This level is dedicated to creating standardized interfaces (usually browser-based) to provide a single interface for a set of applications (legacy). Despite integrations at this level supposes highly coupling with applications, which also means higher maintenance costs, it is also the easiest integration to implement and has significant importance, since it ensures a consistent and effective user experience, especially with legacy applications

All NESTORE modules and applications expose integration APIs as Web services; Web services follows the guidelines provided by the integration platform (see D6.3.2).

The acronym API comes from Application Programming Interface. An API is a set of functions and procedures that fulfill one or many tasks for being used by other software. It allows to implement the functions and procedures that conform the API in another program without the need of programming them back. As RESTful systems usually communicate with the Hypertext Transfer Protocol (HTTP), a REST API is a library based completely on the HTTP(S) standard. It is used to add functionality to a software somebody already owns safely. The functionality of an API is usually limited by the developer so no more functionality can be added.

The communication between NESTORE components is made through representations of resources. For NESTORE API, the format of those representations were selected as JSON. JavaScript Object Notation (JSON) is a lightweight data-interchange format. It was derived from the ECMAScript Programming Language Standard (ECMA-404 2013; RFC 7159.2014).



Using a REST-based approach to implement a Web-service integration platform provide multiple advantages of the NESTORE platform :

- being lightweight: REST-based platforms built over HTTPS have almost all that is needed to process messaging between applications; no additional protocols nor toolkits are needed, thus improving efficiency and time spent developing application interfaces;
- being useful for fast deployments or first prototypes: creating and deploying REST-based Web services can be quickly and ready to be consumed; shared resources are available almost immediately and they can be easily consumed by any application
- interoperability: if applications are able to access a URL and understands the resource semantics, it does not matter in which platform or system they are deployed
- natural use of the Web: REST is completely embedded in the World Wide Web, thus, a REST-based platform over HTTPS provides the benefits of scalability, which means supporting many applications accessing services at the same time;
- interoperation and functionalities sharing of intelligent applications: more complex intelligent applications can be created, while keeping them decoupled, thus allowing them to evolve without negative effects on other applications.

4. Shared modules

Shared modules are represented by Identity Manager, Security Manager and Private Cloud that are used by NESTORE platform and its components.

4.1 Cloud Components

NESTORE Cloud architecture (Figure 2 High Level NESTORE Cloud) introduced on the D6.3.2 is addressing the deployment of all NESTORE services needed for End Users and all needed development tools.

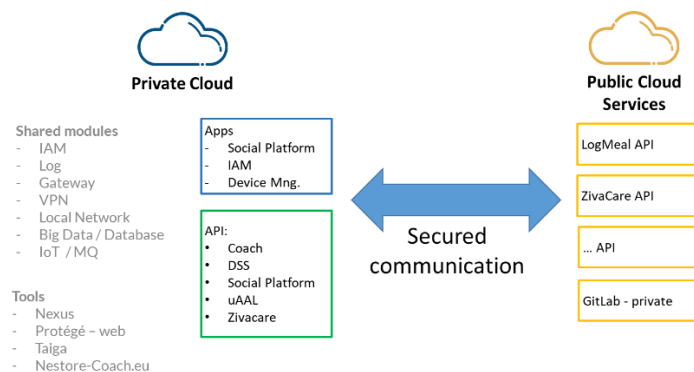


Figure 2 High Level NESTORE Cloud

Some of the NESTORE platform modules are not accessible outside of the Private Cloud because of the security reasons. For all modules that are exposing public user interfaces of public APIs these will be mentioned on the present document.

4.2 NESTORE Identity Manager



NESTORE Identity Manager Module is necessary for end user identities management across the platform components, APIs, the cloud, mobile, and sensors, regardless of the standards on which they are based.

A dedicated Identity and Access Management (IAM) system is set up to manage the End-Users’ credentials and to assure the GDPR compliancy relative to the end-users management. NESTORE IAM is based on the Key Cloak (<https://keycloak.org>) open-source solution and it integrate with NESTORE platform using API’s.

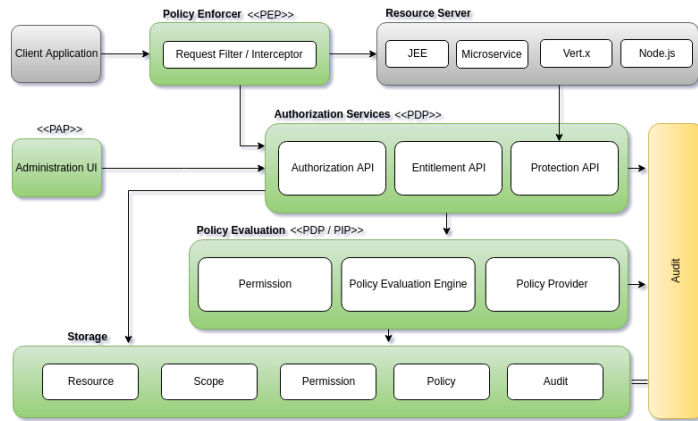


Figure 3 IAM Main components

4.3 NESTORE Developer application

The Developer application enables SaaS business model on top of different self-developed API resources, exposing API for integration with web, mobile and server-side applications.

A Multi-application / Multi-project approach can be easily handled with the Developer application. The Developer Platform (DP) handles authorization and user management for defined application. The developer platform REST API supports an easy integration with NESTORE applications – frontend and backend and enables a proprietary API for a SaaS business model.

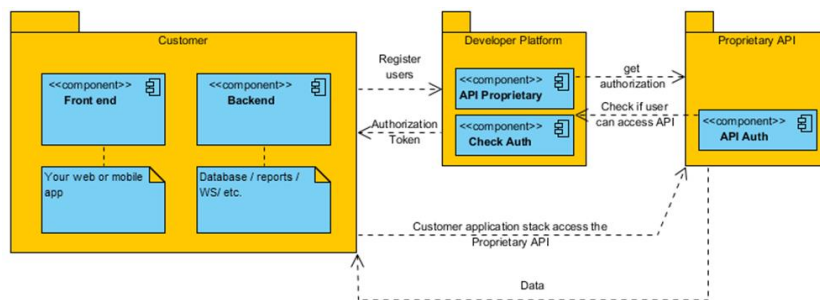


Figure 4 NESTORE Developer

For the different NESTORE API integration a dedicated Module is deployed that deliver the integration.

1. Creating an account on the developer platform

- a. Create Applications “realms”



- b. Assign Users for applications – each user receives an authorization code
 - c. Authorization source (optional): API's can implement a dedicated authorization mechanism where the developer platform verifies the authorization for a specific user
2. From NESTORE applications users are authorized to access data from the API's.
 3. APIs check if a user has the authorization and if server data is authentic (optional – in case it is used through a trusted connection / trusted network, it is not needed)

All the public web services of the NESTORE platform are exposed using the Developer application and for present time other interested parties can create an account and try the public NESTORE API'S.

Accessible at: <https://developer.nestore-coach.eu>

In figure 5, a sample of the Developer application UI show how different “realms” can be defined – point 1.a), here we have realm 1 – Nestore End Users and realm 2 – Nestore Dev.

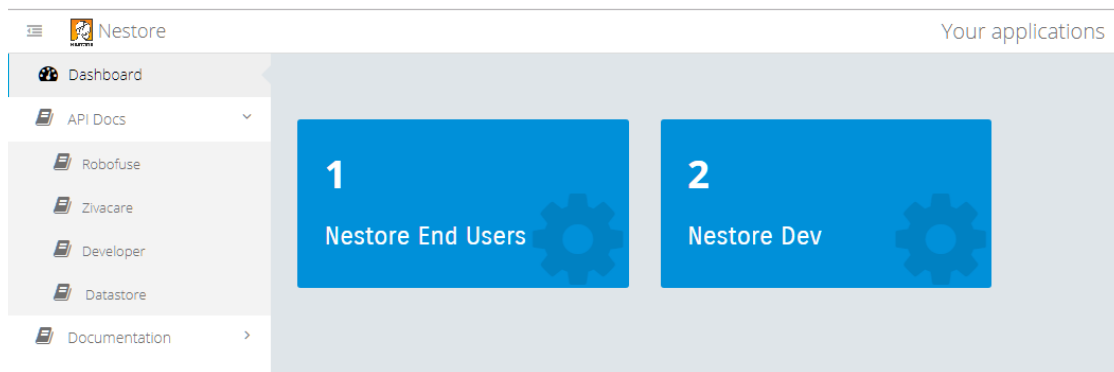


Figure 5 NESTORE Developer UI

Data that are blinded with realm 2 Nestore Dev are not visible to the realm 1 Nestore End Users.

For example on the realm 2 Nestore Dev all tests are conducted.

On the figure 6 is visible how for each “realm” different APIs are enabled or not, many other administrative task can be accomplished easily from the Developer application.

The Developer application is intended to be used only by the Administrator role and is not intended to be used by End User or non technical staff.



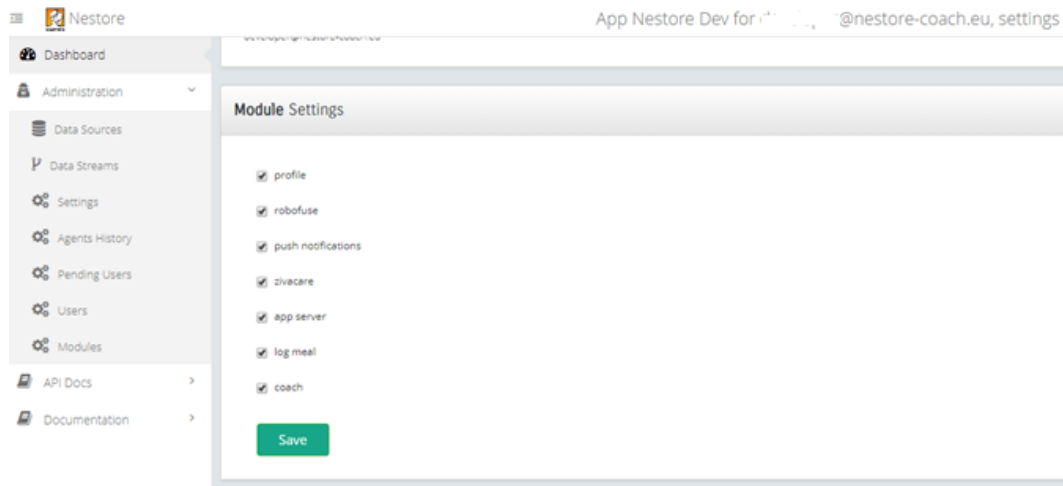


Figure 6 Module Settings

4.4 Third party data providers

The NESTORE system will include the ZivaCareAPI service, which will act as a hub for NESTORE End Users to register and authorize End User accounts from 3rd party providers (i.e. Withings) as was introduced on the D6.3.2. Once the user authorizes and registers their account and sources, NESTORE would have access to the data from 3rd party backend services. For integrating this kind of data into the NESTORE cloud, we will use ZivaCare to interface.

The use of the ZivaCare system in this scenario is the one presented in Figure 7 The process of gathering user data from third-party systems.

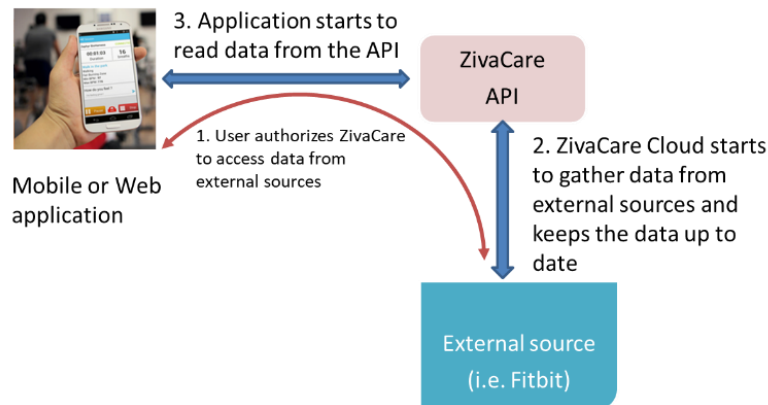


Figure 7 The process of gathering user data from third-party systems

The process starts when a NESTORE End User gives authorization to the ZivaCare system to a third-party health data source. Existing user health data is automatically gathered from third-party sensors or devices, through a synchronization mechanism. Once this data is available in the NESTORE Cloud, NESTORE web and mobile applications can use it to display to the user different statistics. Obviously, the data will be secured and accessible only to the owner. Just the owner decides who can have access to her/his data.

As part of the ZivaCare system, **SyncEngine** is responsible with the synchronization of a user’s data between ZivaCare and various third-party data providers. The synchronization is done by invoking agents (which are



provider specific), using a RESTful Web Service interface, based on a specific and configurable schedule. Details about this are presented in the Annex “Zivacare”.

Accessible at: https://api.zivacare.com/v3?access_token=demo

On similar process like the one presented on 4.4.1 different third party services can be integrated. The integrated services where detailed on the D6.3.2, here we mention one integration on 4.4.1.

4.4.1 Withings integration

As shown in Figure 8, the process starts through a first pairing between the NB+ and a smartphone/tablet. Successively, the sensor is able to send data to the Nokia Health cloud using API services (Withings Health API references are available at <http://developer.health.nokia.com/oauth2/>). Then the Zivacare system is authorized to pull the end user data from the Withings cloud and insert on the ZivacareAPI from where the other NESTORE components will use it.

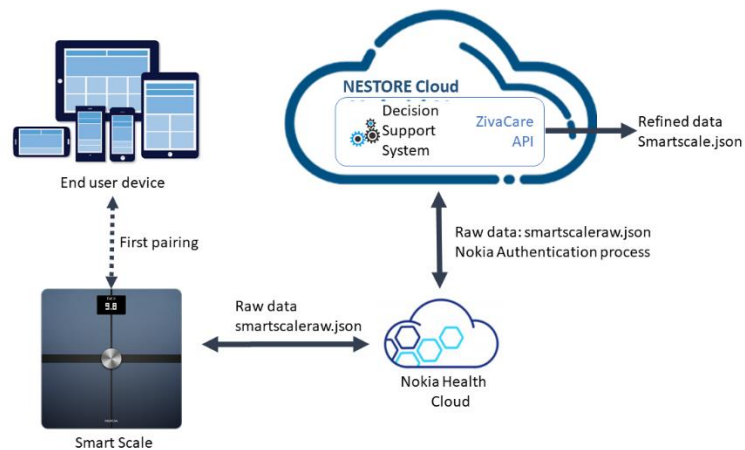


Figure 8 Integration of smart scale

The Withings Health platform requires OAuth2.0 in order to allow data exchange between software applications and users’ device.

5. Device Management

On NESTORE platform different devices need to be integrated and need to send data to it.

Specifically on NESTORE are integrated next devices:

- Wristband
- Social and Environmental Beacons
- Sleep monitoring
- Weight Balance



On developer.nestore-coach.eu account, on the application created where users and devices will be managed must be registered the devices UID.

The IoT security is assured on different levels, one level is pre-registered devices in the developer account. This task can be managed using the Asset option from your application Robofuse dashboard. When a device is defined in the system the Asset ID must be filled and other parameters can be defined.

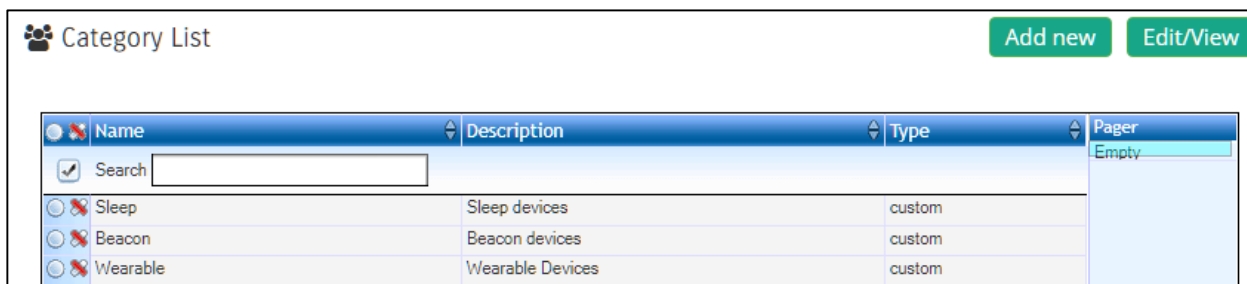
Once the Device is registered using the Asset ID from that device communication, it will be accepted by the IoT Cloud. The IoT Cloud will check if that Asset ID is or not registered on the application/Robofuse, if the Asset ID is defined, it will check the data set format against the JSON Schema if all ok, data are stored on the IoT datastore.

Asset ID must be unique in your dataset, can be generated as custom ID, or a UID. Devices can be registered using API calls.

5.1. Categories

On the categories option from Dashboard can be defined different categories that latter can be assigned to each Device. This can support a better device management – around categories.

Example of defined categories:



Name	Description	Type	Pager
Search <input type="text"/>			Empty
<input type="radio"/> Sleep	Sleep devices	custom	
<input type="radio"/> Beacon	Beacon devices	custom	
<input type="radio"/> Wearable	Wearable Devices	custom	

Figure 9 Example of defined categories

5.2. Assign Devices to User

The most important aspect is to define what devices are used by the end user. Each end user can register different devices from the device list that is already defined in the system.

The entire process is based on the next concept: A specific end-user is “Take Over” a specific Device (Asset ID) and at some point can decide to “Return it”. During the “Ownership” period (Take Over date until Return Date) all device generated data are owned by the specific end-user.

A device that is not already assigned can be assigned to a specific user, for the moment the simultaneous usage of same device by multiple user is not supported.

This operation can be manually handled using the option Users from the Application / Robofuse dashboard. Select Manage Data for a specific user and then select a device from the available devices and Add to user. Starting from that moment all data that are sent by device to the IoT datastore, will be registered on behalf of that specific user.



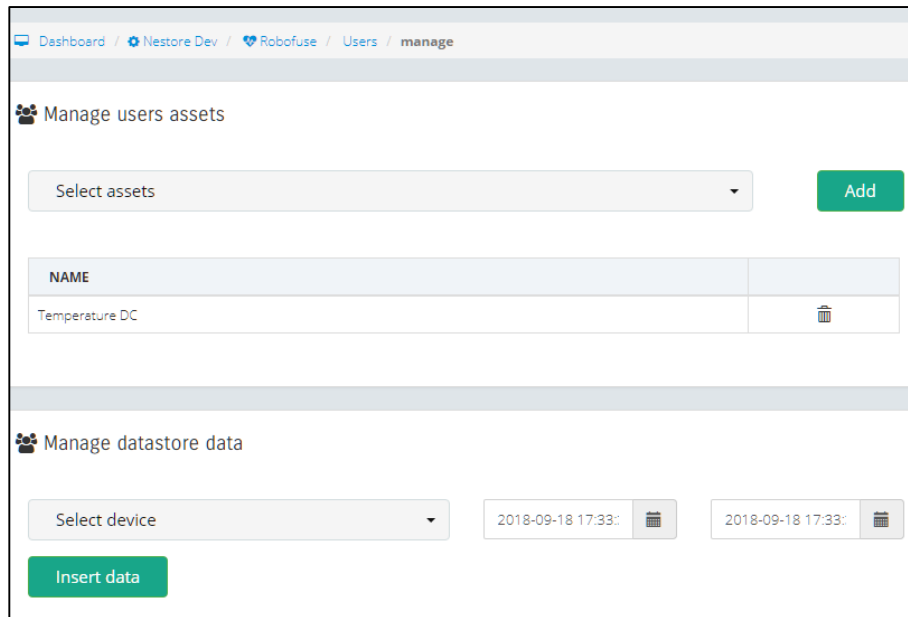


Figure 10 Manage User' Assets

In case that it's needed to reuse the device once that specific user doesn't need it, must delete the device from first user and then go to the new user and register the device to a new user. Once that device is register to a new user all the data sent to the IoT datastore will be assigned to the new user. All data stored for previous user are still assigned to his account.

5.3.API operations

All operations that allow to register/unregister devices to users are available using Robofuse API. For all API operations on this section before must be obtain a valid Access Token – for more references please see Developer API.

Access at: <https://api.robofuse.com/>

6. Other's API's

Under the NESTORE platform a set of different API'S are deployed and used by different components. Following is presented a short list with the public API'S.

- <https://api.robofuse.com/>
- <https://datastore.robofuse.com/>
- <https://developer.nestore-coach.eu>
- https://api.zivacare.com/v3?access_token=demo
- <https://api.coach.zivacare.com/>
- <http://www.logmeal.ml>



- <http://profile.nestore-coach.eu>

7. Server-Side applications

To implement different features of the system, a set of functions must be executed on the back end / server side. There are 2 main categories of those. The first one is triggered by an end-user interaction with the system, that involves a user that is logged in accordingly. A second one is when a back-end system is running based on this one scheduler and must process different pieces of data.

We can label them as:

- API's – first one – an end-user action will trigger one or more API's
- Server Side - second one;

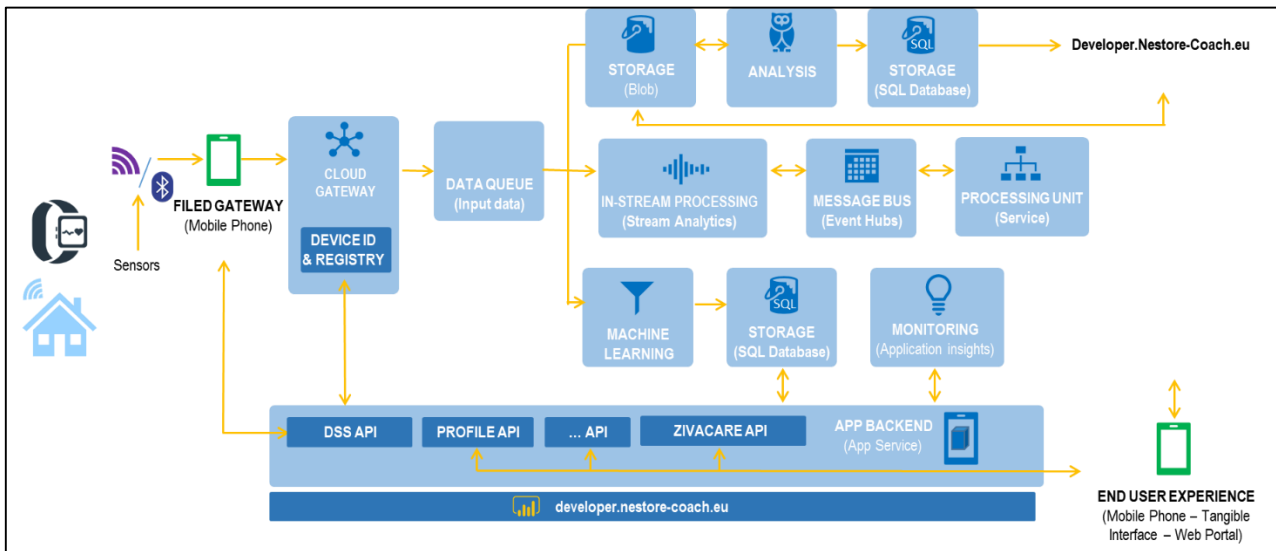
Server side application is supposed to be deployed on a secure network, encrypted data, etc. and in this case the application can use also the Client Secret ID's. With this other, private data can be accessed.

7.1.NESTORE Processing Unit

On the NESTORE platform, once the data are in the cloud, different processing units can do computation and distribute results to other systems.

In this section, the processing unit concept is presented.

The processing unit represents a sub-system that is dealing with specific computing, it can implement his own database, system architecture, etc. Data that are consumed are Platform data, and such unit can communicate with other sub-systems at any time. We can assume that a processing unit is not directly connected to the public Internet and is deployed on the back end private cloud.



Example: Sleep Monitoring can be represented as a Processing unit because it needs to store locally raw data that are extracted from the physical devices for each user, it must apply different methods/algorithms and the results are stored/published on cloud.

For different specific tasks, different processing units can be deployed on the NESTORE Platform.



It is not an imposed architecture, so, each unit can implement a different specific architecture. The main block of the unit is presented below.

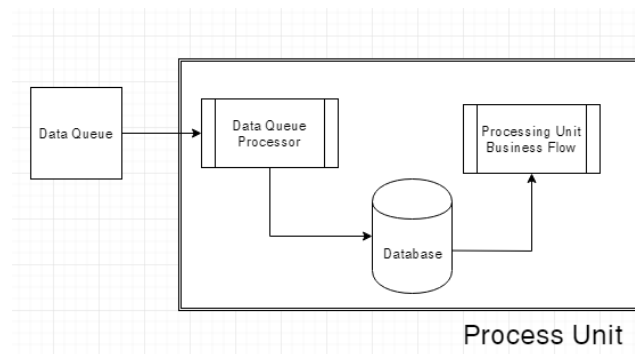


Figure 11 Main blocks of Process Unit

To get data from the NESTORE Platform it is necessary to use specific connectors as well as API's. The architecture of the Data Queue Processor is composed by 3 main blocks:

- Queue manager – it is encoding the MQTT logic, that allows accessing the Platform Data Queue;
- Platform context – it is the logic that allows obtaining the end-user tokens that are needed latter to access different platform blocks;
- Process Data & Store in local database – If the Unit needs to store data for a certain period can implement the logic here.

To access the message bus a local subscriber that is preparing data for the Processing Unit is needed. A complete example can be found on <https://git.nestore-coach.eu> on private site.

Processing Unit can have a proprietary database where to keep for a specific time data needed for processing, i.e., computation stages, raw data, parameters, etc. Recommendation is to remove the local data after these are not necessary for the processing scope.



8. Annexes

8.1. Annex 1 - Ionic Proof-of-Concept

To push end-user functions where realized an Ionic app that uses the RoboFuse API.

NESTORE platform is used for login / sign up

The user must log in with NESTORE credentials:

- enters user and password
- clicks login bar on the bottom of the screen

The screenshot shows a login form with the following elements:

- Header: iam.nestore-co...
- Email input field: mp@nestore-coach.eu
- Password input field: masked with dots
- Links: [Sign up](#) and [Forgot Password](#)
- Bottom bar: Login

End-user Devices

The screenshot shows the 'Devices' section with the following elements:

- Header: Devices
- Section: Register a new device
- Text: Enter below the Device ID to add a ne...
- Input field: m0010 with a checkmark button
- List of devices:
 - motion0010 (with phone icon)
 - test4 (with location pin icon)
 - voltage0010 (with phone icon)
 - test3 (with location pin icon)
- Bottom navigation bar: Dashboard, Devices (selected), Account

To manage devices end-user must navigate to Device section.

To add a new device, he must:

- write the Internal Asset Number
- touch the button.

The device will be added to the list below.

If the device is already assigned, the application will return an error:

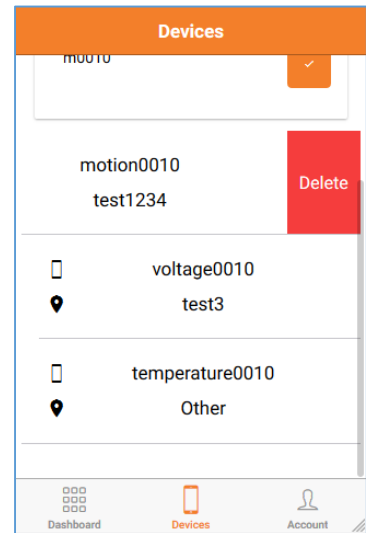
The error message dialog box contains the following text:

Could not assign asset!
The asset is already taken.

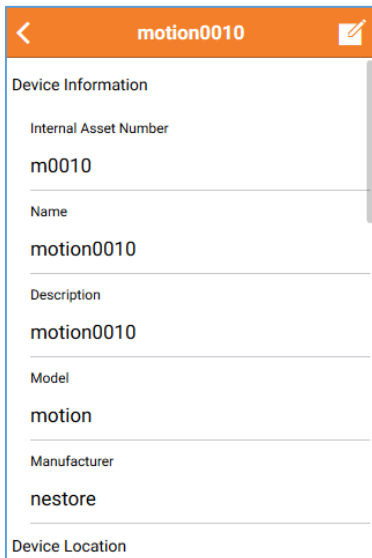
Ok




To remove a device, he must drag the device info to the left until a delete control appears, and touch it. The device will be removed from the list below.

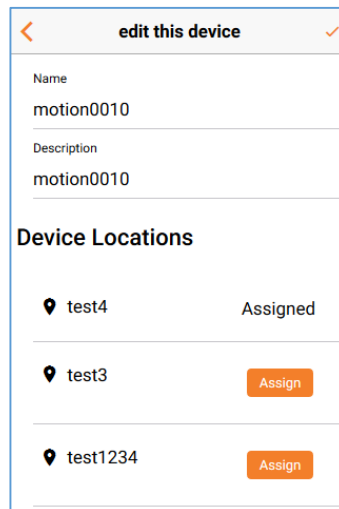


Edit device info




To edit a device, the user must touch the area of the respective device in the list, which will display the device's details, and click the button: 

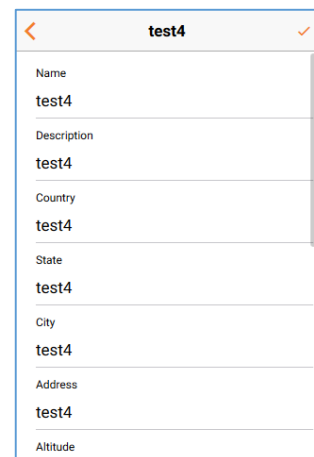
The application will enter the edit device mode, where the user can change the name and the description of the device as well as its locations.

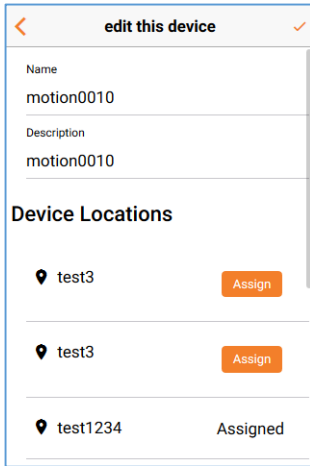


Add Edit Location info

To change location information, must touch the location's area. changing values, he must touch button. When saved, the list will the assigned devices

the user
After
the 
go back to

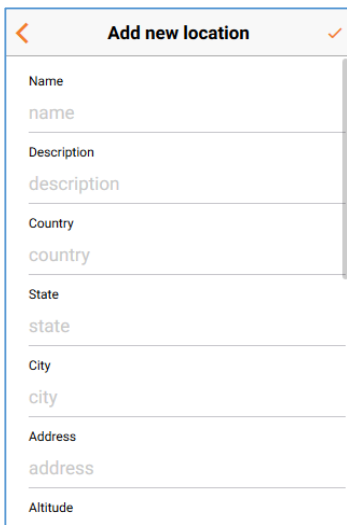
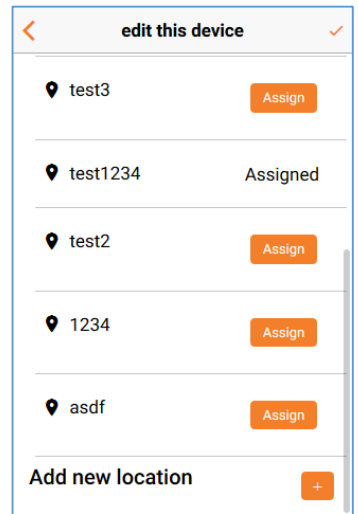




To assign other locations to the device, the user must touch the **Assign** button in line with the unassigned device.

The device will become assigned, and the previously assigned becomes unassigned.

To add a new location to choose from into the devices list, the user must scroll to the bottom and hit the + button.



The application will open a blank form to populate with location info.



8.2. Annex 2 - Data Model definition for the Environmental IoT

BLE beacon advertisements will be collected by the wristband and sent to a mobile device running a dedicated agent. Each related message sent by the agent will have the following data format:

```
{
  "beaconUUID": "b23143fe-ec81-47a8-9c3c-200000000003",
  "timestamp": 1531481898832,
  "temperature": 25.1,
  "humidity": 13.5,
  "accelerometer": true,
  "battery": 95.2,
  "rssi": -82
}
```

BLE advertisements will be of two types: mobile (coming from people wearing them for social interaction) or fixed (coming from BLE beacons deployed in the house). In the case of mobile BLE advertisements, temperature, humidity, and accelerometer field will be null.

The advertisement gathering process can vary depending on the type of deployment. We can have indeed two scenarios:

Scenario 1:

The wristband/agent knows which are the uuids given to the particular user and filters advertisements to be collected (in the case of wristband knowing the uuids associated to the user) or the advertisement to be sent (in the case of agent knowing the uuids).

Scenario 2:

Both wristband and agent do not know the associations between uuids and user collecting and sending all the advertisements (please note that BLE advertisements can be sent by devices present in the house out of the NESTORE scope). In this case, there will be a back-end service filtering the advertisements out of the NESTORE scope.



8.3.ANNEX 3 - Sleep, social and weight Processing Unit

8.3.1. Smart scale

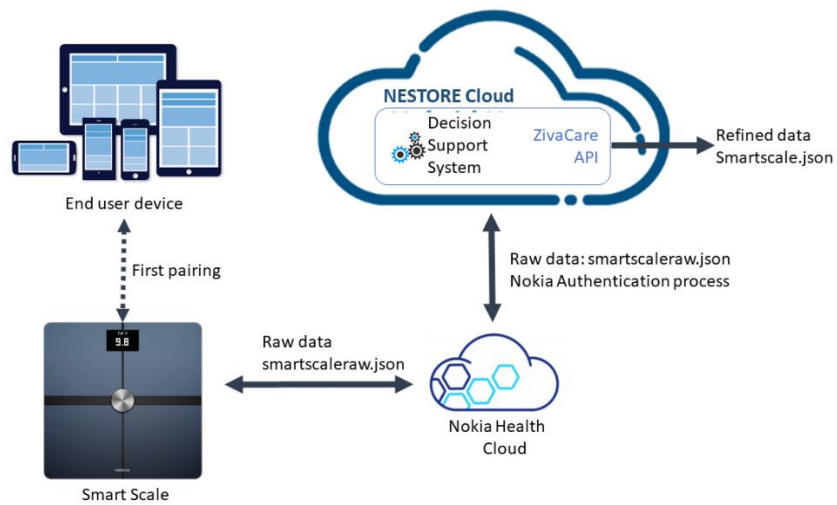


Figure 12 Architecture of the smart scale data process

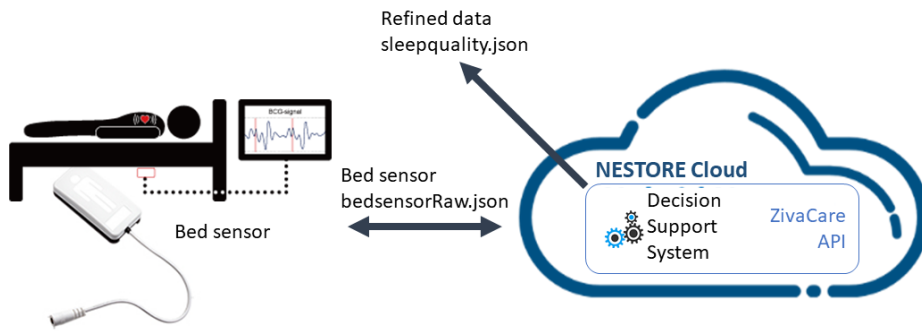
Smartscaleraw.json contains:

```
{
  "scaleID": "xxxxxxx-xxxx-xxxx-xxxx-",
  "timestamp": 1531481898832,
  "weight": 75.88,
  "fatMass": 20,
  "muscleMass": 40,
  "water": 55,
  "boneMass": 30,
  "BMI": 20
}
```

Smartscale.json contains the information in smartscaleraw.json and the information required by the zivacare system.



8.3.2. Sleep monitoring: the SCA11H Murata sensor



Architecture of the sleep monitoring data process

The message body sent by the SCA11H has the following format:

```
<Data version="1.7" xmlns="urn:wsn-openapi:sidf">
<Network id="Network ID">
<Node id="Node ID">
<Sensor id="0">
<Measurement quantity="BioSignal" time="Time">
<Component id="heart rate" unit="bpm" />
<Component id="respiration rate" unit="rpm"/>
<Component id="relative stroke volume" unit="µl"/>
<Component id="heart rate variability" unit="ms"/>
<Component id="measured signal strength"/>
<Component id="status"/>
<Component id="beat-to-beat time" unit="ms"/>
<Component id="beat-to-beat time -1" unit="ms"/>
<Component id="beat-to-beat time -2" unit="ms"/>
<Values tick="sec">
Timestamp, HR, RR, SV, HRV, SS, Status, B2B,
B2B', B2B''
...
</Values>
</Measurement>
</Sensor>
</Node>
</Network>
</Data>
```

BLE Beacon tags: air quality, socialization, and sedentariness detection



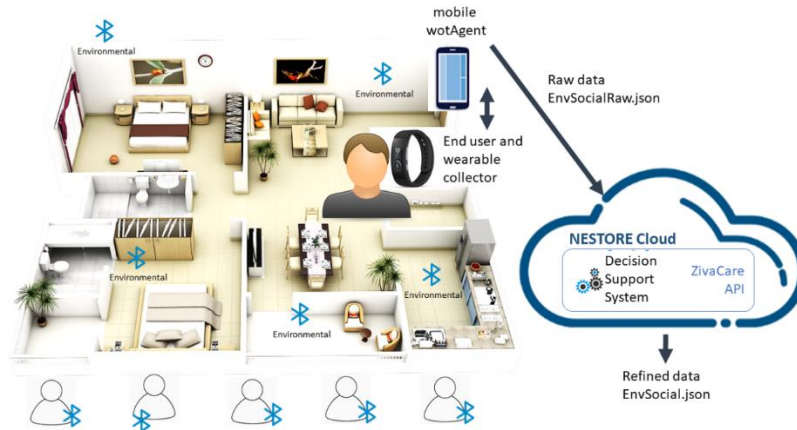


Figure 13 Environmental sensors: architecture of the data process

Envsocialraw.json contains:

```
{
  "beaconUUID": "b23143fe-ec81-47a8-9c3c-200000000003",
  "timestamp": 1531481898832,
  "temperature": 25.1,
  "humidity": 13.5,
  "accelerometer": true,
  "battery": 95.2,
  "rssi": -82
}
```

8.3.3. Sleep quality unit

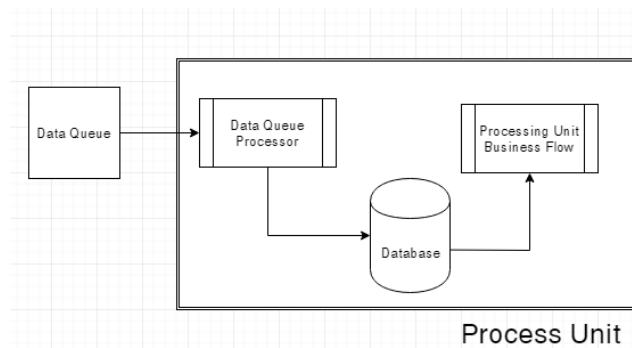
Processing unit to analyses the end-user Sleep Quality is detailed – (SQU – Sleep Quality Unit).

The SQU processing unit is not directly connected to the public Internet and is deployed on the back end private cloud.

The processing unit store locally raw data acquired by the muRata devices for each user, it applies different methods/algorithms and the results are stored/published on cloud.

To pair device data with end-user data, assure that the device is assigned to an end-user – this step can be done by the end-user from web or mobile application.

Generic block architecture is implemented to serve the SQU needs, a local database will store raw data after the Data Queue Processor is pairing device data with end-user data.



The architecture of the Data Queue Processor is assuring that data are correctly paired with end-user specific details.



To access the message bus it is needed a local subscriber that is preparing data for the Processing Unit. The Data Queue Processor flow:

- Connecting to the MQTT – configuration needed: what queue to subscriber, IP, etc.
- When messages are on queue execute the processing
 - Read message
 - Check if the message is valid – if valid xml
 - Extract the node-id (Asset ID)
 - Validate internal number – optional - /assets/{internal_asset_number}/validate
 - Prepare end-user context
 - Call /internal/user/asset/{internal_asset_number} Get User data by internal asset number
 - If user context valid
 - Insert data into the database

SQU maintains the internal database where are stored the valid messages stored by the Queue Processor.

For an easy future processing the messages are linked on the local database with the end-user details – tokens.

2 tables in raw_data schema:

- murata_device - each device and the associated developer id and token
- murata_biosignal - 1 row for each second / device / developer

