Multi-layered Security Technologies

for hyper-connected smart cities

D4.8 Application Security

March 2021

Grant Agreement No. 814917

Multi-layered Security technologies to ensure hyper-connected smart cities with Blockchain, BigData, Cloud and IoT

Project acronym	M-Sec
Deliverable	D4.8 Application security
Work Package	WP4
Submission date	31 March 2021
Deliverable lead	Georgios Palaiokrassas (ICCS)
Authors	Georgios Palaiokrassas (ICCS), Kenji Tei (WU), Nobukazu Yoshioka (NII), Takafumi Komoto (NII), Orfefs Voutyras (ICCS)
Internal reviewer	Mathieu Gallissot
Dissemination Level	Public
Type of deliverable	DEM

The M-Sec project is jointly funded by the European Union's Horizon 2020 research and innovation programme (contract No 814917) and by the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN (contract No. 19501).

Version history

#	Date	Authors (Organization)	Changes
v0.1	01 February 2021	Georgios Palaiokrassas (ICCS)	Full ToC and assignments
v0.2	02 March 2021	Kenji Tei (WU), Takafumi Komoto (NII), Nobukazu Yoshioka (NII)	Initial inputs to Development & Designing Tools FG
v0.3	16 March 2021	Kenji Tei (WU), Takafumi Komoto (NII), Nobukazu Yoshioka (NII)	Updated Section 3
v0.4	17 March 2021	Georgios Palaiokrassas (ICCS)	IoT Marketplace FG contribution
v0.5	18 March 2021	Kenji Tei (WU)	Updated Section 3
v0.6	25 March 2021	Georgios Palaiokrassas (ICCS)	APIs & Integrations
v0.7	27 March 2021	Georgios Palaiokrassas (ICCS)	Additions, corrections
v0.8	29 March 2021	Orfefs Voutyras (ICCS)	Architecture diagrams provided, review
v0.9	30 March 2021	Mathieu Gallissot (CEA)	Internal Review
V0.10	30 March 2021	Georgios Palaiokrassas (ICCS)	Version ready for submission

Table of Contents

List of Tables

List of Figures

Glossary

Acronym	Description
API	Application Programming Interface
SAT	Security Analysis Tool
CCD	Companion Database
MTSA	Model System Transition Analyzer
Dx.y	Deliverable y of WP x
FG	Functional Group
Tx.y	Task y of WP x
P2P	Peer-to-peer
WP	Work Package
UC	Use Case
JSON	JavaScript Object Notation

Executive Summary

The work described in this deliverable (D4.8) was carried out in the framework of WP4 – "Multi-layered Security Technologies", and more specifically, in the framework of T4.4 – "Application Level Security". The report presents the updated and final version of the document (the first version being D4.7), providing the technical details of the Functional Group and Functional Components related to the Task.

All technical partners involved in this task collaborated and developed the appropriate tools to meet the objectives set out in the project, especially with regard to novel Security aspects in IoT contexts. Every partner focuses on the individual modules that they are responsible for during the implementation phase of WP4 and supports the integration activities of WP2 while following the common Architecture framework set by WP3 in D3.4.

All of the updated versions of the WP4 technical deliverables (D4.2, D4.4, D4.6, D4.8, D4.10) follow the same approach and have the same structure. Section 1 provides an introduction to the scope of this document and its relation with other WPs and Tasks. Sections 2 and 3, which aggregate all the main outcomes of the Task, present the FG and the Functional Components covered by the Task, by providing an extensive description of the corresponding functionalities, and details related to the API of the FGs and their interactions with other FGs of the M-Sec solution. Finally, Section 4 concludes the document.

Regarding the differences between 'D4.7 M-Sec Application Level Security – first version' and 'D4.8 M-Sec Application Level Security – final version':

- **Section 1** has remained more or less the same but includes an extra subsection identifying the M-Sec Risks linked to this specific Task.
- **Sections 2 and 3** as a whole provides a more integrated view of the Components, as it focuses on their presentation from an FG perspective. Components have been moved from/to other deliverables accordingly.
- **Section 4** corresponds to Section 4 of the previous version of the document.

All in all, the deliverable is considered to have provided all of the information required to expose the M-Sec technical solutions related to T4.4 as well as the results of the integration and demonstration-related activities.

1. Introduction

1.1 Scope of the document

The main focus of this task is to establish engineering foundations to support the development of secure smart city applications on the top of M-Sec platform. Security requirements for smart city applications should be elicited by identifying security goals, assets to be protected, and threats. In addition, protection mechanisms mitigating the threats should be designed and implemented in applications. M-Sec provides methodologies and tools to develop smart city applications in order to support developers of smart city applications

Following the final version of the architecture presented in Deliverable 3.4, the components about to be discussed in this report are part of the so-called IoT Marketplace Functional Group (FG) and the Development & Security Designing Tools FG.

All in all, this task has as its main objective the definition and ulterior implementation of the M-Sec application security layer and thus starts with the services it comprises, which have evolved from the initial description in Deliverable 4.7, in parallel to the execution of Stage 1 of the different pilots.

This document addresses the main objectives of this task establishing the M-Sec components strengthening the application layer, which will become one of the security layers in the overall Multi-layer Security (M-Sec) platform, providing the needed security and reliability for smart city applications.

1.2 Relation to other work packages and tasks

The following figure summarises the relations of this deliverable (and the corresponding task) to other tasks and WPs.

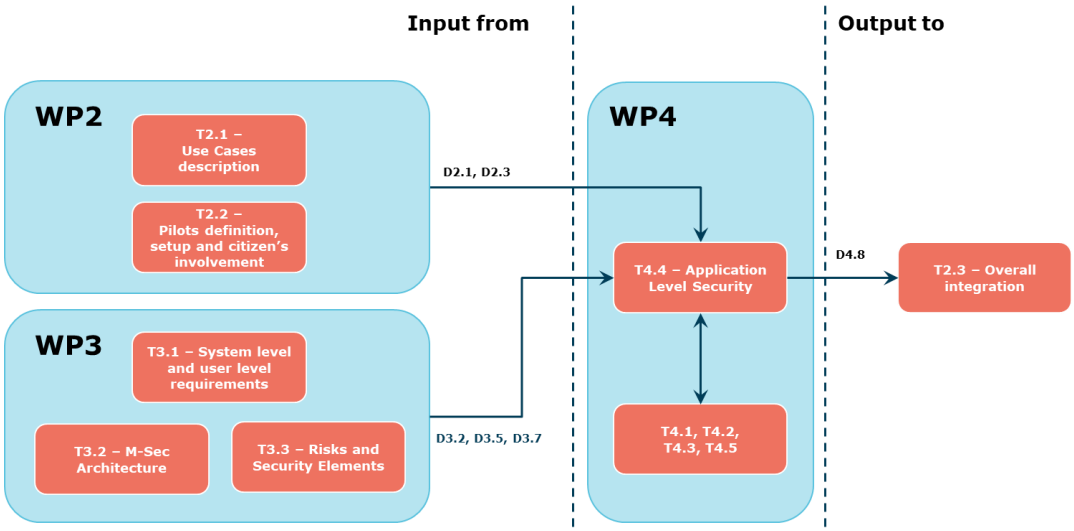

Figure 1. T4.4 and D4.8 relation to other WPs and Tasks

The work done in Task 4.4 is directly related to WP3. T4.4 receives input system and user requirements from T3.1 and Risks- and Threats-related information from T3.3. Moreover, it follows the common Architectural framework that has been identified in T3.2 for the coordination of all the technical activities. Similarly, the Task receives input from WP2 related to the coverage of the needs of the UCs and the pilots.

Within this very same work package, Task 4.4 is related to Task 4.1 and Task 4.2, where the IoT security layer and cloud/data security layer are discussed, respectively. Smart city applications are designed and implemented by using API provided by those layers. Besides, it is also related to Task 4.3 and the relation to all the other WP4 tasks is documented with the direct or indirect integration with other FGs and their components.

Finally, the results of this report are directly provided as input to T2.3 which is focusing on the overall integration activities. Together with the other final deliverables of WP4, D4.2 provides all the information and functionalities required for an integrated security solution.

1.3 Relation to M-Sec Risks

The complete list of potential risks and threats that may affect M-Sec's IoT layer can be checked in Table 1, as extracted from Task 3.3.

All of these threats are of Type "Cloud", and Sub-Type "Data Access", "Storage" or "Management". Specific interfaces are provided in D3.5.

Table 1: M-Sec T4.4 risks and threats

Threat #	Description	STRIDE Threat Class	M-Sec Asset	Source	Probabi-lity	Critica-lity	Ra-ting	Comments/ Mitigation
Thr.App.1	Libraries and modules on which the application is reliant, can be compromised or replaced by malicious versions. (they can be affected by the same threats as the application itself)	S, D, T		All Use Cases	1	3	3	Vulnerability Assessment
Thr.App.2	Other malicious agents can issue requests and data on behalf of the application.	S (e.g. IP Spoofing)	Connected Care	Use Case 2	3	5	15	Companion DB may mitigate some of the risks; the application will not know the keys, only the user will know it. Authentication mechanism.
Thr.App.3	Malicious agents may have read access to the data the application is processing, and results.	S, I	Park Guide, Connected Care, Smile City Report	Use Case 1, Use Case 2	3	5	15	Companion DB may mitigate some of the risks. Only authorized agents can access M-Sec sensitive data. It is encrypted/decrypted through the Companion DB. Authentication mechanism.
Thr.App.4	Malicious agents may have write access to the data the application is processing. Being able to change it and produce unpredicted states	T	Connected Care	Use Case 2	3	5	15	With the Companion DB, only authorized agents can access M-Sec sensitive data. It is encrypted/decrypted through the Companion DB. Authentication mechanism.

Thr.App.5	Data sources may be replaced, feeding erroneous or malicious data into the system workflow. E.g: Buffer overflow; cross-site scripting; SQL injection; canonicalization	E, T	Connected Care, Smile City Report	Use Case 2	1	3	3	With the Companion Database, the data is encrypted and linked to the blockchain, so it cannot be tampering. Authentication mechanism.
Thr.App.6	Compiled, binaries or bytecode of the application may be corrupted or maliciously altered for execution.	T		All Use Cases	3	5	15	Memory Protection
Thr.App.7	Legit requests may have undesirable effects.	T, D		All Use Cases	3	5	15	Vulnerability Assessment
Thr.App.8	The user may be convinced to perform actions that expose their data, or the application workflow (Social Engineering)	R		All Use Cases	1	3	3	Security Learning
Thr.App.9	Stored Data may be compromised. Either the cryptographic keys are not secure enough; the algorithms, the storage container is compromised or there might be some issue in the whole workflow.	T	Park Guide, Connected Care, Smile City Report	Use Case 1, Use Case 2	1	5	5	With the Companion Database, the storage of the sensitive dat is in a different database, so they should compromise at least the two databases. Vulnerability Assessment
Thr.App.10	The user account is compromised. Either because the user has released, forgot, or shared her/his credentials, or because the account is meant to be shared amongst several users.	S	Park Guide, Smile City Report	All Use Cases	3	5	15	Log Mechanism

Thr.App.11	The application may be compromised, because there is some extreme cases that are not considered, or certain assumptions make it susceptible to get to unstable states	I	MTSA	All Use Cases	1	5	5	Vulnerability Assessment
Thr.App.12	The application (or platform) does not provide log of the transactions and/or execution trace. Leaving potential attacks un accounted.	R	Park Guide, Connected Care, Smile City Report	Use Case 1, Use Case 2	1	5	5	The Companion DB provides some logs of interactions, but it is not its main purpose. Vulnerability assessment.
Thr.App.13	The application uses un registered communications (not known to the underlying platform) or without relation to the functioning of the app itself.	E		All Use Cases	1	5	5	Vulnerability Assessment
Thr.App.14	The application does not use the appropriate authorization mechanisms, or these mechanisms can be easily circumvented	S	Connected Care	Use Case 2	1	5	5	In the backend site, the Companion DB uses a smart contract in order to grant access to the sensitive data. Vulnerability assessment.
Thr.App.15	The application does not use the appropriate authentication mechanisms, or these mechanisms can be compromised (e.g.: key logger, un secured password storage or transmission, etc.)	S		All Use Cases	1	5	5	Vulnerability Assessment
Thr.App.16	Vulnerabilities-flaws in smart contracts	T,R,I,D	Blockchain app / Smart contract	All Use Cases	3	5	15	Flaws in smart contracts can cause unforeseen security breaches. Thorough lab testing before going into

								production. Continous code review.
Thr.App. 17	Under-optimized smart contracts	T,R,I,D	Blockchain app / Smart contract	All Use Cases	3	3	9	Dead code, loop fusion, repeated computation can cause denial of service on the long run. Thorough lab testing before going into production. Continous code review.
Thr.App. 18	Transaction privacy leakage	T,R,I,D	Blockchain app / Smart contract	All Use Cases	3	3	9	Once identity is revealed the whole history of transactions is exposed. Thorough lab testing before going into production. Continous code review.
Thr.App. 19	Misunderstanding of the agreement of applications.	I	All apps	All Use Cases	1	3	3	Agreement should be easy and clear to understand.
Thr.App. 20	Personal information and facial images are mistakenly uploaded in the marketplace and traded.	I	IoT Marketplace	Use case 5	3	5	15	Need a check mechanism
Thr.App. 21	Malicious agents may make fake transactions.	I	IoT Marketplace	Use case 5	3	5	15	Ensure by blockchain technology
Thr.App. 22	Malicious agents may upload face data.	I	IoT Marketplace	Use case 5	3	5	15	Ensure by blockchain technology

2. IoT Marketplace FG

2.1 General Description of the FG

The IoT Marketplace FG consists of two main components: the IoT Marketplace and the Mobile Wallet. These components are integrated with other FGs and directly or indirectly communicate with components from other FGs. For the scope of this technical document, only the core M-Sec components are discussed, and as such, the main focus will be on the IoT Marketplace.

In the following section, the components of the FG are described in detail. Section 2.3 presents the interactions of these components with components of other M-Sec FGs. Finally, the Annex presents the position of the FG within the whole M-Sec Architecture.

2.2 Components of the FG

IoT Marketplace

The goal is to create decentralized IoT ecosystems and validate their viability and sustainability. In this direction, we define and implement a novel marketplace where smart objects can exchange information, energy, and services through the use of virtual currencies allowing the real-time matching of supply and demand enabling the creation of liquid markets with profitable business models of the IoT stakeholders. In this section, we cover the basic technical implementation details of the M-Sec marketplace: market participants, from IoT devices to humans using mobile applications are able to exchange data and value through the M-Sec blockchain implementation.

General Description of the Prototype

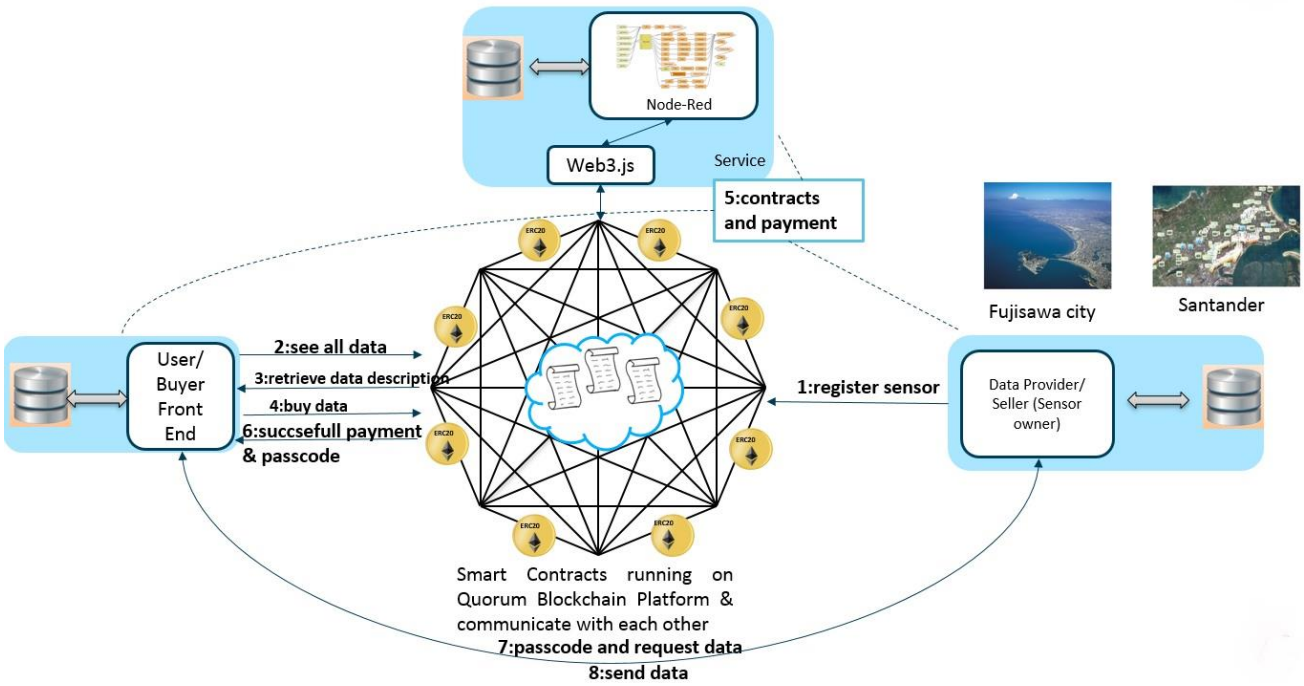

Figure 2. Overview of the M-Sec IoT Marketplace and its components

In the previous Figure 2, we can see an overview of the developed marketplace and its components, explained in detail through a specific example use of it.

1. The owner of a sensor/data source who wishes to make his data available for purchase or exchange registers himself to the dedicated created smart contract providing information about the type of the data, their frequency, the price, the location, etc.
2. A User of the M-Sec Platform who acts here as a potential buyer using our developed front end can see all the available sensors and their data
3. Upon finding some interesting data he/she can retrieve additional detailed descriptions about them and then
4. Buy the data of interest using M-Sec Tokens, which is a cryptocurrency in the form of a smart contract running in on blockchain presented in the previous section
5. The deployed smart contracts communicate with each other to verify the sufficient funds of the buyer and complete the purchase by transferring funds from the balance of the buyer to the one of the data owner. The developed Node-Red flows also assist in this process connecting the different components of the system
6. In the case of successful payment, when the buyer has sufficient funds and after the tokens are transferred, a passcode is returned to the buyer necessary for accessing the purchased data
7. The buyer communicates with the platform and the API of the data owner and using the transactions details requests the data
8. The desired data is returned to the buyer in a predefined format such as JSON

Components

Component Module 1: Node-Red Flows

In order to orchestrate the different components and services we have used Node-Red and have developed several flows. Node-Red is a powerful visual tool for wiring together hardware devices, APIs, and web-services, create flows that connect distributed components into a common IoT application[1].

We developed different flows for the different parts of the IoT Marketplace.

During the development of the system, we simulated the IoT weather sensors provided by public APIs and for this simulation, we used an API provided by Dark Sky[2]. Using Node-RED features we created flows that request current weather data for several locations from the Dark Sky API and then save these data (air temperature, relative humidity, pressure, visibility, wind speed and direction, sky cloud coverage, dew point, UV radiation, and the columnar density of total atmospheric ozone layer) into a local database. We also exposed a RESTful API in order to serve the data to the users when requested. When a request is received, the API key is checked. If it is correct, the data responding to the specified time intervals is retrieved from the database and then sent to the requester.

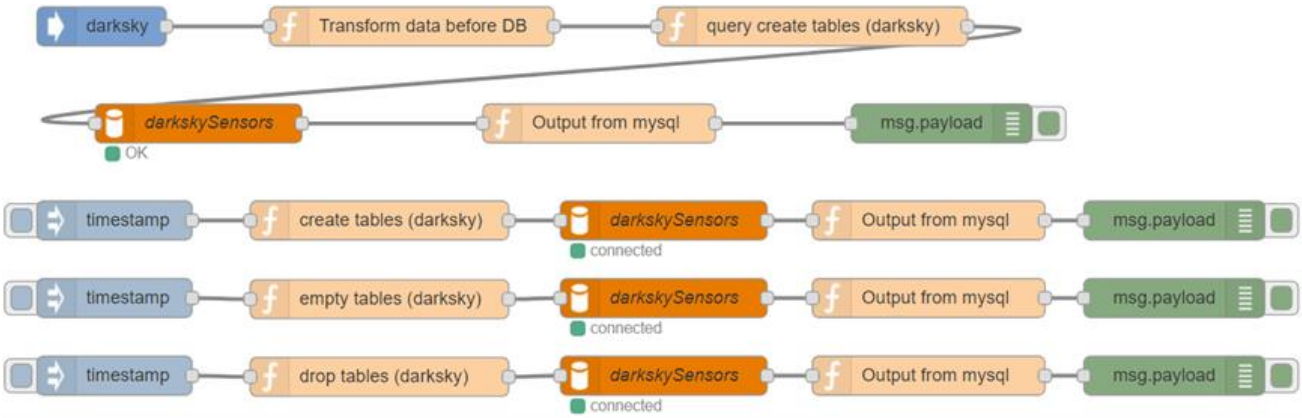

Figure 3. Node-Red Flow for the simulation of IoT sensor data

Component Module 2: Web Application

This web application provides interfaces between the users and the blockchain. It provides functionalities helping users interact with the smart contracts deployed on Quorum Blockchain and access data they have bought. It also allows sending transactions to and reading data of transactions and smart contracts. It also "protects" users from misreading or mistyping info when sending a transaction.

We have used different languages and technologies to create these interfaces such as JavaScript, Bootstrap, HTML, jQuery, Nodejs. Some of the developed interfaces are described below with screenshots and details. We have used Web3.js to interact with the deployed smart contracts.

[1] https://nodered.org/docs/

[2] The Dark Sky Company, LLC, "Dark Sky," The Dark Sky Company, LLC, [Online]. Available: https://darksky.net/dev

The user searches in all the available sensors registered in the Smart Contracts the sensors of interest specifying details in the corresponding fields such as the location, the type the data (temperature, starting date and time, frequency, etc.), as shown in Figure 4.

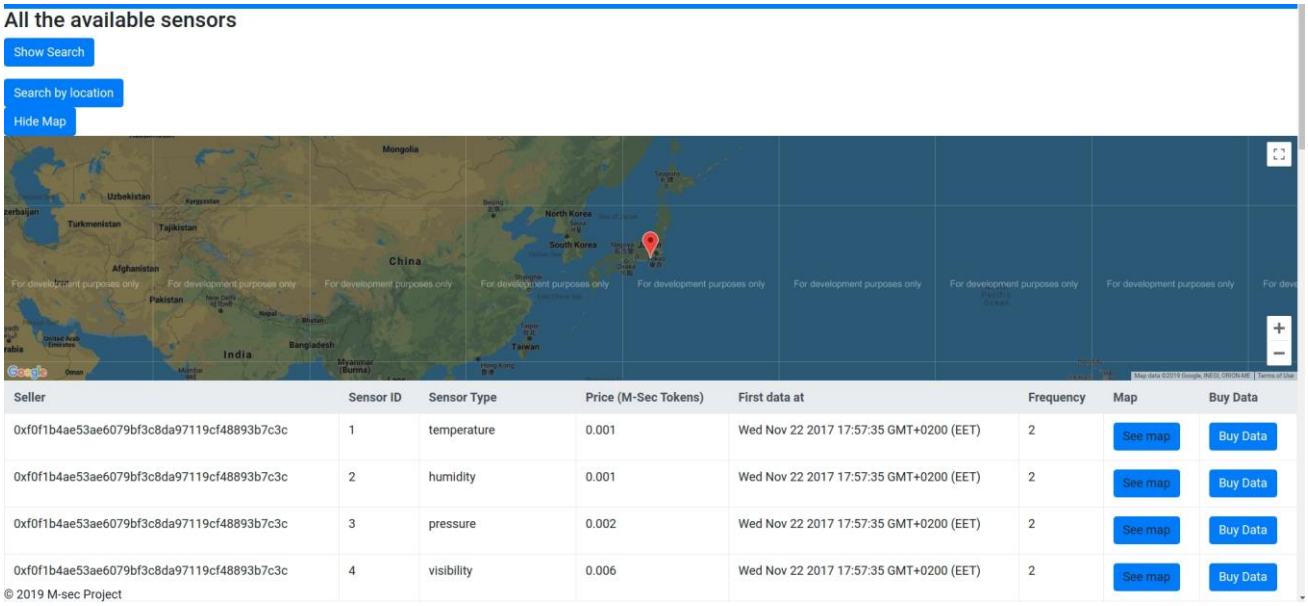

Figure 4. Graphical User Interface enabling searching of sensors in the smart contracts running on blockchain

After specifying all the required information, a query is submitted to the smart contracts running on the Quorum blockchain and a list of all the available sensors is returned with information of the address of the data owner, the sensor type (temperature, pressure, visibility, etc.), the frequency, a link opening a map and the option for the user to buy these data using M-Sec Tokens, as shown in Figure 5.

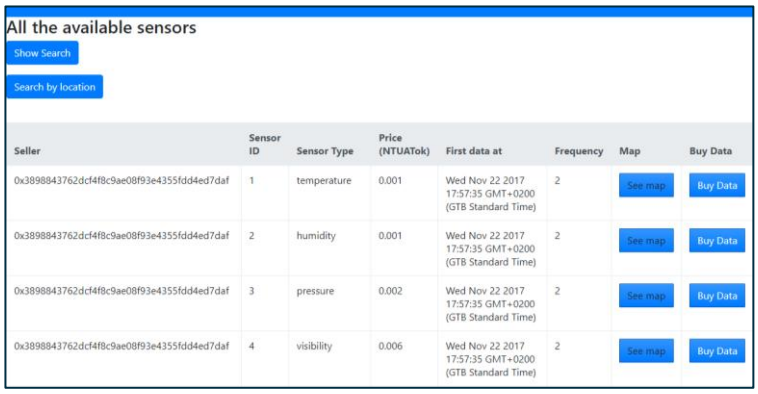

Figure 5. Graphical User Interface of the returned results after the query to the smart contract

An overview of the blockchain and the transactions included in each block is provided in our developed Explorer interface, as shown in Figure 6. The user is able to search for specific blocks, transactions, users, contracts and see the related activity.

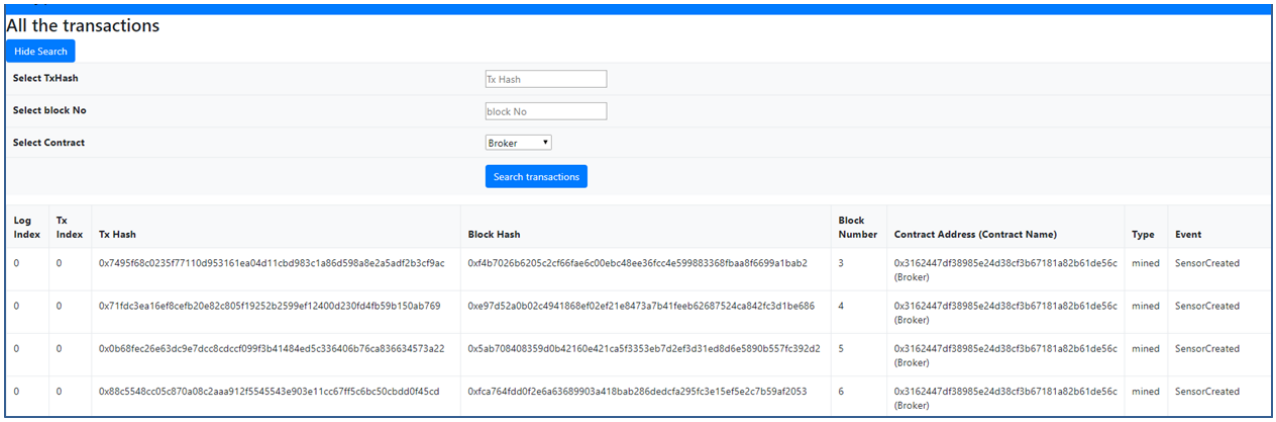

Figure 6. Graphical User Interface of the Explorer

To facilitate the different use cases (mainly use case 3, use case 4, and use case 5) and end-users of the pilots, we implemented the "SOXFire – Blockchain – IoT Marketplace Bridge", which is described in detail in the next section. Through this bridge, data are arriving from sensors, which are registered to the IoT Marketplace and some indicative figures are shown below.

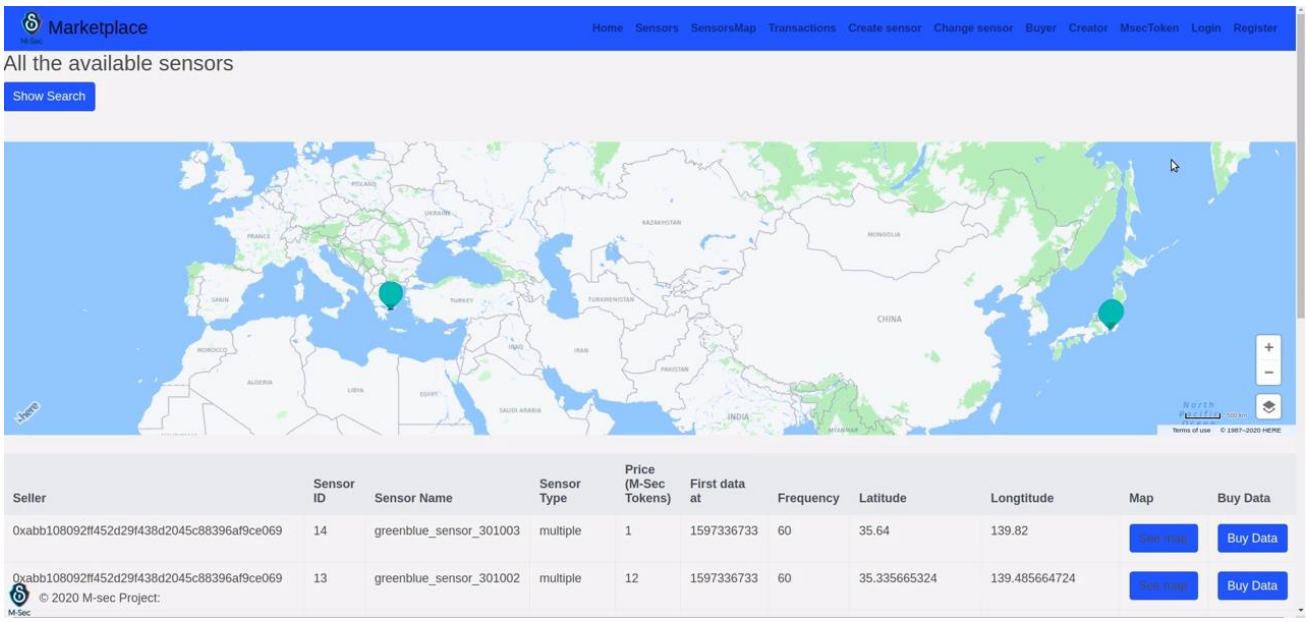

Figure 7. Available SOXFire sensors registered in IoT Marketplace

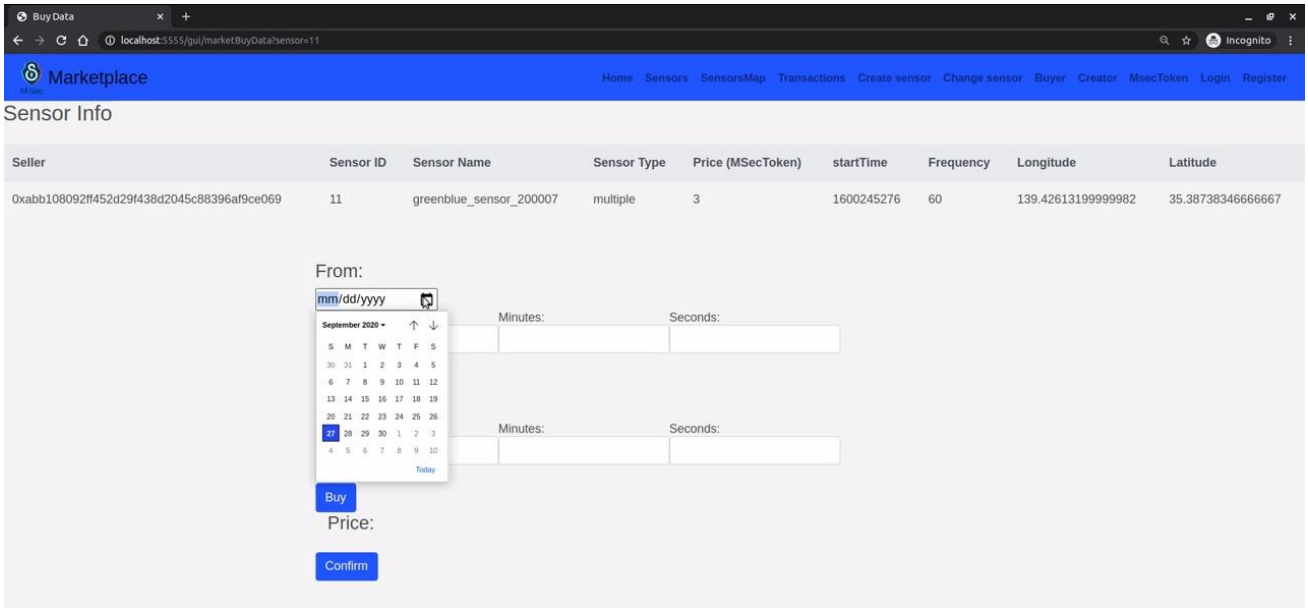

Figure 8. Buying data from a specific sensor

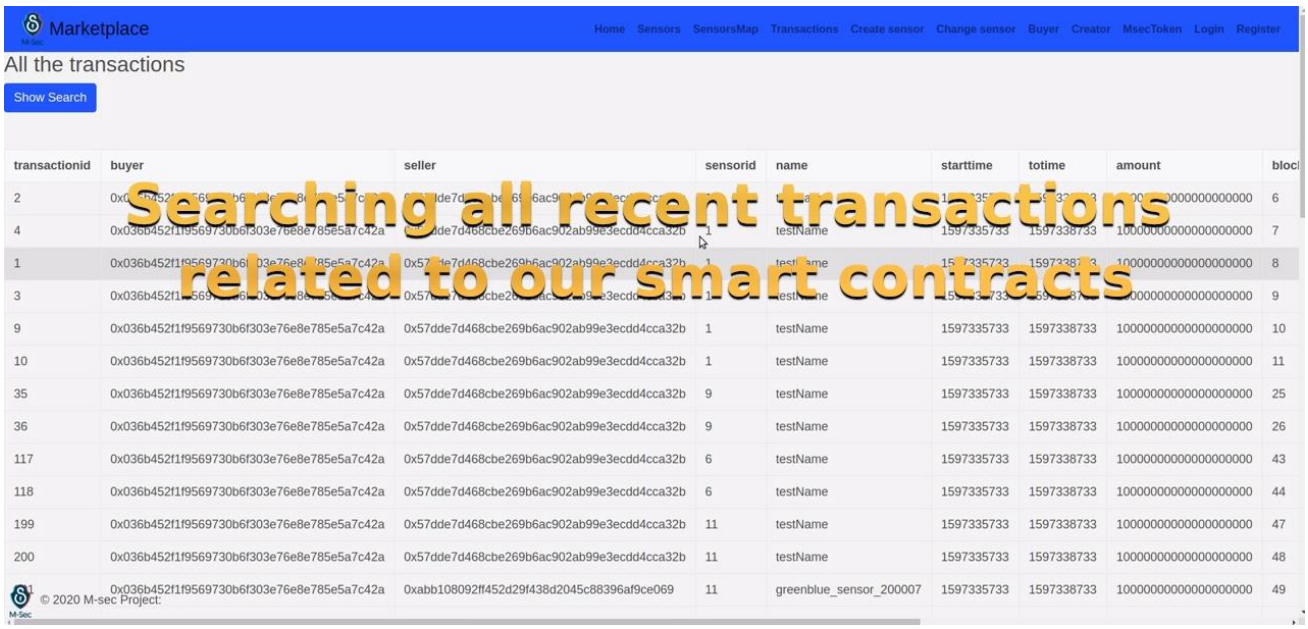

Figure 9. Browsing in the dedicated interface for recent activity and recent transactions

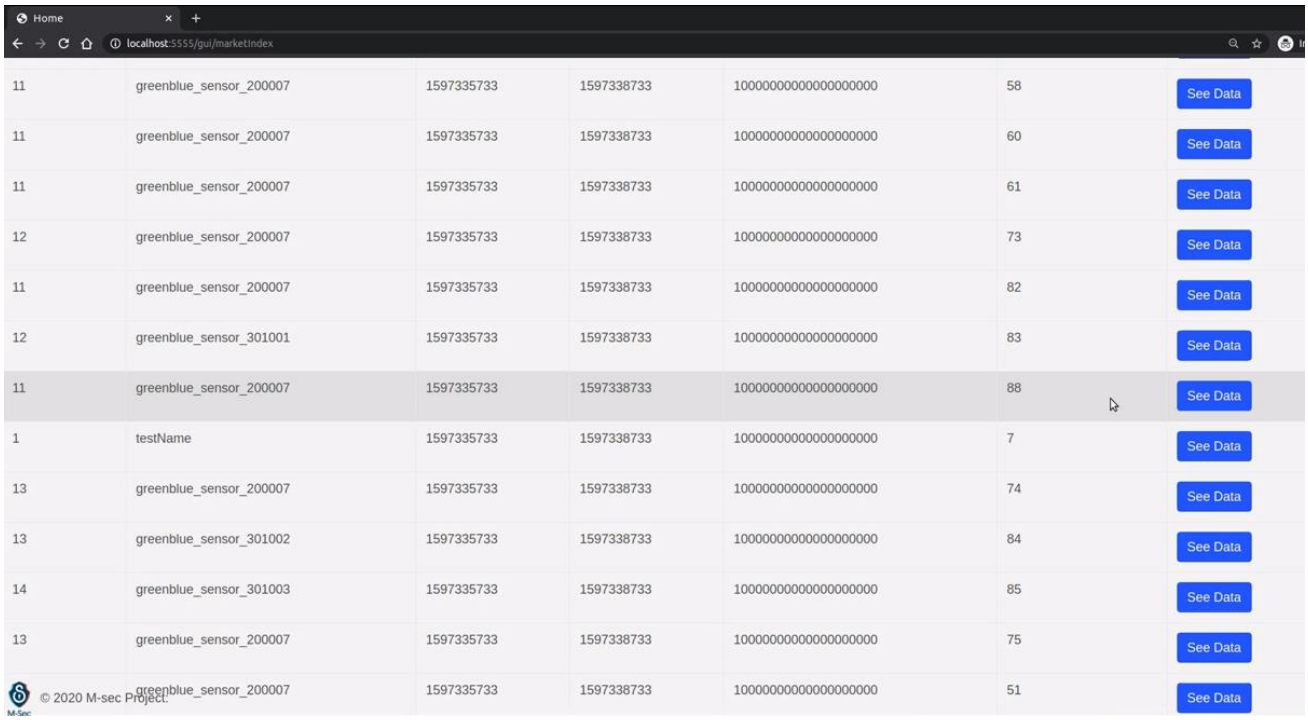

Figure 10. Browsing all the purchased data from sensors.

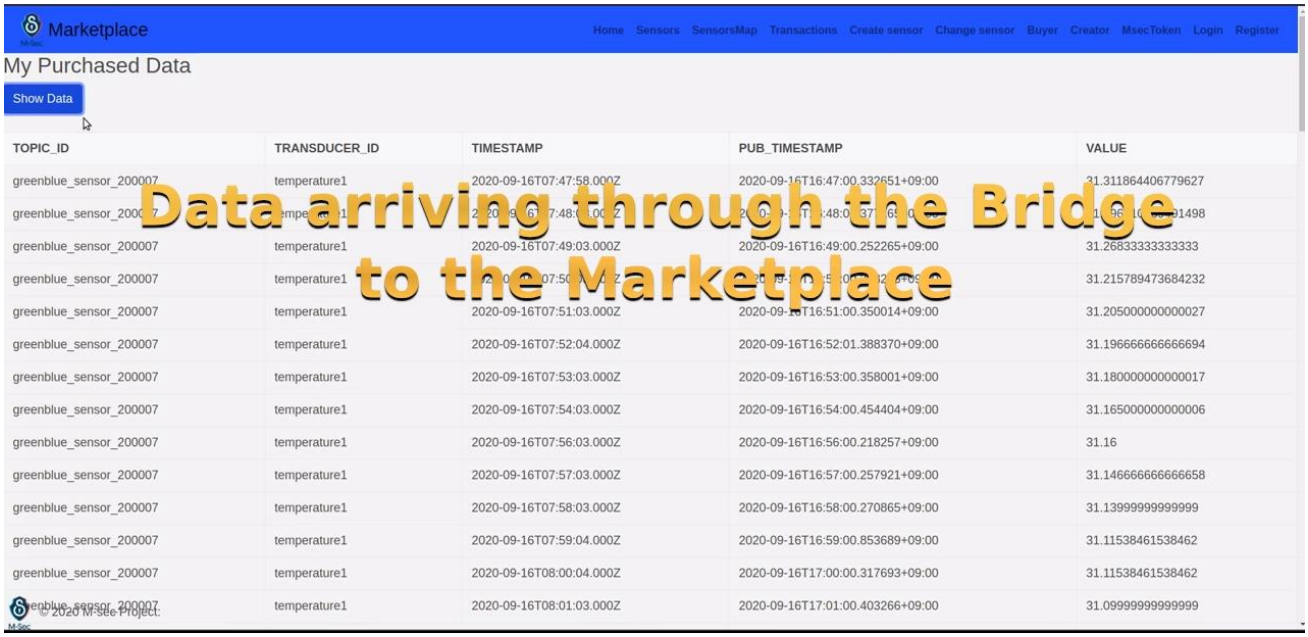

Figure 11. Example of data from sensors located in Japan and arriving to Marketplace through the Bridge

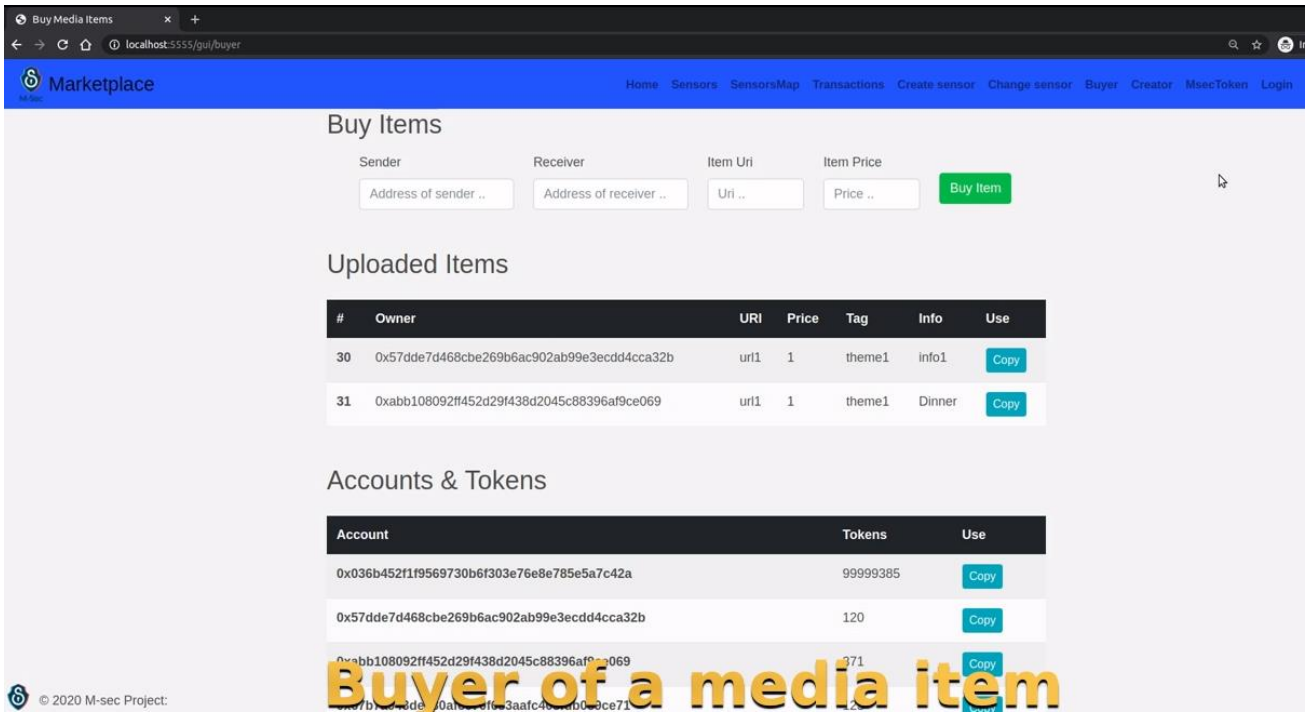

Figure 12. Interfaces supporting the buyers and seller of media items

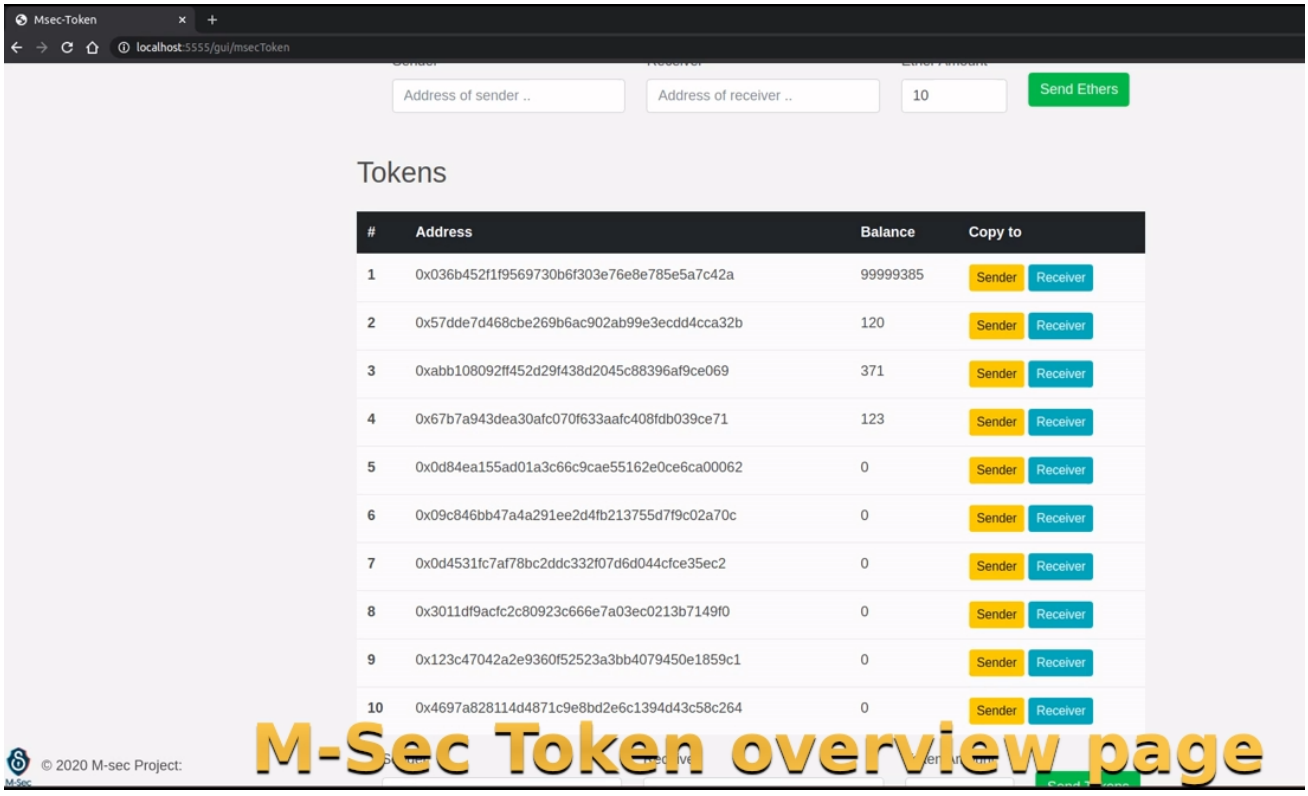

Figure 13. M-Sec Token overview page

API: Blockchain framework exposed methods

In order to allow communication and integration with other components, services, assets, several methods were developed. These methods are exposed via a RESTful API, while respective clients have been developed to facilitate the integration process and documentation with examples and indicative architecture figures and snippets. In the table that follows, some of the methods are presented, while we could note that part of them have a final form, while others are still updated to facilitate the integration with other assets, better support the use cases based on the feedback or improve security aspects of the provided services.

Method "registerNewSensor"

Name	Input	Response	Description
registerNewSensor ()	String publicKey, int sensorType, double price, int startTime, double frequency, double lat, double long, String urlOfData	{"message":" Successfully registered sensor" , "id": int}	This is a POSTmethod, which Registers a new sensor belonging to a specific user. Price is described in M-Sec tokens, timestamp in Unix timestamp, and number of measurements is provided per hour e.g. 6 measurements/hour. The URL of the data is the location, where the buyer can find them, it could be any kind of database: SQL, MongoDB, IPFS, etc

Regarding the sensor type, the possible values are described in the table that follows:

Table 2. Types of sensors

Code	Type of Sensor	Proposed unit of measurement
1	Temperature	°C
2	Relative humidity	%
3	Pressure	Hectopascal – hPa

4	Visibility	km
5	Wind speed and direction	m/s
6	Sky cloud coverage	%
7	Dew point	°C
8	Solar Radiation	watt/m²
9	UV index	0 to 11
10	Columnar density of total atmospheric ozone layer	Dobson – DU
11	Motion sensors	Int [0,..]
12	Door window	Int [0,..]
13	Smart Plug	Volts
14	Smoke	Bool (on/off)
15	Mattress	Bool (on/off)

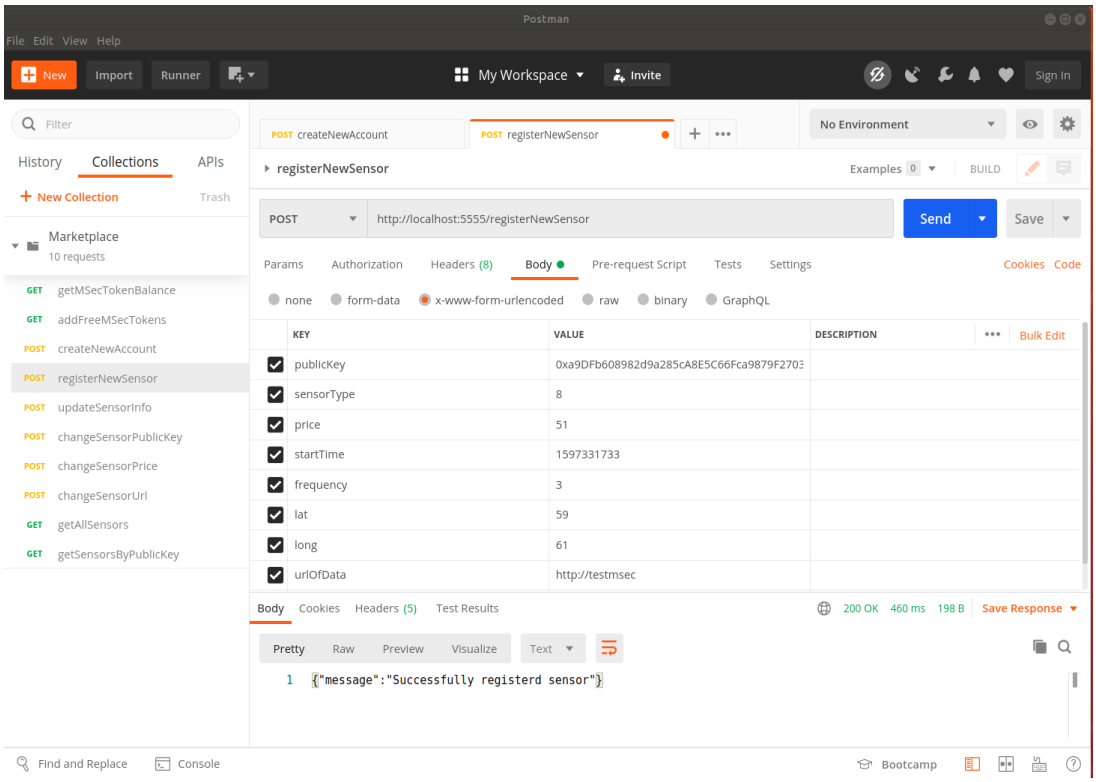

Figure 14. Example call of registerNewSensor function, using Postman

Method “getAllSensors”

Name	Input	Response	Description
getAllSensors()	-	[{ "sensorid": , "seller": , "sensortype":, "price": , "starttime":, "frequency":, "latitude": "", "longtitude": "", "theurl": "", "logindex":, "transactionindex":, "transactionhash":, "blocknumber":, "blockhash": ""	This is a GET method, which returns details about all the sensors registered by all users/ smart cities

		}]	

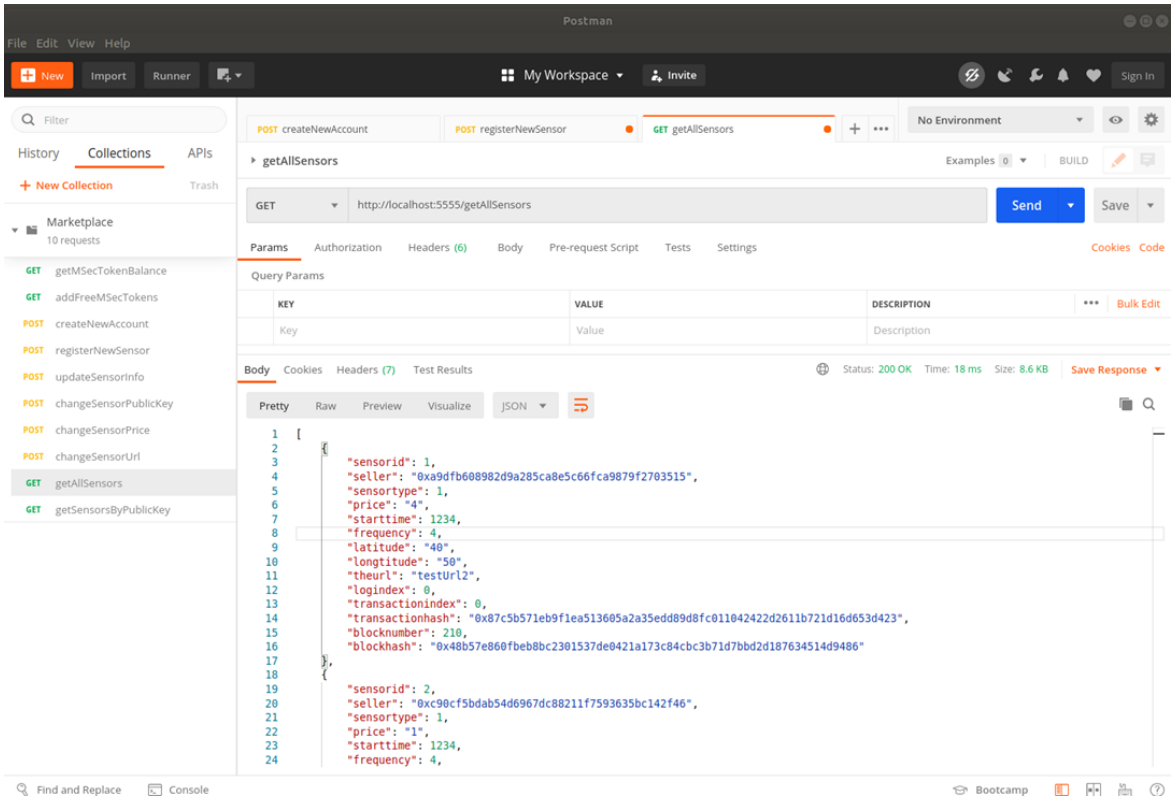

Figure 15.Example call of getAllSensors function, using Postman

Method “getSensorsBySellerPublicKey”

Name	Input	Response	Description
getSensorsBySellerPublicKey ()	-	[{ "sensorid": , "seller": , "sensortype":, "price": , "starttime":, "frequency":, "latitude": "",	This is a GET method, which returns details about all the sensors registered by a specific users/ smart cities

		"longtitude": "", "theurl": "", "logindex":, "transactionindex":, "transactionhash":, "blocknumber": 210, "blockhash": "" }]	

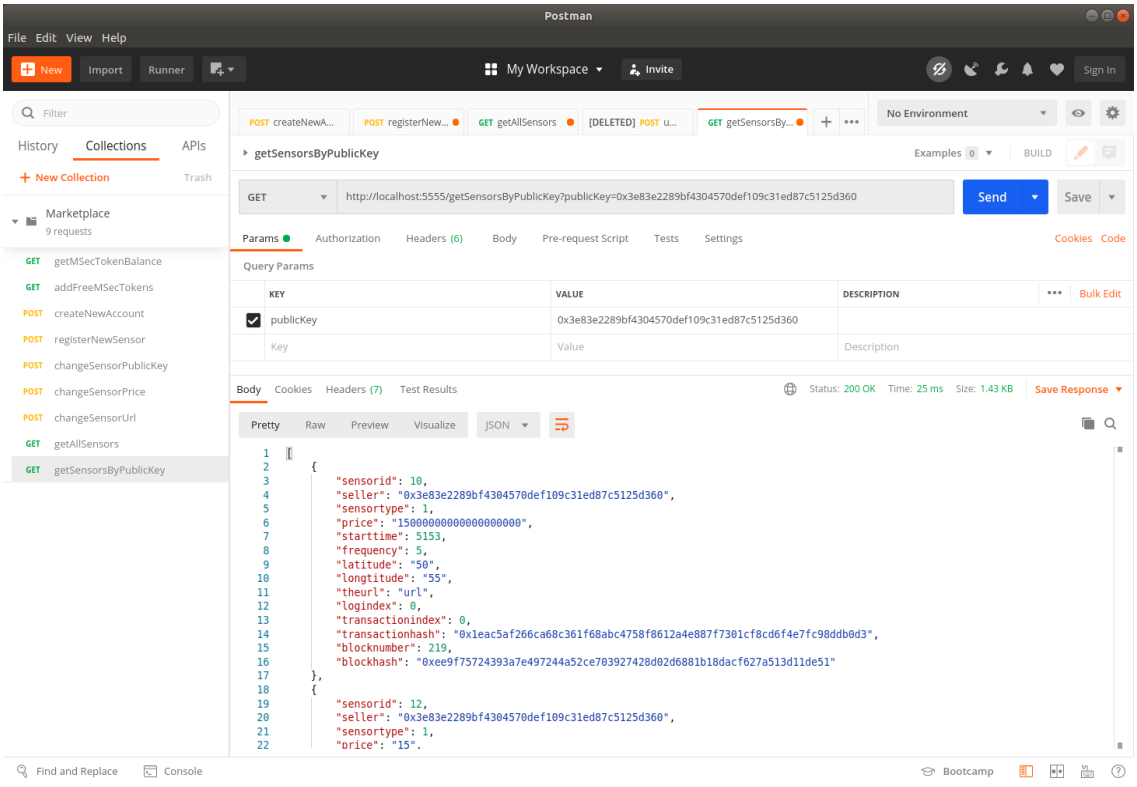

Figure 16. Example call of getSensorsBySellerPublicKey function, using Postman

Method “changeSensorPublicKey”

Name	Input	Response	Description
changeSensorPublicKey()	String publicKey, sensorID, String publicKey	{“sensor”:””, “message”:”success”}	This is a POST method, which changes the owner of a sensor

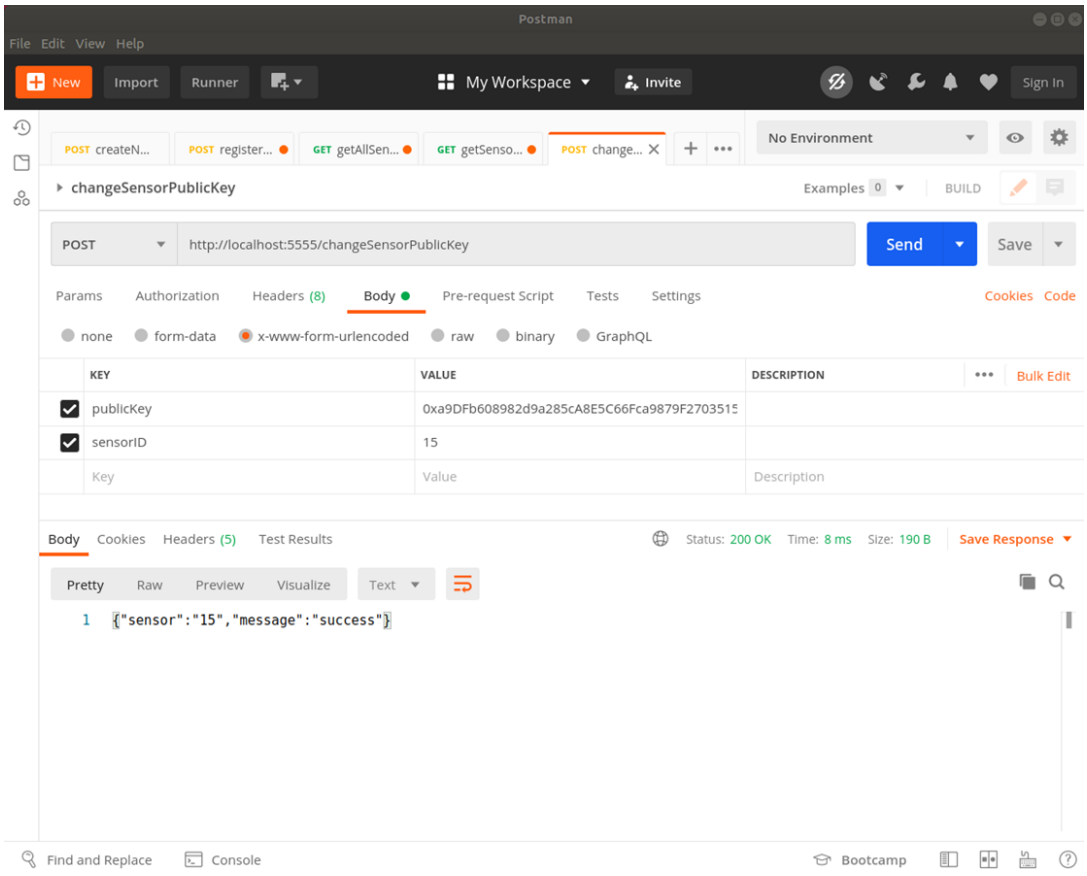

Figure 17. Example call of changeSensorPublicKey function, using Postman

Method “changeSensorPrice”

Name	Input	Response	Description
changeSensorPrice ()	String publicKey, sensorID, newPrice	{“sensor”:””, “message”:”success”}	This is a POST method, which changes the price of data for a sensor

Method “changeSensorUrl”

Name	Input	Response	Description

changeSensorUrl()	String publicKey, sensorID, newUrl	{“sensor”:””, “message”:”success”}	This is a POST method, which changes the url of data for a sensor

Method “purchaseMediaItem”

Name	Input	Response	Description
purchaseMediaItem()	String publicKey, int photoId	json	This is a POST method, for purchasing media items

Method “getPurchasedPhotos”

Name	Input	Response	Description
getPurchasedPhotos ()	String publicKey	{“sensor”:””, “message”:”success”}	This is a GET method, which changes the url of data for a sensor

Method “browsePhotos”

Name	Input	Response	Description
browsePhotos ()	filters [location, time]	json (details, urls to IPFS)	This is a GET method, which returns media items or photos with specific filters

Package Information & Installation Instructions

Required Tools and dependencies

The following tools and dependencies are required to install and use the IoT Marketplace:

- NodeRed
- Nodejs

- MySQL
- Ethereum/Quorum Blockchain
- Nodejs and Javascript

Install NodeRed

- **Node-Red**: Node-RED is a powerful visual tool for wiring together hardware devices, APIs and web-services, create flows and connect distributed components into a common IoT application [https://nodered.org/].
    - Installing Node-Red: The easiest way to install Node-RED is to use the node package manager, npm, which comes with Node.js [https://nodered.org/docs/getting-started/installation]. Installing as a global module adds the command "node-red" to your system path:
        - For Ubuntu:
            - sudo npm install -g --unsafe-perm node-red
        - For Windows, execute CMD with administrator rights and then execute:
            - npm install -g --unsafe-perm node-red
    - Version: We have installed Node-Red v0.17.5
    - Running: after installing Node-Red as a global npm package, open a terminal and run the "*node-red*" command. You can then access the Node-RED editor by pointing your browser at: http://localhost:1880
    - After accessing the editor, you have to left-click on the menu button (three lines on the top right corner), then click on manage palette, switch to the install tab and search for the node-red-contrib-neo4j package and install it. This will add the node required by our flows ensuring the dependency.

Install Nodejs

- **Node.js**: Before installing Node-Red, a Node.js installation is required. We have installed Node.js version v8.9.3.
    - On Ubuntu machines we have to run the following commands:
        - sudo apt-get update
        - sudo apt-get upgrade
        - sudo apt-get install node.js -y
        - sudo apt-get install npm -y
    - On Windows machines, we can download the appropriate installer from https://nodejs.org/en/download and execute it.

Install MySQL

- We used the MariaDB SQL, but any other SQL relational database can be used. Full instructions of how to install MariaDB database can be found here: https://downloads.mariadb.org/.

Install Front End

- Front End: We have developed a web front end, useful for end-users of our application. It provides a Graphical User Interface
    - Based on HTML, Javascript, Vue Javascript framework and other libraries
    - Running: it is deployed on our server (and cloud servers as well) and accessible in http://snf-755174.vm.okeanos.grnet.gr

Install Java

- Most of the systems used are built on top of java engines so a Java distribution needs to be installed in the system before anything else.
    - On Ubuntu machines a simple list of commands is enough to install the latest distribution of Java:
        - sudo add-apt-repository ppa:webupd8team/java
        - sudo apt-get update
        - sudo apt-get install -y oracle-java8-installer
        - sudo apt-get update
    - On Windows machines we have to download the appropriate installer from https://java.com/en/download and execute it

Install Ethereum/Quorum Blockchain

Instructions are provided in the previous Section related to the Blockchain demonstrator.

Operating System

- We have tested the platform on Windows 10 and Ubuntu 18 but all of the software listed here is available in a large number of other distributions.

Okeanos

- Okeanos: We have deployed our Node-Red and Neo4j services to Okeanos cloud service for Greek Research and Academic Community
    - https://okeanos.grnet.gr/home/

Licensing (if applicable)

Since ICCS/NTUA is a non-profit Academic Research Body, we will be releasing all related M-Sec results as open-source contributions under Open Source licenses. Concretely, permissive licenses are not restrictive licenses and they can be used to create a proprietary good, allowing commercial exploitation and ensuring high impact. Examples of those are Apache, BSD, etc.

2.3 Interactions with other FGs

The following figure presents the interactions of the Secure City Data Access FG with other FGs and components of the M-Sec solution. The Annex presents the position of the FG within the whole M-Sec Architecture.

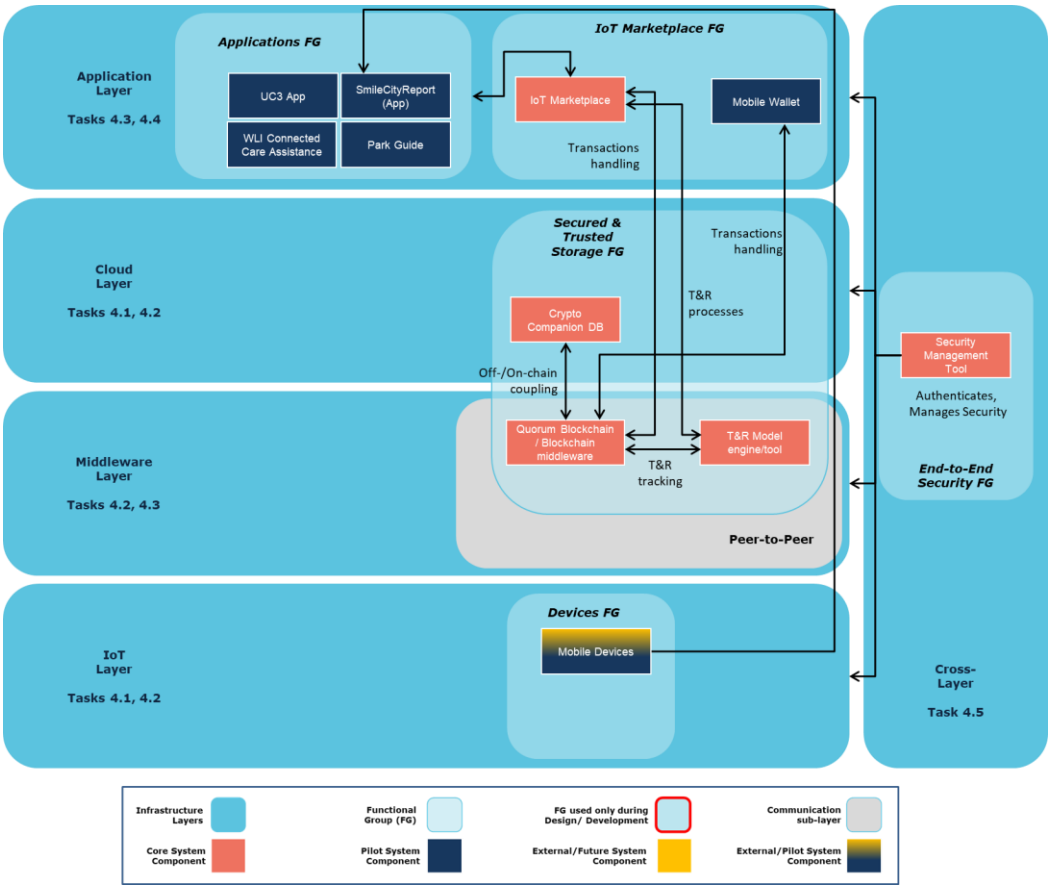

Figure 18. IoT Marketplace FG with other FGs

Interaction with Security and Trusted Storage FG

Integration between the IoT Marketplace FG and the Security and Trusted Storage FG was implemented. In more detail, IoT Marketplace was directly integrated to all components of Security and Trusted Storage FG namely Blockchain Framework, Middleware Services, and Crypto Companion DB.

In the following figure, the flow of the data can be seen, showing the integration of both FGs having as a communication component the Worldline Connected Care Assistance.

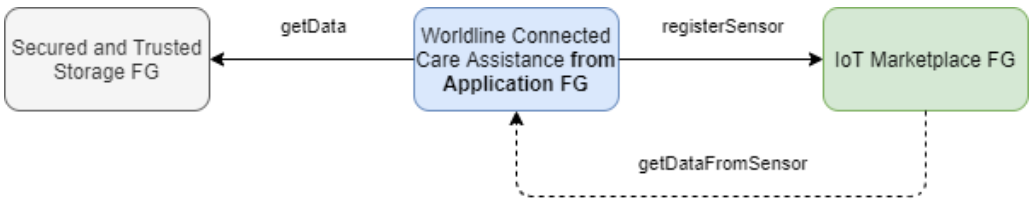

Figure 19. Example of data flow between Secured and Trusted Storage FG and IoT Marketplace FG

Interaction with Security city-data Access FG

A point of integration with Security city-data Access FG and Security and Trusted Storage FG was implemented to facilitate different use cases (mainly use case 3, use case 4, and use case 5) and end-users of the pilots. To this direction, a new component was developed, namely “SOXFire – Blockchain – IoT Marketplace Bridge”

allowing registration of sensors, purchase/exchange of data, and visualization of data. An overview of this component is shown in the following figure.

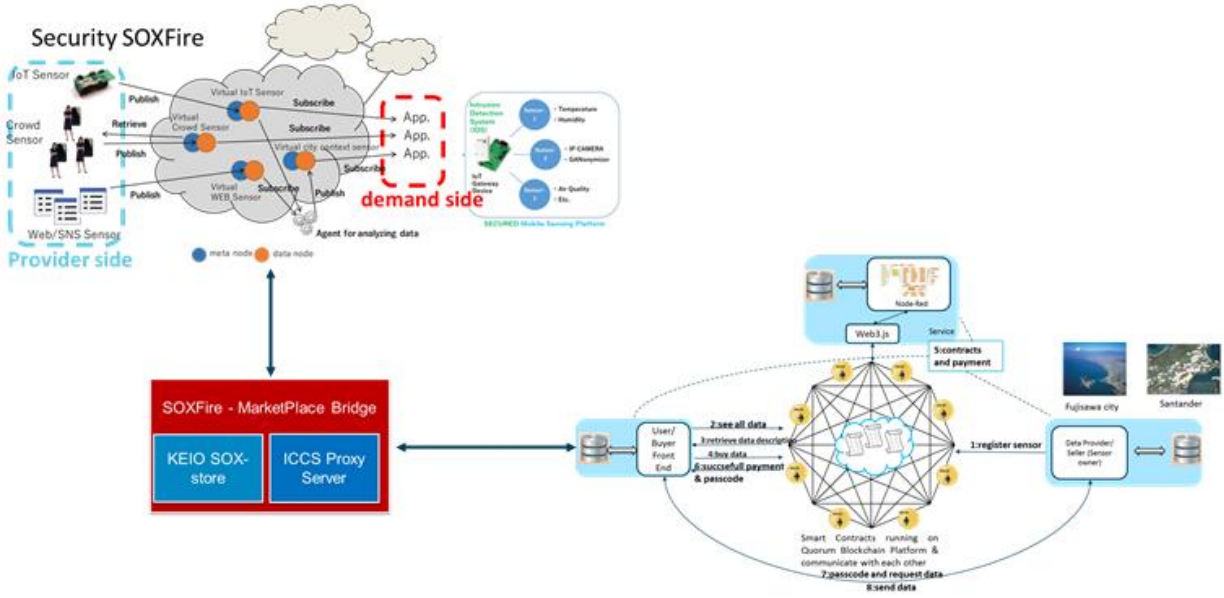

Figure 20. Overview of SOXFire – Blockchain - IoT Marketplace Bridge

A more technical figure is provided below, displaying details of this integration.

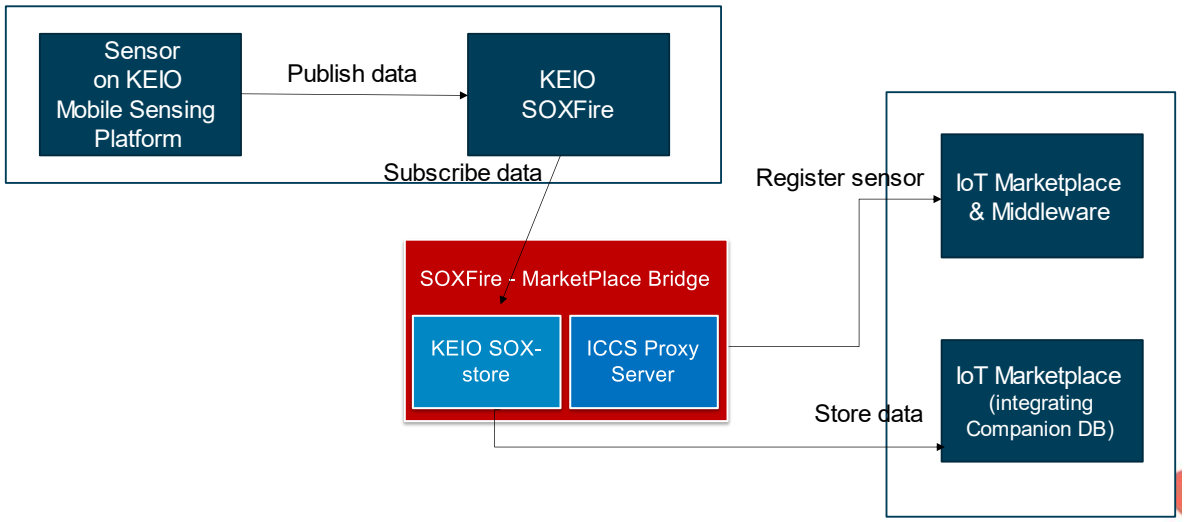

Figure 21. Technical overview of SOXFire – Blockchain – IoT Marketplace Bridge

Interaction with End To End FG

End-to-end security functional group provides accounting for the IoT Marketplace functional group. It enables clients of the marketplace to be authenticated either as the owner of devices, owner of data, or simply a consumer.

One interest in using the security manager instead of an internal database is to:

- prevent the inclusion of falsified data in the marketplace by verifying and attesting the authenticity of the source cryptographically.
- enable to trace the usage of the marketplace, in particular, to provide automatic breach notification and other forms of remediation.

We present in the diagram below a mutual authentication between an IoT marketplace app and a secure device. The secured device uses HTTPS flow to verify the IoT. The IoT marketplace app uses TLS Client Authentication to verify the identity of the client (secure device).

In the diagram in Figure 22, we present a use-case allowing a secure device to send monitoring data to an IoT marketplace app (here replaced by an InfluxDB database for development and testing purposes) every x seconds. The proxy server (Nginx in the figure) verifies and checks the cert validity of every request using the M-Sec OCSP mechanism. If the cert is valid, data will be stored on the influxDB. Otherwise, data will be rejected, and users' owners are notified.

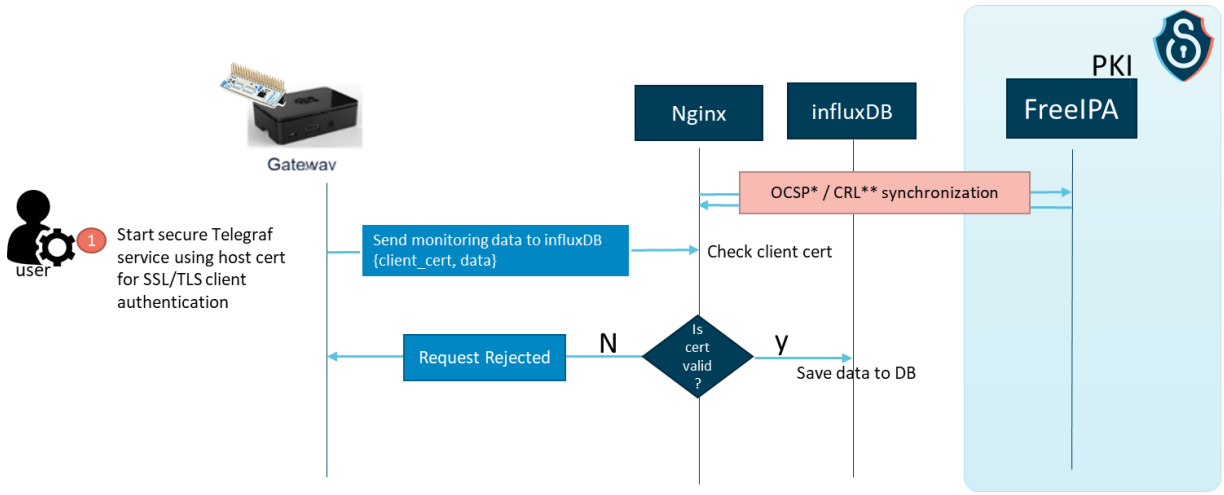

Figure 22. Example of certificate-based authentication and authorization between an IoT device and a backend

3. Development & Security Designing Tools FG

3.1 General Description of the FG

The Development & Security Designing Tools FG aims to establish engineering foundations to support the development of secure smart city applications. Although the smart-city platform itself is secured, application-level vulnerabilities make systems insecure. However, ensuring application-level security requires tremendous effort for developers. Components in the Development & Security Designing Tools FG provide a set of methodologies and tools to support the development of secure smart city applications on top of the M-sec platform. Key benefits achieved by those components include:

- Reduce developers' effort for analyzing security requirements
- Mitigate risks to miss typical security threats
- Mitigate risks to include application-level vulnerabilities in specification by human errors

Specifically, the Development & Security Designing Tools FG in M-Sec provides the following key components, which will be explained in this section:

- Security Analysis Tool
- Modal Transition System Analyzer

3.2 Components of the FG

Secure Analysis Tool (SAT)

SAT is a software security requirements modeling support tool for a misuse case diagram that enables the creation of a diagram by embedding the element of the security knowledge.

SAT is implemented as a plugin of astah* (a UML modeling tool developed by Change Vision, Inc) Main features provided by SAT are as follows:

- ***Misuse case diagram editor*** to draw misuse case diagrams,
- ***Knowledge base browser*** to represent security knowledge in the knowledge base.
- ***Knowledge search*** to search an appropriate security knowledge in the knowledge base.

Application developers with M-Sec platform can define new security countermeasures for the threats identified by creating a misuse case diagrams.

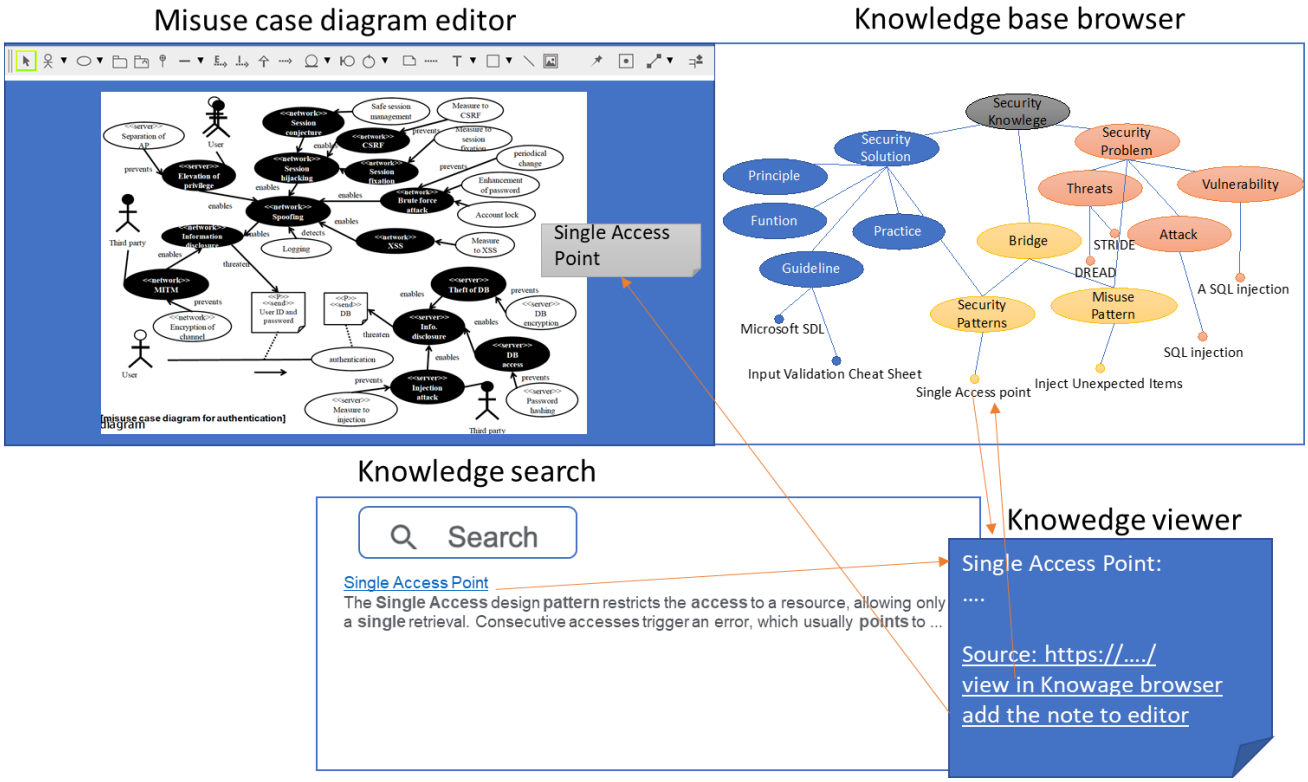

Figure 23. The Security Analysis Tool

Modal System Transition Analyzer (MTSA)

MTSA is a software development tool to automatically synthesize a software behaviour specification with a formal guarantee. A technique supported by the MTSA is called discrete controller synthesis. The discrete controller synthesis is a generation technique of “correct” behaviour specification based on the two-player game theory. It takes requirements specified as fluent linear temporal logic (FLTL) and environmental assumptions specified as labeled transition system (LTS) as inputs and synthesizes a discrete event controller as an LTS. The synthesis algorithms are based on the two-players game. Hence, the synthesized behaviour specification is ensured to satisfy the requirements under the environmental assumptions.

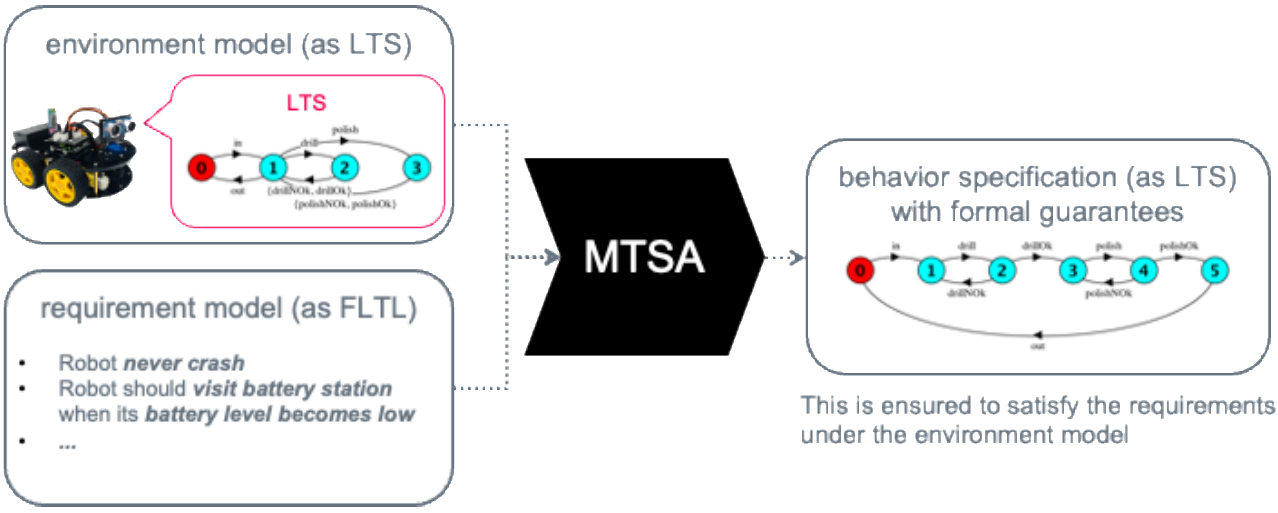

Figure 24. MTSA Overview

MTSA is an open-source, integrated development tool for discrete controller synthesis. The main features provided by MTSA are as follows:

- ***model editor*** helping developers to specify LTS-based environment model and FLTL-based requirement model,
- ***synthesis engine*** generating a behaviour specification model in LTS, and
- ***model animator*** to validate the environment model or the synthesized behaviour specification model.

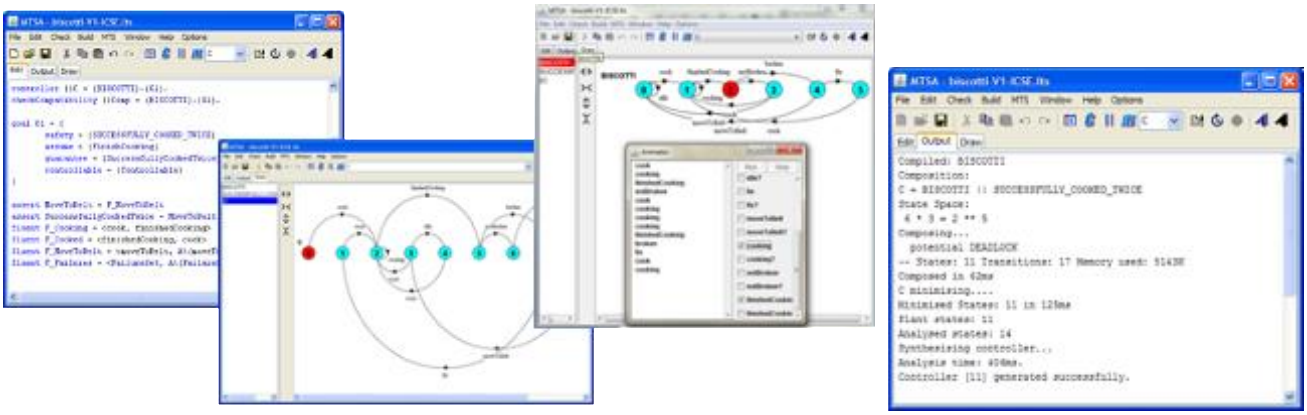

Figure 25. MTSA Functionalities

Application developers with M-Sec platform can utilize the MTSA to synthesize "secure application specification" against to business logic vulnerabilities by formalizing business logic vulnerabilities as FLTL properties. MTSA can synthesize vulnerability-free behaviour specifications with formal guarantees. If developers correctly implement the synthesized specification as programs that are executable on the M-Sec platform (such as Node-RED programs), the developed smart city application is also ensured to satisfy the security requirements under the environment model.

3.3 Interactions with other FGs (or assets)

Interaction with Node-RED

Although MTSA automatically synthesizes the “correct” behaviour specification model, developers might incorrectly implement programs. To avoid such bug injections by human errors, we developed a bridge tool connecting MTSA and Node-RED. The bridge tool automatically generates a Node-RED program correctly implementing an LTS-based behaviour specification model synthesized by MTSA.

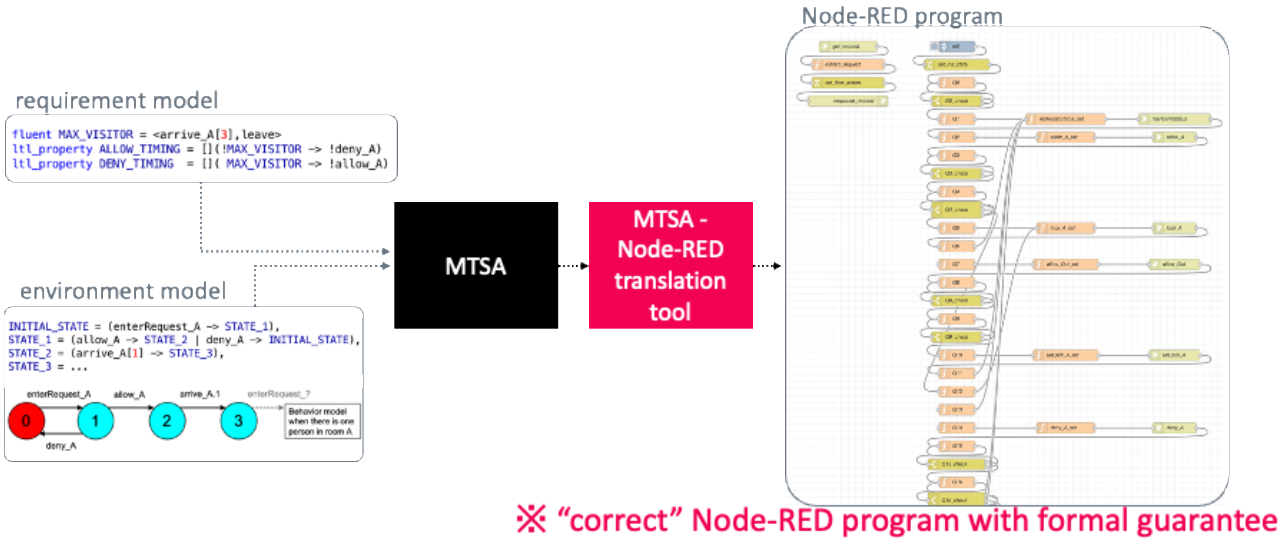

Figure 26. MTSA-Node-RED Integration via Translation Tool

The translation is not straightforward, because a behaviour specification synthesized by MTSA is LTS, which represents an event-triggered controller (a kind of state machine) whereas Node-RED program is a kind of dataflow graph. The translation tool implements some translation patterns to correctly maps application logic specified in an LTS into a dataflow graph.

behavior specification model(LTS)
as ***event-triggered controller***

```
Transitions:
        PartController = Q0,
        Q0      = (reqEnter_Hall -> Q1),
        Q1      = (resRoomStatus -> Q2),
        Q2      = (allow_Hall -> Q3
                  |deny_Hall -> Q10504),
        Q3      = (arrive_Hall[1] -> Q4),
        Q4      = (reqEnter_A -> Q5
                  |reqEnter_Hall -> Q10497),
        Q5      = (resRoomStatus -> Q6),
        Q6      = (deny_A -> Q7
                  |allow_A -> Q8),
        Q7      = (arrive_Hall[1] -> Q4),
        Q8      = (arrive_A[1] -> Q9),
        Q9      = (reqEnter_B -> Q10
                  |reqEnter_Hall -> Q10493),
        Q10     = (resRoomStatus -> Q11),
        Q11     = (deny_B -> Q12
                  |allow_B -> Q13),
        Q12     = (arrive_A[1] -> Q9),
        Q13     = (arrive_B[1] -> Q14),
        Q14     = (reqEnter_C -> Q15
                  |reqEnter_Hall -> Q10489),
        Q15     = (resRoomStatus -> Q16),
        Q16     = (deny_C -> Q17
                  |allow_C -> Q18),
        Q17     = (arrive_B[1] -> Q14),
        Q18     = (arrive_C[1] -> Q19),
        Q19     = (reqEnter_D -> Q20
                  |reqEnter_Hall -> Q10485),
        Q20     = (resRoomStatus -> Q21),
        Q21     = (deny_D -> Q22
                  |allow_D -> Q23),
        Q22     = (arrive_C[1] -> Q19),
        Q23     = (arrive_D[1] -> Q24),
        Q24     = (reqOut -> Q25
                  |reqEnter_Hall -> Q31),
        Q25     = (resRoomStatus -> Q26),
        Q26     = (deny_Out -> Q27
                  |allow_Out -> Q28),
        Q27     = (arrive_D[1] -> Q24),
        Q28     = (leave -> Q29),
        Q29     = (reqEnter_Hall -> Q30),
        Q30     = (resRoomStatus -> Q2),
        Q31     = (resRoomStatus -> Q32),
        Q32     = (allow_Hall -> Q33
```

Node-RED program (json)
as ***data flow graph***

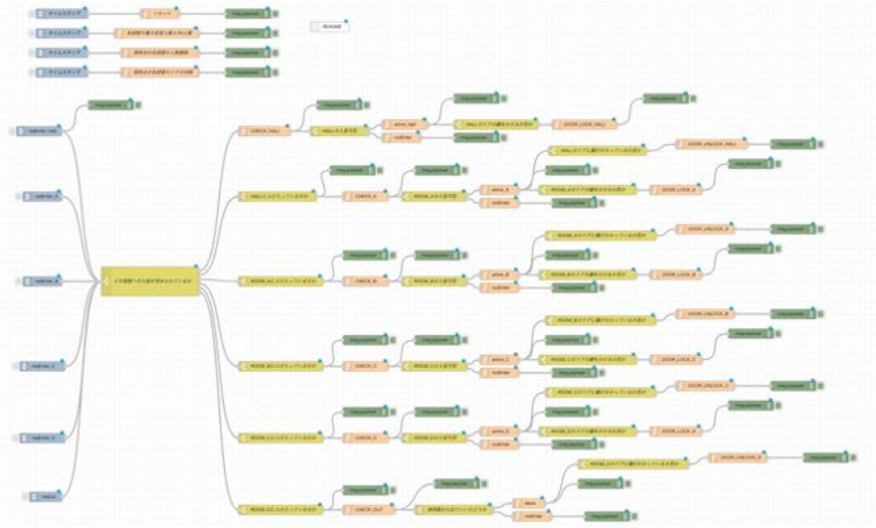

Figure 27. An Example of LTS-based behavior specification and corresponding Node-RED program

Developers should perform the following procedure to obtain Node-RED program.

- Specify requirements as FLTL and environment assumptions as LTS
- Synthesize behaviour specification model by using MTSA by inputting them. Then, obtain LTS-based behavior specification
- Translate the LTS-based behavior specification into NodeRED program by using the translation tool.

The generated a Node-RED program can be executed on Node-RED platform orchestrating sensors and actuators provided by a smart city platform. Thanks to MTSA, the program is ensured to satisfy given security requirements under the environmental assumptions.

3.4 API

Components in the Development & Security Designing Tools FG are tools and methodologies used by developers at development time. No APIs are provided.

4. Conclusion

This deliverable D4.8 is the final version of the Deliverables related to T4.4. As presented in the original M-Sec architecture already introduced in previous reports such as Deliverable 3.4, the layer addressed by this task is the Applications Layer. Two main FGs have been identified and documented in detail namely the IoT Marketplace FG and the Development & Security Designing Tools FG. Their components, the interactions and integrations with other components as parts of a unified architecture is also documented in detail.

Annex

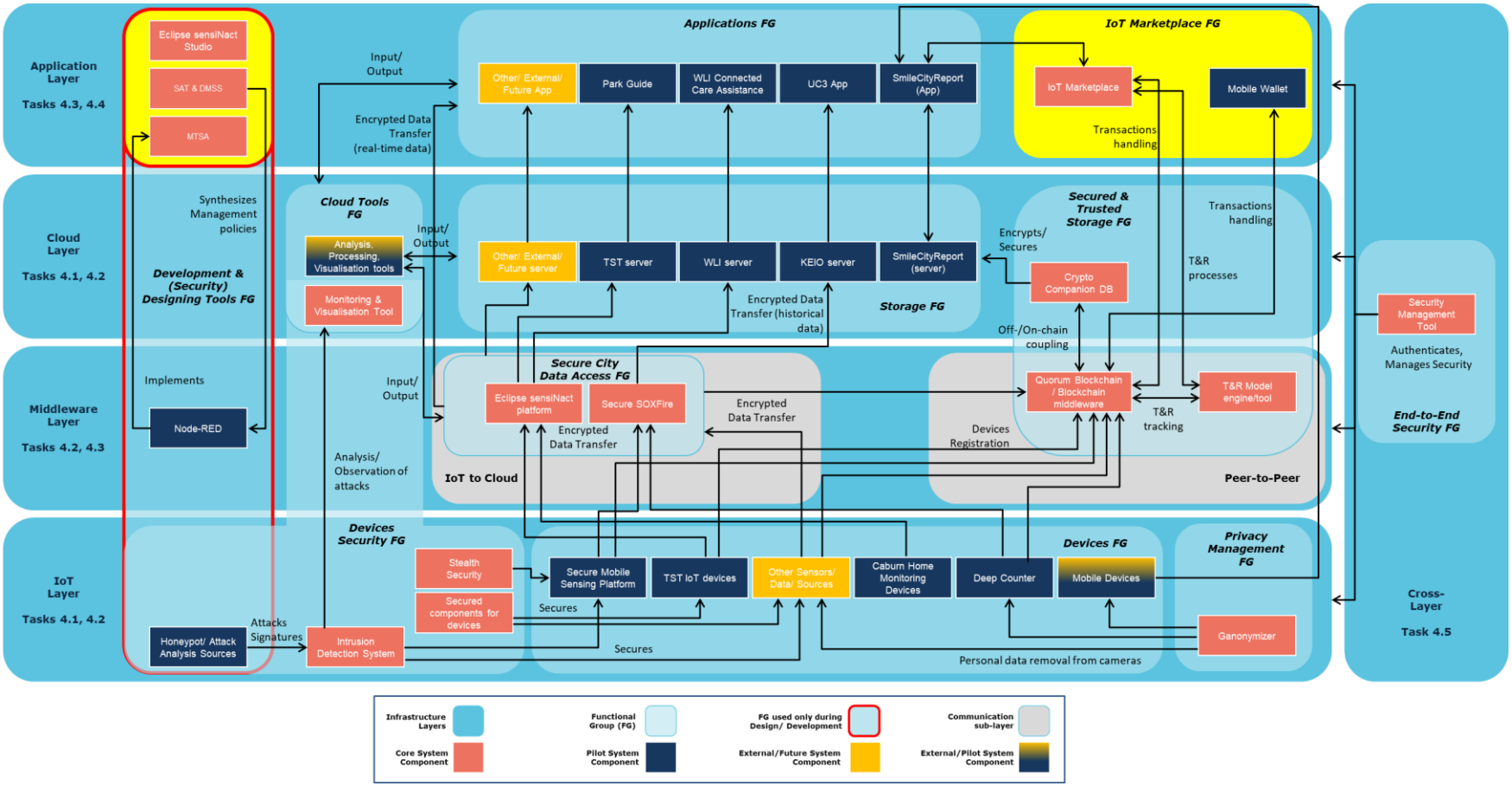

Figure 28. The M-Sec Architecture (T4.4 FGs in yellow)