



## D3.1 High Level System Architecture

Deliverable ID:	D 3.1
Deliverable Title:	High Level Architecture
Revision #:	1.0
Dissemination Level:	Confidential
Responsible beneficiary:	VTT
Contributing beneficiaries:	All
Contractual date of delivery:	04.03.2019
Actual submission date:	19.02.2019

## Version history

Date	Contributor	Contribution	Version
19.06.2018	<b>SIVECO</b>	D3.1 structure	<b>1.0</b>
07.08.2018	<b>ETRI</b>	Contributions in chapters: 4.1 Storage;	<b>2.0</b>
18.08.2018	<b>CUNI</b>	Contributions in chapters: 3.1 Presentation; 4.1 Storage;	<b>2.0</b>
21.08.2018	<b>SIVECO</b>	Contributions in chapters: 3.1 Presentation; 3.2 Web and Mobile Apps; 3.3 Business Logic; 4.1 Storage; 4.2.1 Advanced Analytics and Machine Learning; 5.2 IoT Gateway Layer; 5.2.1 IoT Gateway Data Management	<b>2.0</b>
21.08.2018	<b>IMA</b>	Contributions in chapters: 3.2 Web and Mobile Apps;	<b>2.0</b>
24.08.2018	<b>CGI</b>	Contributions in chapters: 5.1.1 IoT Backend Data Management, 5.1.2 IoT Backend Device Lifecycle Management; 5.2.1 IoT Gateway Data Management; 5.2.2 IoT Gateway Logic	<b>2.0</b>
12.09.2018	<b>Hi Iberia</b>	Contributions in chapters: 3.2 Web and Mobile Apps; 4.1 Storage; 4.2.1 Advanced Analytics and Machine Learning	<b>2.0</b>
24.09.2018	<b>DEKPROJECT</b>	Contribution in chapter 3.2 Web and Mobile Apps	<b>3.0</b>
25.09.2018	<b>VTT</b>	Contribution in chapter 4.2.1. Advanced Analytics and Machine Learning	<b>3.0</b>
19.10.2018	<b>BEIA</b>	Contributions in chapters 5.1. IoT Backend Layer; 5.2. IoT Gateway Layer; 5.2.1. IoT Gateway Data Management	<b>3.0</b>
31.10.2018	<b>IMA</b>	Contributions in chapters 3.3 Business Logic; 5.1.1 IoT Backend Data Management; 5.1.3. IoT Backend Protocol Adapters	<b>3.0</b>
31.10.2018	<b>Prodevelop</b>	Contributions in chapters 2. Define ESTABLISH Generic components; 3.2 Web and Mobile Apps; 4.1.1 Storage for indoor air quality management; 4.2.2. Complex event processing; 5.2. Web Services; 5.3. OpenData	<b>4.0</b>
01.11.2018	<b>CGI</b>	Contributions in chapters 3.3 Business Logic; 5.1. IoT Backend Layer; 5.1.3. IoT Backend Protocol Adapters; 5.2.3. IoT Gateway Protocol Adapters	<b>4.0</b>
05.11.2018	<b>Beia</b>	Contributions in chapters 3.2 Web and Mobile Apps; 5.1.1 IoT Backend Data Management; 5.1.3 IoT Backend Protocol Adapters; 5.2.2 IoT Gateway Logic; 5.2.3 IoT Gateway Protocol Adapters	<b>4.1</b>
09.11.2018	<b>Turkgen</b>	Contributions in chapters 3.4 Business System Integration; 4.1.2 Storage for Tracking of Professional / non-professional athletes with wearable sensors; 4.2.1 Advanced Analytics and Machine Learning; 4.2.2 Complex event processing; 4.2.3 Event mediator	<b>4.2</b>
10.12.2018	<b>SIVECO</b>	Technical revision	<b>4.3</b>
11.12.2018	<b>SIVECO</b>	Contribution in Chapter 6.5 and final revision	<b>5</b>
17.12.2018	<b>SIVECO</b>	Feedback implementation	<b>6</b>
18.12.2018	<b>SIVECO</b>	Final revision	<b>7</b>

## Table of Content

Version history.....	2
Table of Content.....	3
1. Introduction.....	5
2. Define ESTABLISH Generic components.....	6
3. Application and Presentation Layer.....	13
3.1 Presentation.....	13
3.2 Web and Mobile Apps.....	15
3.3 Business Logic.....	27
3.4 Business System Integration.....	30
4. Knowledge and Storage Layer.....	31
4.1 Storage.....	31
4.2 Knowledge layer - Analytics and processing.....	43
5 Acquisition / Interconnection Layer.....	51
5.1. IoT Backend Layer.....	51
5.2 IoT Gateway Layer.....	54
5.3 Web Services.....	58
5.4 OpenData.....	59
5.5 SOA Architecture “Decision support tools for behavioural choices and treatment options”.....	60
5.6 SOA Architecture layers.....	60
6 Constraints.....	62
Optimized City and Mobility Planning.....	62
Developing smart HVAC systems that ensure a healthy indoor environment.....	63
Promoting independence of specific vulnerable groups.....	65
7 Conclusions.....	69
8 References.....	70

Figure 1.	ESTABLISH Logical View (architecture Layers) .....	7
Figure 2.	ESTABLISH Development View mapped on Logical View .....	9
Figure 3.	ESTABLISH Architecture, process view .....	11
Figure 4.	ESTABLISH Architecture, physical view .....	12
Figure 5.	EVIF architecture .....	14
Figure 6.	MVC Architecture .....	16
Figure 7.	Architectural design principle Model-View-Controller (MVC) .....	17
Figure 8.	React Virtual DOM .....	20
Figure 9.	Kibana charts .....	22
Figure 10.	Grafana chart .....	22
Figure 11.	The typical structure of a mobile application.....	23
Figure 12.	IoTLoRaWAN Web Application .....	26
Figure 13.	Main layers.....	28
Figure 14.	The indoor air quality system architecture .....	31
Figure 15.	IAQ Sensors for Indoor Air Quality Management.....	32
Figure 16.	Data management server and storage .....	33
Figure 17.	Database schema.....	33
Figure 18.	Flexible storage architecture, optimising MongoDB for unique application demands.....	37
Figure 19.	Data management server and storage .....	39
Figure 20.	Data transformation formulas .....	42
Figure 21.	Architecture of Indoor air quality analytics .....	43
Figure 22.	Real time analysis form .....	46
Figure 23.	CEP overview .....	47
Figure 24.	Event processing .....	49
Figure 25.	Rule definition form.....	50
Figure 26.	Rule application results.....	50
Figure 27.	The ESTABLISH data flow mapped on logical architecture. ....	52
Figure 28.	IoT Gateway Architecture .....	54
Figure 29.	Soa based architecture .....	61
Figure 30.	Azure App Service: Basic Service Plan .....	64
Figure 31.	Azure SQL Database: Single database model.....	64

## 1. Introduction

In WP3 work package the main objective is to define, design and document a flexible, scalable and open architecture based on the user needs and the functional and non-functional system requirements identified in WP2. The design process – based on the technological value chain - will have an enhanced focus on the security and privacy, since the architecture will be used in the health domain. Furthermore, in order to attain a future-proof architecture that allows scalability; a strong emphasis will be put on reusability, modularity and changeability. In addition, the WPs aims to develop an UI (User Interface) and to produce a test plan.

The architecture definition task will specify the high level building blocks for the technical solutions that will be developed in the ESTABLISH project. The detailed design of each component will be performed in the corresponding tasks in subsequent work packages. The architectural specification will contain the description of:

- The decomposition of the system into tiers, layers, components;
- Dependencies between components (external interfaces);
- Finding and binding of components (identification and communication);
- Semantics of the components;
- Integration of open source platforms (e.g. the sensor management framework and suitable APIs);
- Important data structures;
- Synchronization aspects;
- Architectural design patterns and styles;
- Different external run-time support components and tools (servers, interpreters, libraries, virtual machines, etc.);

## 2. Define ESTABLISH Generic components

**ESTABLISH** is a concept used to denote a central part of the system, having similar functionality and similar role for all implementations, but the deployment and even the software components might be different, according with the particular use case or particular hardware infrastructure.

The ESTABLISH platform is organized in the following layers:

- Application and presentation layer
- Knowledge layer
  - a. Storage sub-layer
  - b. Analytics and processing sub-layer
- Acquisition and interconnection layer

In this document **ESTABLISH** is referred and considered the common concept used in each implementation. For particular components and deployments, we are specifically making reference to the elements of each use case or implementation.

The presentation layer is the layer that users will see and with which they will interact, it is also called the graphical user interface (GUI) of the platform and it is very important that it be clear, easy to use and optimized.

The application layer contains the functionality the user interacts with. It is based on web and mobile applications, which use a Model-View-Controller architecture<sup>1</sup>. The MVC architectural pattern has existed for a long time in software engineering.

The knowledge layer consists of all the components used to store data and transform it to information and knowledge. Here is referred the storage sub-layer and Analytics and processing sub-layer.

The Storage sub-layer comprises all SW components able to store any kind of data in the ESTABLISH solution, which uses RDBMS and NoSQL systems.

The Analytics and processing sub-layer consists of components having functionality for

- Data Analytics (including time series processing)
- CEP (Complex Event Processing) engine
- Machine Learning (ML) unsupervised learning

The acquisition and interconnection layer contain all the components used for data acquisition from IoT, wearable or other input devices. It also refers to the whole SOA based architecture used for data access and interconnection.

The architecture of the system is presented considering 4 views, valid for all implementations and deployments:

Logical View

Development View

Process View

Physical View

In the logical view we are presenting the main building blocks of the system. The other views are next mapped on that logical view.

Considering a layered architecture, the logical view refers to:

Application and presentation layer, which is the main interaction point with final clients. It consists of web and mobile applications, based on the EVIF (ESTABLISH Visualisation Framework).

The middle layer, called Knowledge Layer, consists of modules grouped together in the Storage and BackeEnd. They are responsible for data storage and processing, and also as protocol adapters in relation with devices used to collect data.

The acquisition/Interconnection layer is responsible for collecting data from wearable and also from the environmental devices. It is also responsible for exposing API and defining interfaces to which further devices can be linked.

A special attention is considered here for defining the specifications common to all input devices, so that, in the future, other devices for data acquisition can be used.

The layer contains the specifications for drivers to be designed for any other input device which in the future can be plugged in. This is similar the way the printer drivers works for current operating system.

In the operating system there is a layer implementing common specifications for all printers. Then when a new printer is designed and produced, the manufacturer will create a driver implementing the specifications, and the printer can next be used by the host operating system.

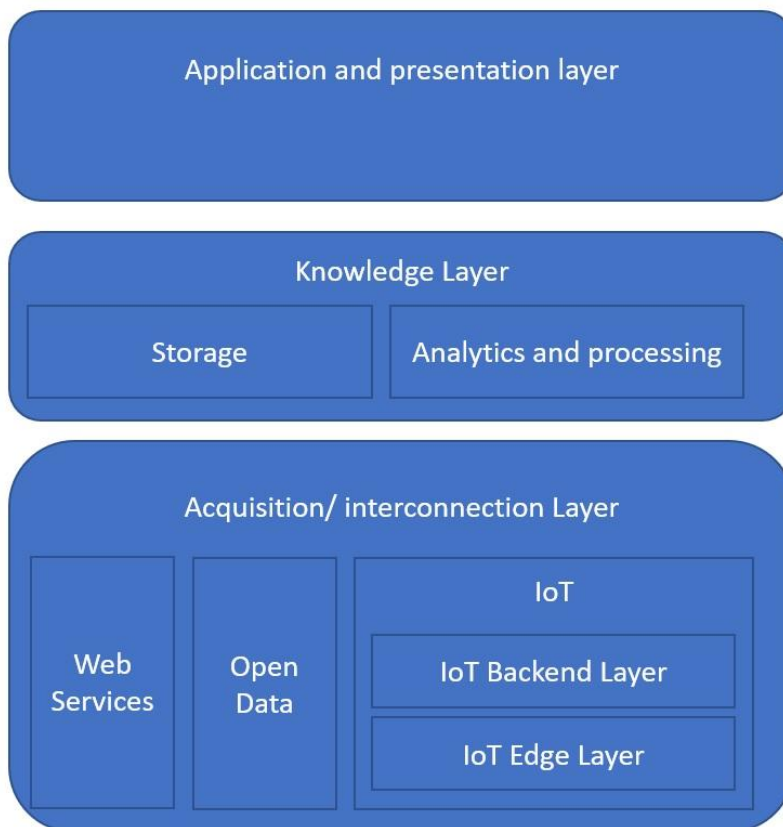


Figure 1. ESTABLISH Logical View (architecture Layers)

The development architecture view maps the software components on the logical view. It also explains how work packages will consider different software components.

On the application, an presentation layer, covered by WP5, the main components considered are:

- EVIF (ESTABLISH Visual Framework) which is at the basis of all UI
- Mobile applications
- WEB applications (Client side)
- Reports and Graphs
- Analysis visualisation tools
- User interaction system

On the Knowledge layer, considered in WP4 and WP5, we are distinguishing three main parts:

- Backend support system composed of
  - CEP (Complex Event Processing)
  - ML (Machine Learning)
  - Big Data Analytics
  - Storage (JPA based)
  - Search Engine
- ESTABLISH Backend Components
  - Data pre-processing (Normalisation, Filtering)
  - IoT Data Process
  - Wearable Data Process
  - Life Cycle Management
  - History and Audit
- Protocol adapters. They are components in the backend responsible for adapting the drivers of input devices to the specifications of ESTABLISH. This is a very important part of the system, as it allows system extension with any other devices. This part is implementing a common way of treating input data, and considering that in future, the input (IoT or Wearable) will implement the common specifications, data will be easy gathered. The protocol adapters developed are for
  - Wearable(FITBIT)
  - IoT (BEIA and ETRI)
  - Specifications for drivers, and Interface implementation

The acquisition and interconnection layer, considered in WP4, is implementing components for wearable and IoT used by ESTABLISH. They are

- Wearable components: FITBIT
- IoT components: BEIA and ETRI
- Wearable and IoT logic
- Wearable and IoT data management

Establish system is offering also an API for external application, which allows the access to process data (Reports, commands) but also to raw data from IoT and wearable.



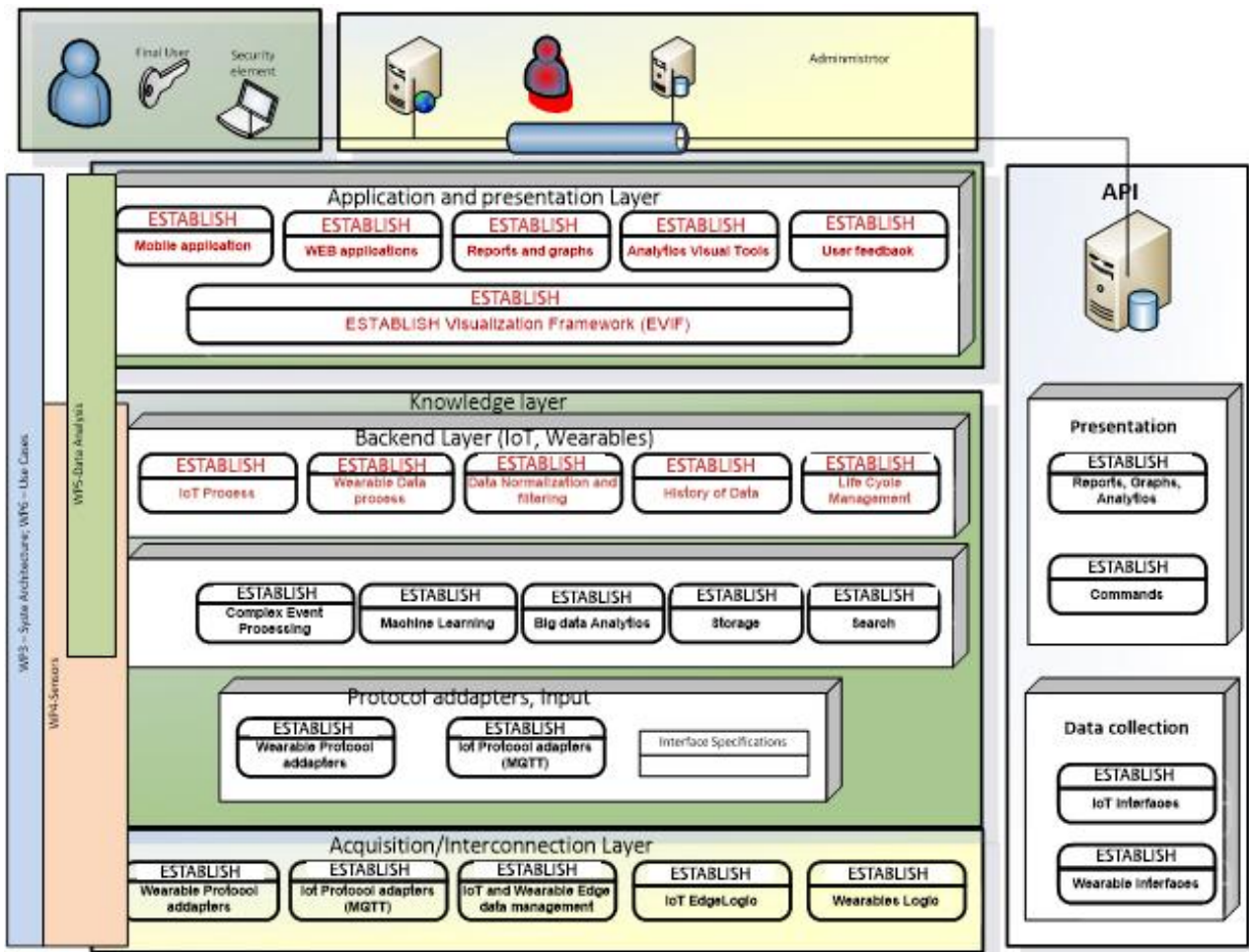
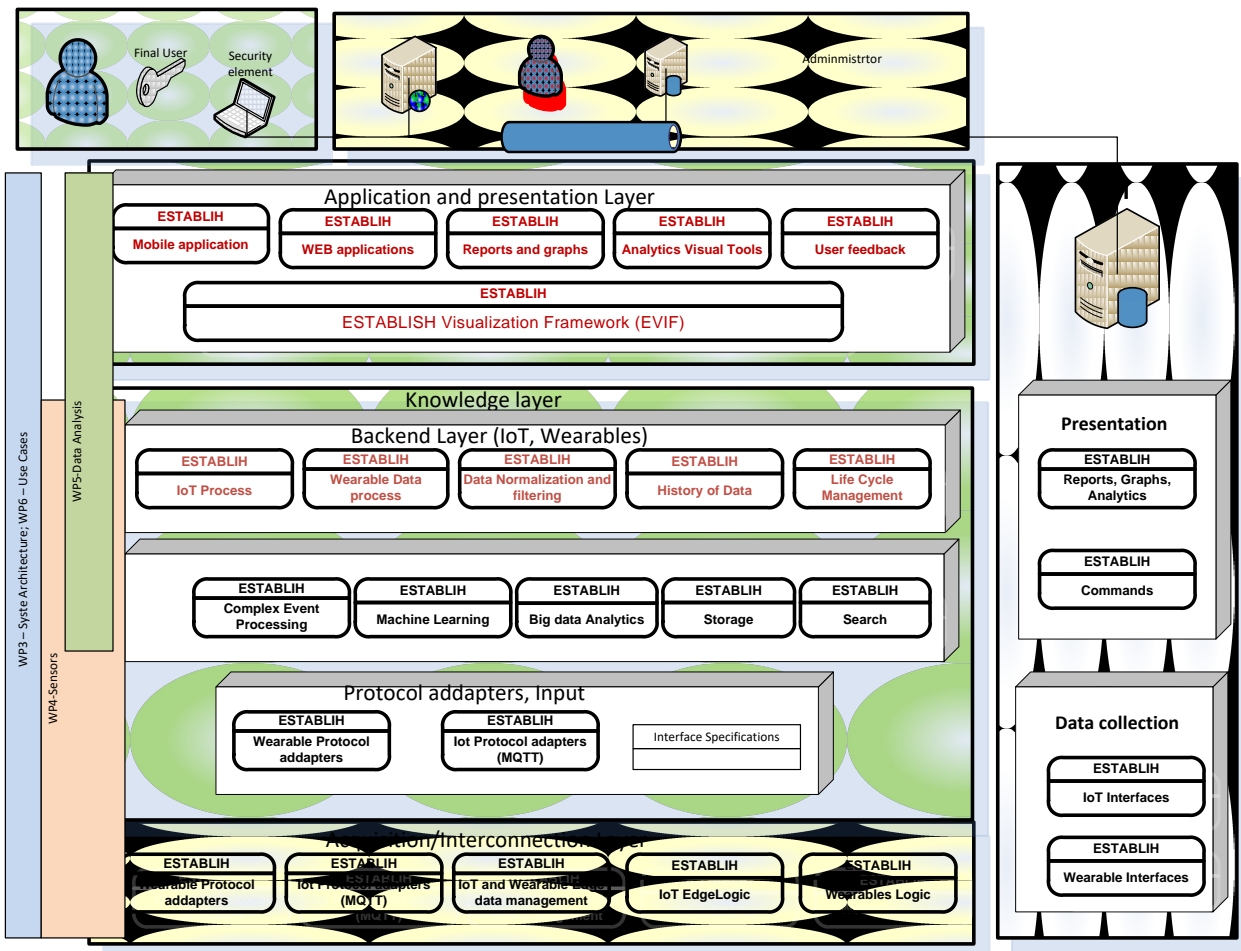


Figure 2. ESTABLISH Development View mapped on Logical View



The process view explains how the main process of ESTABLISH are followed

This is described as:

First data is collected from wearable or IoT. Then it is preprocess (filtered, normalized, fill gaps, etc)

From normalized data, the defined features are computed. Next the features are processed by several engines (CEP, ML, Decision Support). Finally the end user can use the output of decision support mechanism to take actions, or to plan actions.

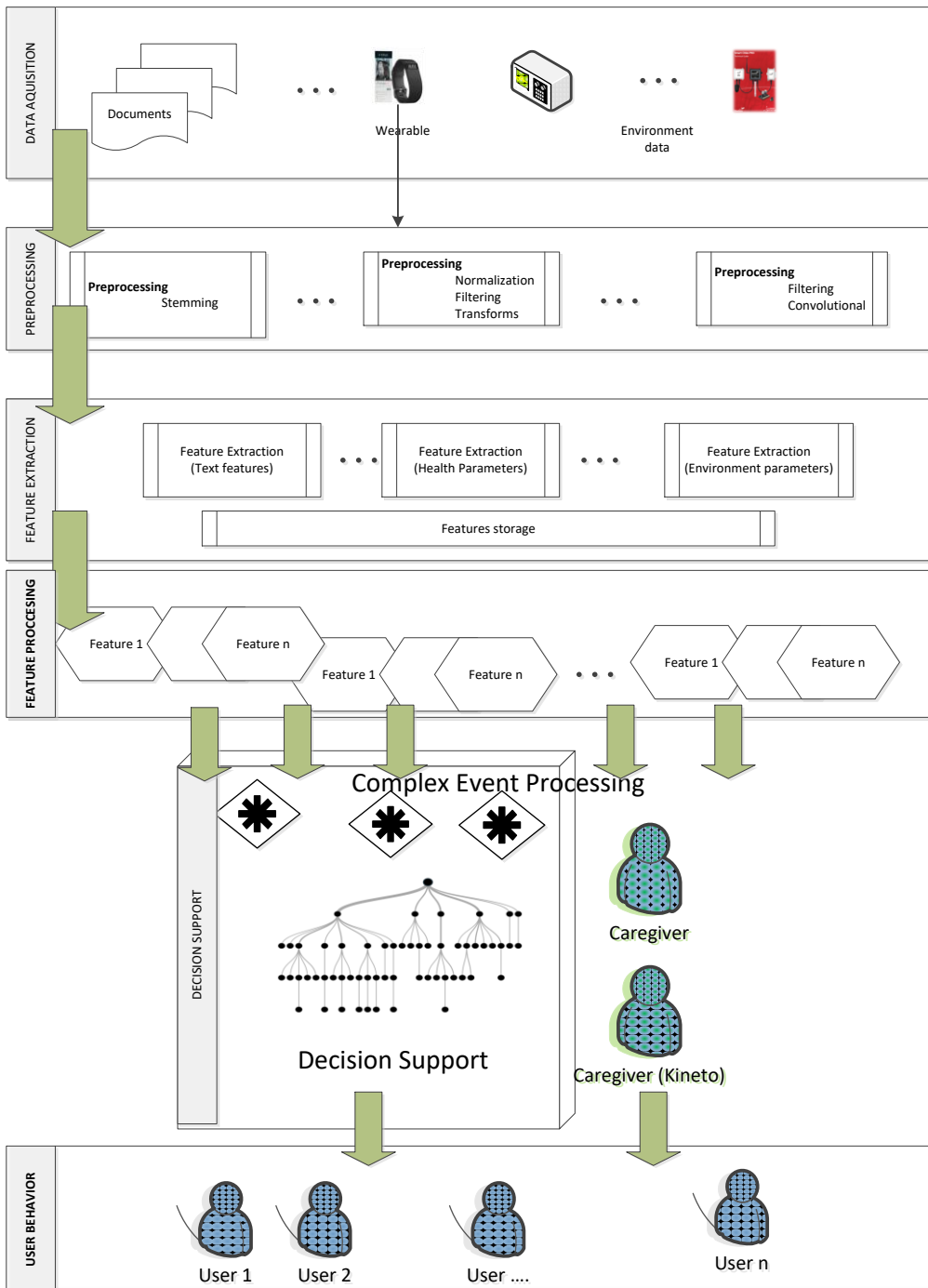


Figure 3. ESTABLISH Architecture, process view

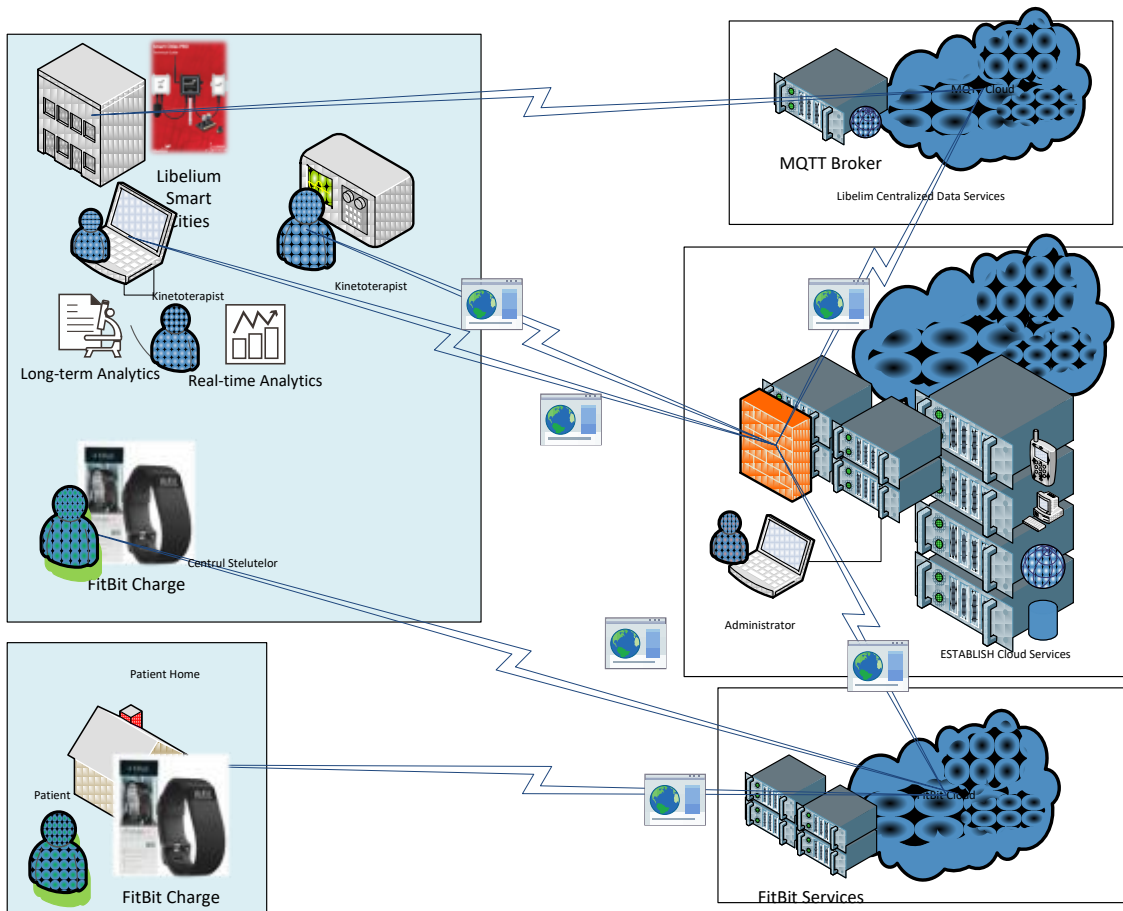


Figure 4. ESTABLISH Architecture, physical view

In the physical view, an example of deployment on physical component is presented, It is just an example, as the system can be deployed also in other configurations to be defined on each pilot. The example fits the Rehabilitation decision support pilot

## 3. Application and Presentation Layer

### 3.1 Presentation

The presentation layer is the layer that users will see and with which they will interact, it is also called the graphical user interface (GUI) of the platform and it is very important that it be clear, easy to use and optimized. We have approached responsive web design and implementation as current SOA standards at the time of development. This approach will be beneficial in terms of compatibility and scalability, making ESTABLISH optimized and usable on different devices and web browsers. The user interface is based on Bootstrap, a HTML5/CSS3 front-end web framework. Other technologies like Primefaces (JSF based), jQuery, jQuery UI and Modernizr could also be used to complete and extend the framework. The presentation layer of ESTABLISH consist of several components. In a broad sense, these can be divided to targeted use-case focused UI and general component used across the use-case focused UIs. Both of these are indispensable because the use-case focused UI provides low-threshold and highly intuitive interface to the end-users. The general components typically have more complex general functionality, which however is less use-case specific, and thus at least from the configuration perspective has a bit higher threshold to use.

ESTABLISH Visualization Framework (EVIF) represents one of the main general purpose components that can be used across the use-cases. EVIF targets highly customizable visualizations and customized reports. EVIF features visualization widgets (e.g. line chart, bar-chart, legend, 3D building browser), that can be composed to create complex visualization. The creation of the visualization is performed via a web-based administrative interface that EVIF provides.

The high-level use-case pattern for EVIF is thus as follows: (1) EVIF web-based administration interface is used to develop targeted visualizations (e.g. historical comparison of temperature/humidity data across all classrooms in a school), (2) URL of the panel showing the visualization is embedded in use-case specific UI. The end-user thus perceives only the use-case specific UI in which EVIF's advanced visualizations seamlessly embedded.

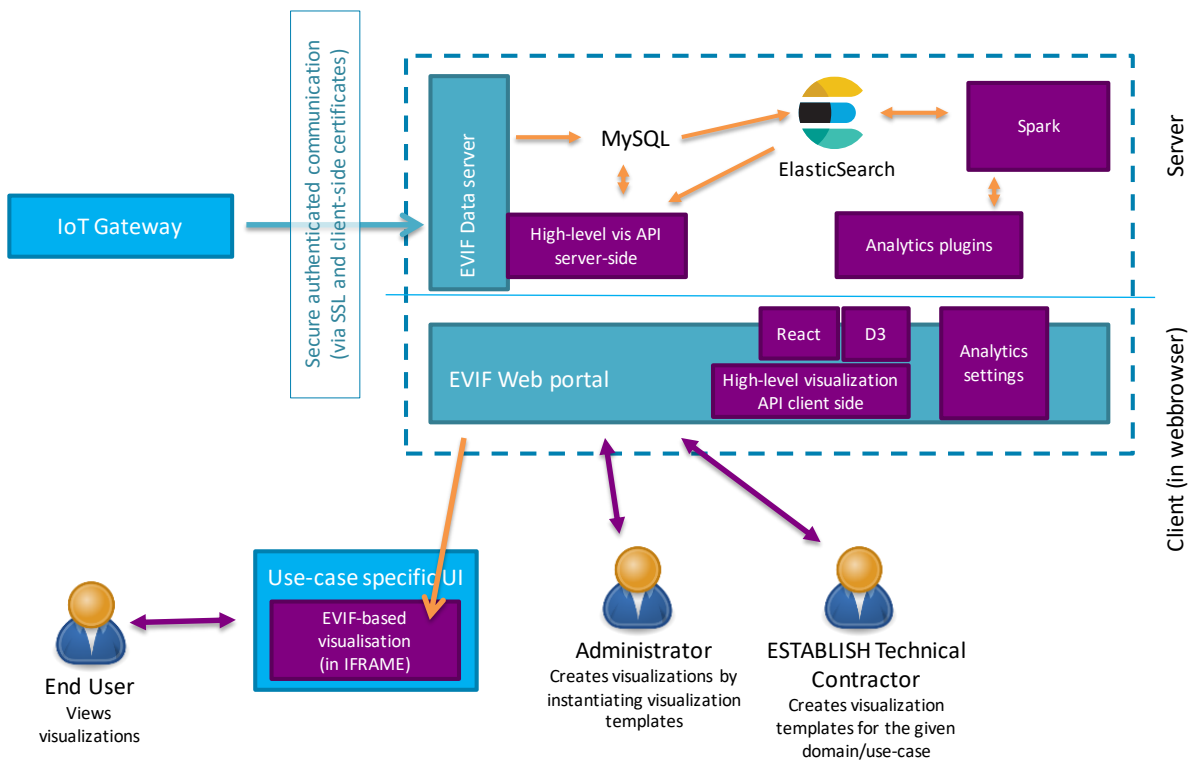


Figure 5. EVIF architecture

From the technical point of view, the architecture of EVIF is as shown in the figure above. EVIF provides (1) a server that takes care of sensor data indexation and post-processing, and (2) a web-based client that visualizes the data provided by the server. Internally, EVIF is implemented in Javascript (both client and server). For visualisations on the client side, it uses ReactJS and D3.

From the business perspective, EVIF recognizes three main types of users (shown in the figure).

The Technical Contractor is responsible for setting up visualization templates in which he/she exploits EVIF's visualization widgets. These widgets are glued together by simple Javascript code which is entered via EVIF's web-based administration UI. The technical contractor is required to have basic knowledge of Javascript, HTML and CSS.

The Administrator is responsible for configuring and deploying panels. A panel in EVIF's terminology is an instantiation of a visualization template (previously provided by the Technical contractor) over a particular set of sensor data. For example, the Technical contractor prepares a template to display comparison of temperature, humidity and energy consumption across several school rooms. The Administrator then configures several panels – a panel per rooms on the same floor, a panel per rooms on the same side of the building, etc. The Administrator is not required to have knowledge of HTML or any programming skills. He/she is assumed only to be a computer power-user.

The End user (e.g. building maintenance) views EVIF's panels through the use-case specific UI to get insights into which rooms exhibit abnormal behavior (e.g. because of broken gasket in windows).

EVIF significantly simplifies the development of visualization templates by providing a growing number of reusable widgets. At this point the list includes: line chart, bar chart, pie chart, legend, time range selector, data access. Widgets which are further envisioned include navigation widgets (3D building browser, floor plan), and statistics charts (XY chart, violin plot, box plot).

EVIF visualization framework (developed by CUNI) internally uses two types of storage: (1) permanent master storage for signal data, and (2) temporary storage for fast data visualization. The master storage is implemented over a MySQL database. It stores sensor data obtained from IoT gateways or other ESTABLISH components (e.g. from LORAWAN portal – developed by IMA). The temporary storage is implemented over Elasticsearch. It indexes all data from the master storage and further keeps derived data (i.e. data computed based on sensor readings via different analytics) – e.g. pre-computed future trends, filtered and smoothed data for displaying seasonal trends.

The visualization requires a large number of aggregates (this is to avoid transferring all data to the client, but only pre-computed points which are enough to draw a respective chart). This involves a lot of computation that needs to be performed at real time. Elasticsearch is a solution specifically intended to compute such aggregates in a highly scalable cluster.

It is an open-source, distributed, multi-tenant search engine implemented in Java. There are clients available for most of commonly used programming languages (including Javascript). It has an HTTP interface and works with schema-free JSON documents. Based on the DB-engines ranking<sup>2</sup>, it is the most popular search engine and is used in many services (such as GitHub, Netflix, etc.).

As it offers near real-time searches, it can be considered as a distributed noSQL database however it lacks distributed transactions. This is however not an issue as in EVIF, it is used primarily for data retrieval.

A single Elasticsearch instance is called a node. A set of connected nodes forms a cluster. The nodes can be of different types – depending on their functionality (Master-eligible node, Data node, Ingest node, or Tribe node). Internally, Elasticsearch is based on Apache Lucene (information retrieval library). Via it, Elasticsearch provides indexing of full text and fuzzy string searching. The indices are divided to shards, which are distributed with their replicas over the nodes. Compared to traditional databases (SQL), an index corresponds to a database, a type of a documents corresponds to a table, a document to a row and a field in a document to a column. For queries, Elasticsearch provides own JSON-based DSL (`{ "filter" : { "term" : { "id" : "12345" } } }`). There is also an experimental support for executing actual SQL queries against Elasticsearch indices.

Elasticsearch further features integration with Hadoop via a bi-directional connector (called ES-Hadoop). In particular, it has direct support for Spark framework which is used for performing cluster computation over large sets of data and thus provides a convenient way of performing scalable data analytics. In EVIF, the intended use is to perform analytics as Spark jobs over sensor data stored in Elasticsearch. The analytics results are stored back in Elasticsearch from which they can include in visualizations in the same way as original sensor data.

The choice of Elasticsearch thus not only brings a convenient API for computing the aggregates, but also addresses to a great deal the scalability of EVIF and provides a platform that is ready for including data analytics (themselves described in Section 4.2).

### 3.2 Web and Mobile Apps

Web and mobile applications use a Model-View-Controller architecture<sup>3</sup>. The MVC architectural pattern has existed for a long time in software engineering. All most all the languages use MVC with slight variation, but conceptually it remains the same. MVC stands for Model, View and Controller. MVC separates application into three components - Model, View and Controller.



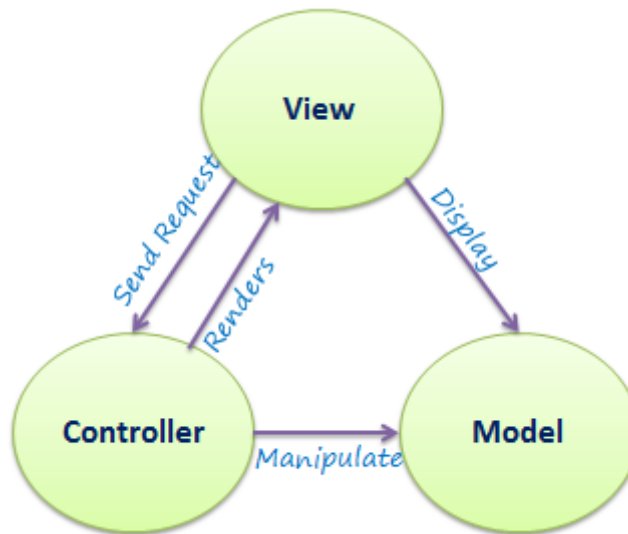


Figure 6. *MVC Architecture*

**Model:** Model represents shape of the data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database.

**View:** View is a user interface. View display data using model to the user and also enables them to modify the data.

**Controller:** Controller handles the user request. Typically, user interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response.

The following figure illustrates the interaction between Model, View and Controller implemented on the server side.



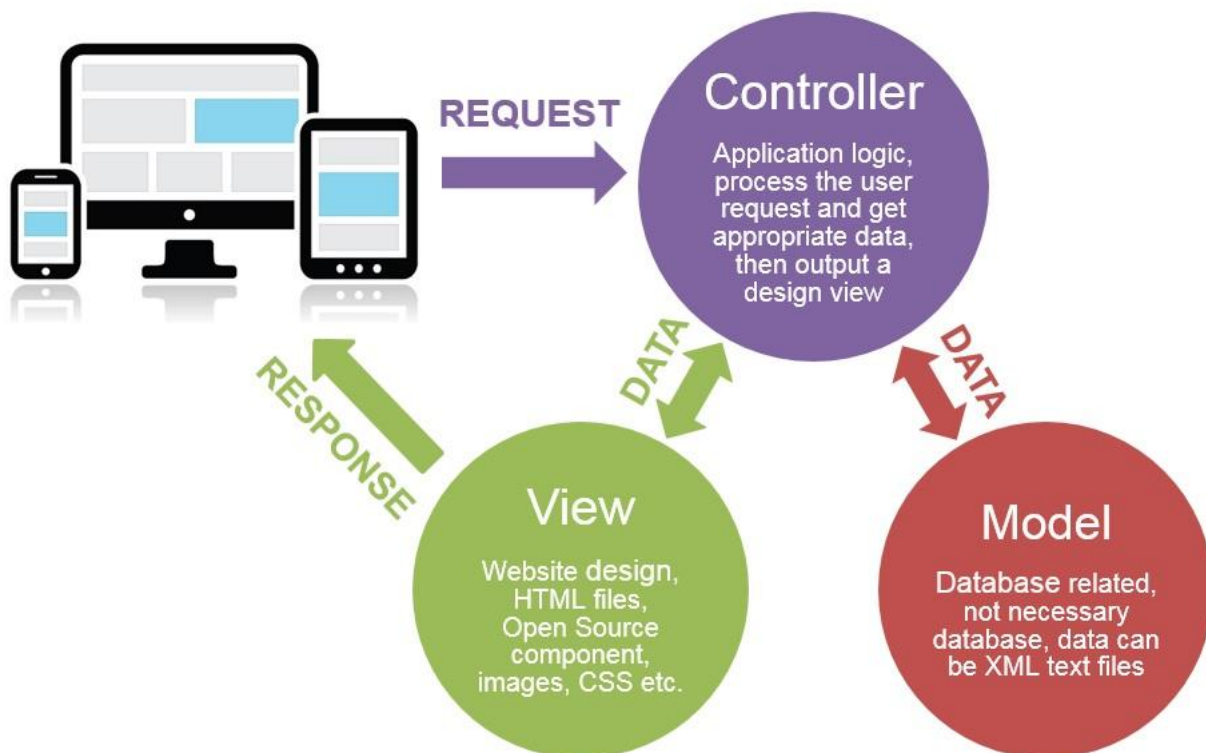


Figure 7. Architectural design principle Model-View-Controller (MVC)

Model deals with data. All the codes and queries related to data retrieve or insert, update, delete are placed in Models.

No business logic, algorithm or queries are suggested to place in Views. Only data that are needed and codes that represents those data as user expectation (e.g: html tags in web apps) are kept in a View.

Controller is responsible to coordinate with Model and View. All the business logics, algorithms and functions are placed in Controllers. When a user request something from View, it's handled by the Controller first. If the request needs data from the database, controller retrieves it from the database through Model and then throws it to View after necessary process. By this process user can view the response according to his/her request.

When developing the IoT system then it's an immense need to consider about its maintenance, version controlling and future development.

With MVC pattern, codes are easy to adorn since all are distributed according to category. So, the coding standard can be maintained precisely.

It's easier to review or change all the codes when it comes to deal with requirement or version changes or future development since the codes are already distributed in Model-View-Controller.

It is important to modularize all the features inside when the project is large. There are cases where all the features are not important for all the users. In those cases, Model View and Controllers are created according to particular modules so that it can be easier to give or revoke features as per the user need.

A level of system security can be maintained by following the MVC pattern. Though there are so many things to consider for ensuring system security, MVC helps developer control access violation by

restricting user from accessing Model directly from View. So data access and manipulation is bit secure here. In the meantime, database queries, all the logics and algorithms according to software features and all other codes are arrange precisely which ensures the code security.

### 3.2.1 Web application candidate technologies and frameworks

#### **BACKEND FRAMEWORK:**

ESTABLISH Backend is a complex system responsible for

- Data acquisition from devices
- Data processing
- Rule engine
- ML system
- Analysis and presentation system
- API

It is a web application, running on top of TomEE server. [10]

The backend is also the place for deployment of the data models

The data Model contains the structures necessary to cover all the processes described.

We are grouping the individual components of the data model in the following groups”

- User account management
- Patients and caregivers
- Activities
- Collected data
- Rule engine and ML
- Nomenclatures

Access to data model is using JPA. The backend server is using EJB technologies. Security is using OAuth 2.0 specifications.

Rule engine is based on DB stored procedures

ML is using clustering algorithms, and is developed in Java

Analysis and presentation is using Grafana

APIs are based on REST services.

#### **FRONTEND FRAMEWORK:**

**Bootstrap** is a responsive front-end web framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Bootstrap offers a large OS (operating system) and browser compatibility pool as described in the tables below:

	Chrome	Firefox	Safari	Android Browser & WebView	Microsoft Edge
<b>Android</b>	Supported	Supported	N/A	Android v5.0+ supported	N/A
<b>iOS</b>	Supported	Supported	Supported	N/A	N/A
<b>Windows 10 Mobile</b>	N/A	N/A	N/A	N/A	Supported

Table 1: Bootstrap mobile browser compatibility

	Chrome	Firefox	Internet Explorer	Microsoft Edge	Opera	Safari
<b>Mac</b>	Supported	Supported	N/A	N/A	Supported	Supported
<b>Windows</b>	Supported	Supported	Supported, IE10+	Supported	Supported	Not supported

Table 2: Bootstrap desktop browser compatibility

The current stable version is Bootstrap 4.0. By using Bootstrap web framework we can ensure both modern browsers compatibility (eg. IE Edge +, Firefox, Safari and Chrome), and high flexibility in design component changes, and the responsive design approach.

**jQuery** is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today, with installation on 65% of the top 10 million highest-trafficked sites on the Web.

**jQuery UI** is a collection of GUI (graphical user interface) widgets, animated visual effects, and themes implemented with jQuery, Cascading Style Sheets, and HTML. According to JavaScript analytics service, Libscore, jQuery UI is used on over 197,000 of the top one million websites, making it the second most popular JavaScript library.

**Modernizr** is a JavaScript library which is designed to detect HTML5 and CSS3 features in various browsers, which lets JavaScript avoid using unimplemented features or use a workaround such as a shim to emulate them. Modernizr aims to provide this feature detection in a complete and standardized manner.

**PrimeFaces** is an open-source user interface (UI) component library for JavaServer Faces-based applications. PrimeFaces is a lightweight library, all decisions made are based on keeping PrimeFaces as lightweight as possible. Usually adding a third-party solution could bring an overhead however this is not the case with PrimeFaces. It is just one single jar with no dependencies and nothing to configure. (Prime Faces, 2018)

**CSS3 (Cascading Style Sheets)** is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

**HTML5** is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and current version of the HTML standard. It was published in October 2014 by the World Wide Web Consortium (W3C) to improve the language with support for the latest multimedia, while keeping it

both easily readable by humans and consistently understood by computers and devices such as web browsers, parsers, etc. HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML. HTML5 includes detailed processing models to encourage more interoperable implementations; it extends, improves and rationalizes the mark-up available for documents, and introduces mark-up and application programming interfaces (APIs) for complex web applications. For the same reasons, HTML5 is also a candidate for cross-platform mobile applications, because it includes features designed with low-powered devices in mind.

The technologies presented here will ensure an optimal format for content to be available through commonly used web browsers, e.g. IE, Chrome, Firefox, in commonly used document formats, e.g. doc, xls, ppt, pdf, and in compatibility with commonly used operating systems including mobile technology, e.g. Windows, MAC, OS, iOS, Android, and Windows mobile.

**D3 visualization library**<sup>4</sup>, which is a data-driven document visualization library, is a JavaScript library that provides interactive visualization of data on web pages. Using D3 with HTML, CSS and another JavaScript, the developers are able to display data from different sources in advanced graphics and create infographs with SVG objects. In other words, adding d3 functions to graphical objects like SVG objects providing them with rich and dynamic features such as scaling, events, transitions, and animations.

**React**<sup>5</sup> is a JavaScript library for building interactive user interfaces (UIs) that is developed by Facebook, Instagram and community. The design of UIs using React follows “Model-View-Controller” (MVC) pattern and is component-based in which components are typically written in JavaScript extension (JSX). Hence, the components are declarative which work as reusable APIs that build a representation of the needed view. Therefore, it displays data dynamically without reloading the whole page by rendering only the components that are effected by data changes. More specifically, React introduces the concept of virtual DOM (**Error! Reference source not found.**), or virtual Document Object Model, that manages the browser’s DOM for the developers in order to improve performance. Thus, React computes the minimal needed changes using virtual DOM and then re-renders the browser’s DOM.

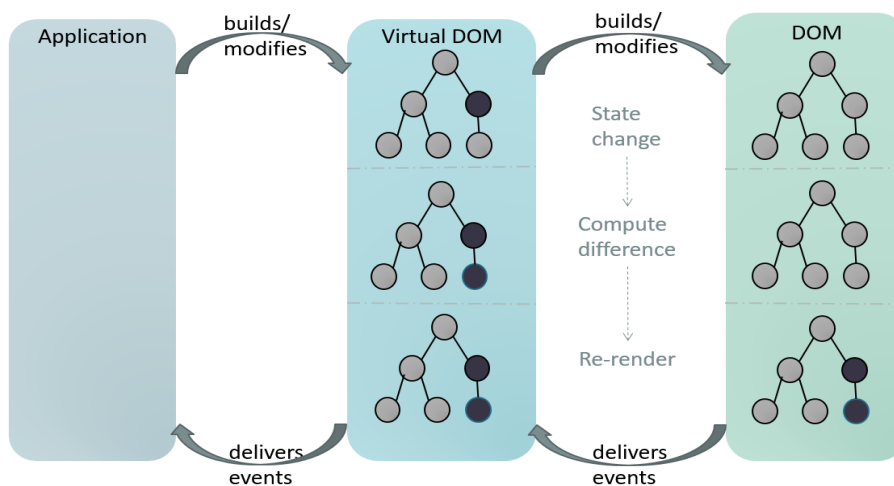


Figure 8. *React Virtual DOM*

**Vue.js**<sup>6</sup> is an open-source JavaScript front-end framework for building user interfaces. Integration into projects that use other JavaScript libraries is simplified with Vue because it is designed to be

incrementally adoptable. Vue can also function as a web application framework capable of powering advanced single-page applications.

The project focuses on making ideas in web UI development (components, declarative UI, hot-reloading, time-travel debugging, etc.) more approachable. It attempts to be less opinionated and thus easier for developers to pick up.

It features an incrementally adoptable architecture. The core library focuses on declarative rendering and component composition and can be embedded into existing pages. Advanced features required for complex applications such as routing, state management and build tooling are offered via officially maintained supporting libraries and packages

**JSX**<sup>7</sup> is inline HTML markup that is used to declare components in React applications. It is faster in running on web browsers than JavaScript because of its generated code, safer since it is statically-typed object oriented language, and easier than JavaScript.

## SECURITY

**OAuth**<sup>8</sup> 2.0 is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords. This mechanism is used by companies such as Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites.

Generally, OAuth provides to clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The third party then uses the access token to access the protected resources hosted by the resource server

## SEARCH AND ANALYTICS

**Elasticsearch** is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases, and designed for horizontal scalability, maximum reliability, and easy management. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.

## VISUALISATION

**Kibana** is a powerful visualisation tool. Kibana gives you the freedom to select the way you give shape to your data. And you don't always have to know what you're looking for. With its interactive visualizations, start with one question and see where it leads you.

Kibana core ships with the classics: histograms, line graphs, pie charts, sunbursts, and more. They leverage the full aggregation capabilities of Elasticsearch. Kibana developer tools offer powerful ways to help developers interact with the Elastic Stack.



Figure 9. Kibana charts

**Grafana**<sup>9</sup> is an open-source visualization tool that can be used on top of a variety of data stores such as Graphite, InfluxDB, and also Elasticsearch. Grafana is a feature-rich replacement for Graphite-web, and through its unique Graphite target parser can produce various types of dashboards with easy metrics and function editing.



Figure 10. Grafana chart

### 3.1.2 Mobile application candidate technology and frameworks

A mobile application is usually structured as a multi-layered application consisting of presentation, business, and data layers. The below figure illustrates a common rich client mobile application architecture with components grouped by areas of concern.





Figure 11. *The typical structure of a mobile application*<sup>10</sup>

When developing a mobile application, it is possible to choose developing a thin Web-based client or a rich client. For a rich client, the business and data services layers are likely to be located on the device itself; and for a thin client, such layers are located on the server.

A mobile application generally contains user interface components in the presentation layer, and perhaps may include presentation logic components. The business layer, if it exists, usually contains business logic components, any business workflow and business entity components that are required by the application, and, optionally, a facade. The data layer usually includes data access and service agent components.

Mobile applications can be developed based on the following approaches:

- A **Fully Native** approach uses Apple and Google original tools respectively to specifically target the mobile device. They are developed stand alone and use different languages and development environments in each context.
- A **Hybrid Cross Platform** development framework is a set of development, configuration and build tools that allow standard web technologies (HTML 5, CSS 3 and Javascript) to be packaged and deployed onto a mobile device. Typically, the assets will be uploaded to the Apple store for IOS or the Play store for Android. At run-time, the application uses the mobile device browser (web view).
- A **Native Cross Platform** development framework offers the same as the Hybrid approach. However, it cuts HTML and CSS from the equation and allows the packaged app to talk directly to the mobile Operating System. This offers two advantages over the hybrid platform: the apps can use native User Interface controls (rather than mimicked ones), that makes development easier since there is consistency in devices; and the applications will potentially run faster (since there is less processing to perform).

Due to differences in the underlying technology, each approach has inherent advantages and drawbacks, development frameworks, and appropriate use cases, so a careful analysis is required to ensure that an application is built using the right technology for the functionality required. The following table summarizes the different approaches:

Development Approach	Fully Native	Native Cross platform	Hybrid Cross Platform
<b>Definition and Tools</b>	Build the app using native frameworks: <ul style="list-style-type: none"> <li>• iPhone SDK</li> <li>• Android SDK</li> <li>• Windows Phone SDK</li> </ul>	Build once, deploy on multiple platforms as native apps: <ul style="list-style-type: none"> <li>• React Native</li> <li>• Angular 2 Nativescript</li> <li>• Xamarin</li> </ul>	Building using standard web technologies: <ul style="list-style-type: none"> <li>• Apache Cordova</li> <li>• PhoneGap</li> <li>• Ionic</li> </ul>

<b>Underlying Technology</b>	<ul style="list-style-type: none"> <li>• iPhone: Swift/Objective C</li> <li>• Android: Java</li> <li>• Windows Phone: C#/Visual Basic</li> </ul>	<ul style="list-style-type: none"> <li>• React Native: Javascript</li> <li>• Angular 2 Nativescript: Javascript/Typescript</li> <li>• Xamarin: C#</li> </ul>	<ul style="list-style-type: none"> <li>• Apache Cordova: HTML5, CSS3, Javascript</li> <li>• PhoneGap: HTML5, CSS3, Javascript</li> <li>• Ionic: HTML5, CSS3, Javascript</li> </ul>
<b>Deployment</b>	<p>App stores</p>	<p>App stores</p>	<p>Over the web</p>
<b>Key Use Cases</b>	<ul style="list-style-type: none"> <li>• Apps requiring high-end user experience, more transactional in nature</li> <li>• Large user base on one device <ul style="list-style-type: none"> <li>• Offline usage</li> </ul> </li> <li>• Apps requiring extensive device and/or OS functions</li> </ul>	<ul style="list-style-type: none"> <li>• Simpler apps, more informational in nature <ul style="list-style-type: none"> <li>• Offline Usage</li> </ul> </li> <li>• Multiple device types distributed across key users</li> <li>• Works well for a number of enterprise applications that do not require heavy device functions</li> </ul>	<ul style="list-style-type: none"> <li>• Generic user experience, performance depends on Internet connection</li> <li>• Distributed user base across smart phone platforms</li> <li>• Need to maintain single code base</li> <li>• Moderate amount of device functions</li> </ul>

Table 1. Development approach for mobile applications

## FULLY NATIVE FRAMEWORK

### Swift/Objective C (iOS)

**Objective C** has been the dominant language to develop for iOS for many years. It has a lot of support on the devices in terms of libraries and, at the moment, the majority of applications are written in it.

**Swift**<sup>11</sup> first appeared in 2014 and superseded Objective C for iOS devices. Development using both languages is carried out using OSX using XCode.

### Java (Android)

**Java** can be developed using Android Studio<sup>12</sup> to target Google's Android operating system. It provides a very good developer experience and great tooling and support.

### C# (Windows Phone)

C# or Visual Basic can be used alongside Microsoft own tools<sup>13</sup> to create windows phone apps.

## NATIVE CROSS PLATFORM FRAMEWORK

### React Native (Supported devices: iOS, Android)

**React.js**<sup>14</sup> is an open source Single Page Application (SPA) framework that allows web developers the ability to build large-scale JavaScript applications for the browser. It composes common application



requirements (such as rendering views) into a single framework that can be leveraged for productivity. **React Native**<sup>15</sup> is an extension of react that removes browser specific features and introduces mobile specific knowledge. Facebook are key contributors to this project.

*Angular 2 Nativescript (Supported devices: iOS, Android)*

The newest release of the most popular SPA Angular 2 has built in support to Teleriks Nativescript. Although developed by Telerik, **Nativescript**<sup>16</sup> is open source. It allows development to target mobile devices using JavaScript or Typescript<sup>17</sup>. The benefit is that Angular 2 is a very mature platform.

*Xamarin (Supported devices: iOS, Android, Windows Phone)*

**Xamarin**<sup>18</sup> uses C# to target multiple devices. It comes with a powerful Integrated Development Environment (IDE's) and works well with Microsoft Visual Studio (via an extension).

## HYBRID CROSS PLATFORM FRAMEWORK

*Apache Cordova*

**Apache Cordova**<sup>19</sup> is a development framework that is command line driven, it includes build tools that will take an input of (HTML, CSS and Javascript) and will produce something that runs on a mobile device.

*Phonegap (built on Cordova)*

**Phonegap**<sup>20</sup> is a development framework created by Adobe that is built on top of Apache Cordova. The main benefit is the integration with the Adobe universe. For example, Adobe offers build tools that run in the cloud.

*Ionic (built on Cordova)*

**Ionic**<sup>21</sup> is another framework that is built directly on top of Apache Cordova. Ionics integrates the well-known Single Page Application framework called Angular. Ionic is open source, so this means that the focus is on the product itself (as opposed to the services offered around it like with Phonegap).

Web application IoTLoRaWAN monitors the status of the environment using the LoRa wireless network.

Effectively monitors:

- Environmental features
- CO2, temperature, humidity, dust, signal strength...
- Flow of water and air
- Sewerage, piping, air conditioning, chimneys...
- Existence of waste disposal facilities
- Dump stations, landfills, septic tanks, reservoirs...
- Remote readings of electronic meters
- Gas meters, electricity meters, water meters...
- Number, movement and behavior of people
- Attendance, pervasiveness, overcrowding...



Figure 12. *IoTL0RaWAN Web Application*

Get the data you need to optimize:

- **Buildings and cities**  
Setting thermostat according to real demand, searching of insufficient volume of ventilation, warning of critical values of toxic substances in air, soil or water...
- **Alert of critical utility utilization values**  
Rapidly falling water in the water, clogged sewerage, clogged chimney...
- **Waste removal frequency, septic tank cleaning etc.**
- **Quantification of the cost of the service**  
Readings of gas meters, water meters ...
- **Security risks involved with the occurrence and movement of people**  
Cancellation of unused stops, adjustment of timetables, changes opening hours and so on.

### 3D building browser

A web application running on server: the 3D building browser visualizes a building as a 3D BIM model (imported from IFC format). It allows a user to navigate in the building (by panning and zooming the camera and by sliding building floors). This way, it allows a user to locate sensors in the building, select them and investigate their data. This complements the traditional list view of sensors by allowing users to locate a sensor by the intuitive understanding of approximate sensor location. Internally, the application uses BIMsurfer, which it integrates in EVIF. The application accesses the browser via REST API based on JSON.

### *Mobile application for citizens*

This mobile application intends to provide citizens with useful predictions and recommendations about mobility and pollution in the city in real time, which is produced by the cloud platform for data fusion and data analysis; and it also allows them to have access to a route planner focused on calculating the most ecological route considering the municipal open data.

This mobile application is an Android native application developed with the Android SDK framework in Java, which acts as the front-end side of the cloud platform for data fusion and data analysis and the multi-modal route planner.

The inputs of the mobile application are the results (recommendations and predictions) derived from the cloud platform for data fusion and data analysis, and the results produced by the multi-modal route planner when the user requests a certain route; while the outputs are the user requests to get a certain route.

### *Web application for city authorities*

This web application intends to help the city authorities in the decision-making about the urban mobility planning in terms of pollution levels, by allowing them to predict the traffic loads and the pollutions and analyse configurable situations/scenarios related to mobility and pollution through a traffic simulation platform.

This web application is developed with the Foundation framework, while the main technologies used are HTML5, CSS3 and JavaScript; and it acts as the front-end side of the traffic simulation platform. Furthermore, it is required that this web application must have an easy-to-use user interface in order to present the information in a comprehensible way and should be time responsiveness for basic user interactions.

The inputs of the web application are the results derived from the traffic simulation platform; while the outputs are the user requests to carry out different simulations.

## 3.3 Business Logic

The business logic layer is a layer of code that implements the functionalities of the system. This layer is responsible for accessing, processing and transformation of data. Additionally it manages the business rules and assures the data consistency and accuracy. The business logic layer is accessed from the presentation layer to make the functionalities available to the users and it can also offer the functionalities to external information systems through the data exchange interfaces.

The business logic layer has following characteristics:

- It is completely independent from the presentation layer and from external applications that use the data exchange interfaces.
- It has a completely modular architecture based on reusable components and abstract interfaces. There must be no identical functions made by different components (e.g. data access).
- It contains and delimit the “business workflow” and “business entity” components.
- Access to business entity components will be done through business workflow components.
- The business entities are clearly defined at the business logic layer.
- Business entity components contain all data and business logic related to the business entity, for undertaking the business operations, implementation of relevant business rules and for the maintenance of the integrity and accuracy of contained data.

- The components related to business logic layer communicate through dedicated interfaces/ internal functions (tight coupling).
- The components of the business logic layer are accessible for external applications only through the data exchange interfaces.
- The architecture of the business logic layer is allowing simultaneous access to the functionalities of the system.

Reference e.g. <https://objcsharp.wordpress.com/2013/07/22/what-is-a-business-logic-layer-anyway/> and/or

[https://www.ibm.com/support/knowledgecenter/en/SSZLC2\\_7.0.0/com.ibm.commerce.developer.doc/concepts/csdbusinesslogicbase.htm](https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdbusinesslogicbase.htm)

Business logic layer visualized. The blue boxes make up the user interfaces of the entire system.

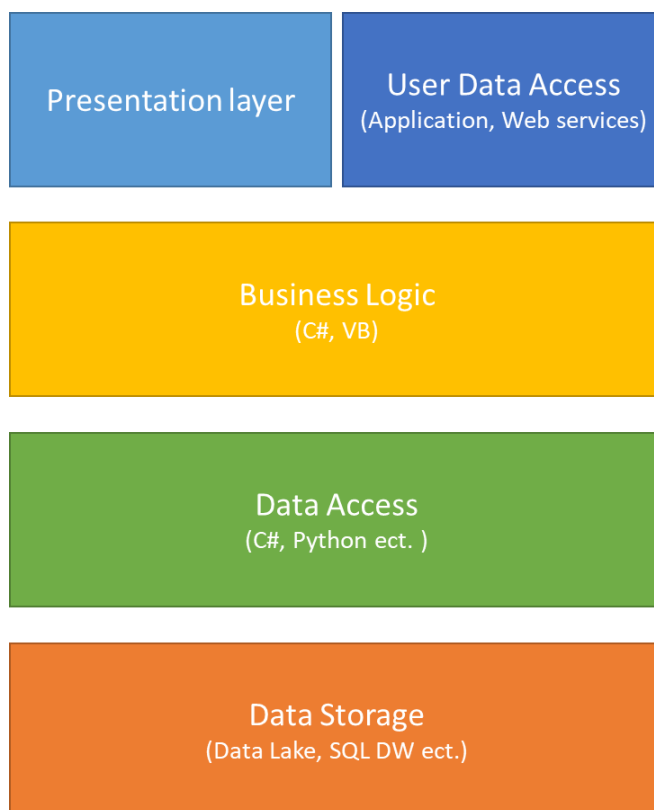


Figure 13. *Main layers*

In the Establish project the business logic layer has three major components:

1. Web portal logic written in Java
2. A service sub-layer consisting in an instance of Elastic Search, a search engine working on large databases ([www.elastic.co](http://www.elastic.co))
3. Mobile app RESTful service, designed to push data to the UI of the mobile application

The components above mentioned will work with business records such patient and physical trainer data, therapeutical programs and assigned or recommended activities, notification rules and conditional static and behavioural, localisation algorithms, tracking and historical data management.

Monitoring the quality of the internal environment has always been a priority for IMA s.r.o. since its establishment in 1992.

The quality of the internal environment is a critical factor in both public and private sectors (schools, offices, residential buildings and family homes).

We measure and evaluate a variety of variables that determine the quality of the indoor environment. The most important ones are temperature (T), humidity (H) and especially carbon dioxide (CO<sub>2</sub>). The highest acceptable value of carbon dioxide for public buildings is set to 1500 ppm. However, it is known that even a slightly elevated carbon dioxide values of about 900-1000 ppm already result in fatigue, headaches and reduced overall body activity.

#### **IMA activities:**

- Testing, measuring and evaluating the results of the deployed IoT sensors
- Daily operation
- LORAWAN mobile app created by IMA
- ImaWAN information brochure

#### **Tests and measurements**

IMA has been testing several different types of IoT sensors focused on measuring the internal environment – temperature (T), humidity (H), levels of carbon dioxide (CO<sub>2</sub>), airflow (A) and received signal strength (RSSI). We've deployed two sensors to primary schools in Prague in March 2018 as a part of the InAirQ project.

Objectives of the tests and measurements:

- To verify the expected development of air quality depending on the number of people present
- To compare the results of common IoT sensors with laboratory devices

A positive conclusion was achieved as both main objectives have been met, and therefore confirmed the possibility of real-life deployment.

#### **From development to business**

Our systems and applications are designed as open, so we have the option to choose different technologies. We can connect with any IoT sensor; and as for the transmission networks itself – we're able to communicate over LoRa, SigFox and NB IoT networks. Furthermore, we have an open custom-made LoraWAN / imaWAN application that allows us to receive and visualize data and evaluate it. Real-life applications and use cases are the primary building block of our entire system and the representatives of the Czech Republic have always been positively evaluated.

Based on our testing experience regarding actual business, we have been more focused on public buildings, especially primary schools and nurseries. Several primary school directors expressed their interest in IoT sensors measuring the internal environment. However, some of those orders have been postponed due to the lack of funding; despite the sensors' reasonable pricing. That is why we had to expand our business activities to other customers and segments.

We negotiated with a major insurance company about the possible deployment of IoT sensors in households – to measure CO<sub>2</sub> levels, a flooding alarm and a basic security system - door opening sensor and motion detection sensor.

We also offered our IoT sensors to Czech Technical University in Prague and Brno University of Technology - to monitor the internal environment in terms of study, research and tests in various types of projects.

Currently, we are testing the networks for temperature and humidity measurement in a pharmaceuticals storage segment.

We also offer monitoring and evaluation of the quality of the indoor environment and ventilation control in public agencies and hospitals.

### **Use case on the CEZ Group premises**

To measure and evaluate the current status of the indoor environment related to ventilation systems is a frequent requirement.

We installed IoT sensors to measure the quality of the indoor environment on the premises of our major client – the CEZ Group. We measure and evaluate the levels of T+H+ CO<sub>2</sub> on a long-term basis. The sensors had to be equipped with their own batteries, as our client wanted them to be mobile. Therefore our IoT sensors came out as the best possible solution.

Our outputs were used to negotiate with the owner and administrator of the building to improve the quality of the environment, where the real values of T+H+CO<sub>2</sub> were being measured and graphically evaluated over a long-term period.

Airflow and temperature IoT sensors are being installed in ventilation ducts at this time.

Furthermore, we are negotiating about the possible use of IoT sensors for electric meters, water meters and gas meters.

## **3.4 Business System Integration**

In the Establish project there are many different use cases according to Countries. One of the important steps is Business System Integration that will ensure the alignment of the different use cases in one platform. Business integration is a simple idea: all the use cases working together, so Establish may be considered as one solution for different project ideas. The projects that are on the platform, can communicate with each other.

What any good integration platform will do is acting as a data translator between different use cases. The platform will simply translate data from one file format's "language" to another.

## 4. Knowledge and Storage Layer

### 4.1 Storage

This sub-layer comprises all SW components able to store any kind of data in the ESTABLISH solution.

Particularly for the “Optimized City and Mobility Planning”, the following SW components for data storage are used:

- A NoSQL database (**MongoDB, Elasticsearch, Cassandra**) to store the following information:
  - Historical data coming from the open data sources
  - Recommendations about mobility and pollution
    - Predictions about mobility and pollution
- A SQL database (**PostgreSQL**) to store all those personal information related to the user.

Thereby, the stored data in the NoSQL database are the results (recommendations and predictions) derived from the cloud platform for data fusion and data analysis and the historical data coming from the open data sources; and the stored data in the SQL database are the user information entered from the mobile application.

Finally, it is important to mention that both databases can be accessed through a REST API operating over them.

#### 4.1.1 Storage for indoor air quality management

The indoor air quality system consists of the following basic goals in order to keep indoor environment healthy through environmental data analysis.

- IAQ/OAQ sensor device
- Data management server
- Analysis and adaptive control system

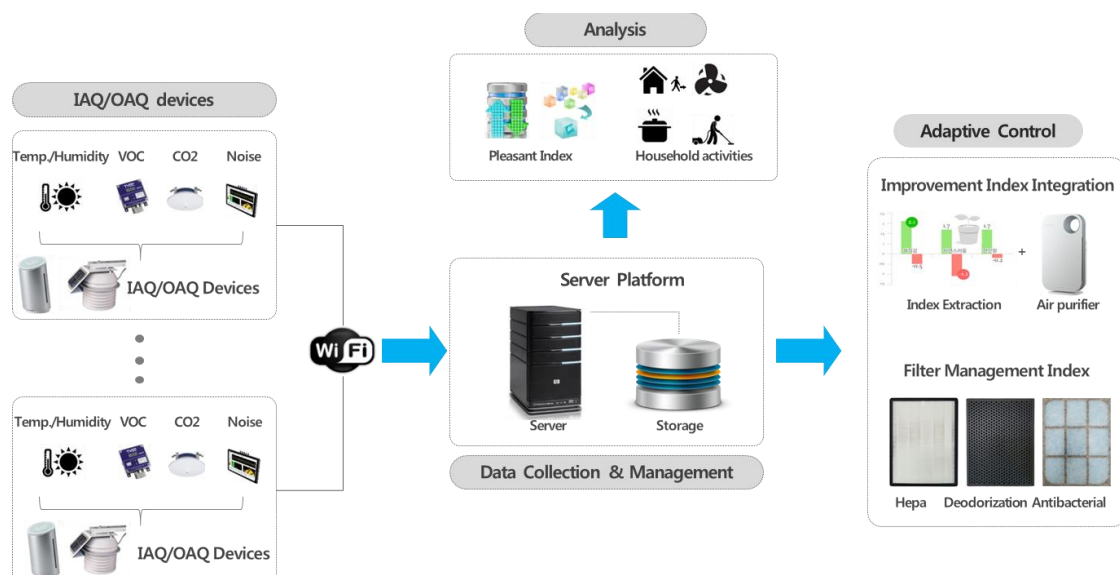


Figure 14. *The indoor air quality system architecture*

The key technologies for achieving the goals

- Efficient battery charging for OAQ sensor devices



- Low power consumption, minimum maintenance
- Flexible design of OAQ sensor devices for enduring tough outdoor weather
- Two-way communication between sensor devices and the backend server
- Data server system for managing environmental data
- Data analysis system
- Adaptive control of air purifier

IAQ and OAQ devices integrate many kinds of sensors to measure the various environmental information into one hardware module. The devices also include the functions of the wireless communication to transmit the measured information. Each IAQ sensor device for indoor air quality measurements includes temperature, humidity, CO<sub>2</sub>, illuminance, noise, VOC (volatile organic compounds), Formaldehyde and dust sensors. An OAQ sensor device for outdoor air quality measurements includes temperature, humidity and dust sensors. The OAQ sensor devices use solar energy as its battery charging and they are designed to endure the tough outdoor weather.

The following figure shows indoor the specification of the air quality sensors:





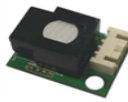


Type	CO <sub>2</sub>	illuminance	Noise	VOC	Formaldehyde	Dust	Humidity/ Temperature
Product Picture							
Measurement Range	0~2000 ppm	10~1,000 Lx	-42dB	1~10ppm	0~500ppb	0~500ug/m <sup>3</sup>	-40~125°C 0~100%RH
Accuracy	<±50ppm+ 2% of measuring value	-	±3dB	Coway's Algorithm	<±30%@Full Range	±15%	±0.3°C ±2%
Interface	I2C	Analogue	Analogue	Analogue	I2C	UART	I2C

Figure 15. IAQ Sensors for Indoor Air Quality Management

The data management server collects data from IAQ/OAQ sensor devices and stores them to its storage. It is implemented with Spring framework over Apache Tomcat for HTTP processing and PostgreSQL for environmental data storage. An IAQ sensor communicates with its subordinate OAQ sensor through Radio Frequency (RF) to gather both IAQ and OAQ environmental data and send it to the backend data server via Transmission Control Protocol (TCP). The backend server platform communicates with the IAQ sensors via TCP in order to collect and store the data. In addition, the server provides data access service with JSON query format describing various query constraints over HTTP. The server uses Spring framework over Apache Tomcat for HTTP processing and PostgreSQL for environmental data storage.



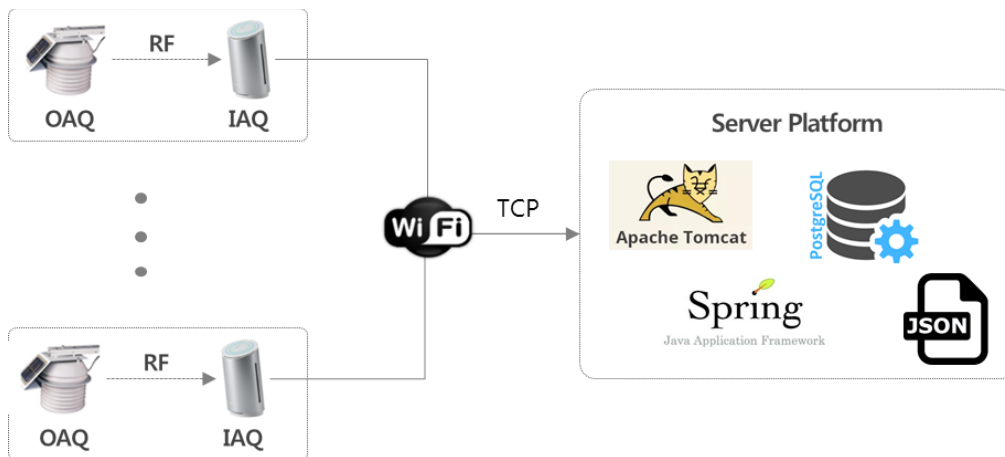


Figure 16. *Data management server and storage*

The system uses PostgreSQL for its data storage and some of database entities are as follows:

- building: this entity represents a building in which IAQ devices are installed.
- indoor\_map: this entity represents the indoor map of a building
- iaq: this entity represents a IAQ device
- sensor: this entity represents the sensors belonged to a IAQ device

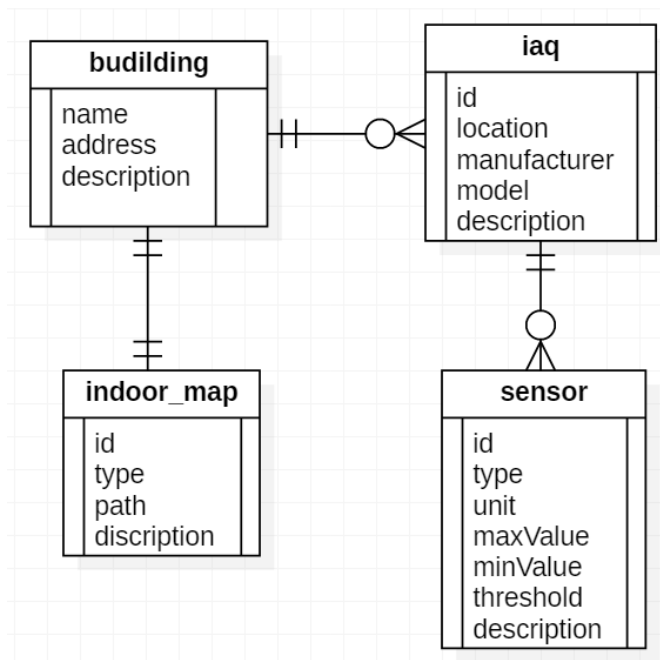


Figure 17. *Database schema*

Entity	Column	Description
<b>building</b>	name	the name of the building
	address	the physical address of the building
<b>indoor_map</b>	id	identifier of the map
	type	type of the map (ex, jpg, png)
	path	directory of the map
<b>iaq</b>	id	identifier of the IAQ device
	location	the location of the IAQ device on the indoor map of the building
	manufacture	the manufacture of the IAQ device
	model	the model of the IAQ device
<b>sensor</b>	id	identifier of the sensor
	type	type of the sensor (ex, particle, co2, ...)
	unit	unit of the sensor (ex, ug/m3, lux, dB, ...)
	maxValue	maximum value of the sensor
	minValue	minimum value of the sensor
	threshold	threshold value of the sensor

Knowledge and storage layer is about storing and organizing data. Due the specific of the project data acquisition part, in Establish data will be stored in two ways. One is the big data component, which will be used to store industrial data organised as time-series and gathered from sensors and other detection devices. The other one is the ore classical approach of a relational database management system (RDBMS), used to store the relational data processed and stored inside the Establish solution. Of course both storage layer components will communicate with the business logic layer and its sub-components.

- noSQL component

First, because of large amount of diversified time series generated at a high speed by industrial equipment as sensors and other devices, we have an **industrial big data** component, through what we name nowadays as the **Internet of things**. Big data refers to data generated in high volume, high variety, and high velocity that require new technologies of processing to enable better decision making, knowledge discovery and process optimization. However, industrial big data should be also visible, which refers to the discovery of unexpected insights of the existing assets and/or processes and in this way transferring invisible knowledge to visible values. The most important characteristic of big data is the obtained value. That implies that, due to the risks and impacts industry might face, the requirements for analytical accuracy in industrial big data is much higher than other analytics, such as social media and customer behaviour.

Industrial big data is usually more structured, more correlated, more orderly in time and more ready for analytics. This is because industrial data is generated by automated equipment and processes, where the environment and operations are more controlled and human involvement is reduced to minimum.

Proliferation of structural, semi-structural and no-structural data, has challenged the scalability, flexibility and processability of the traditional relational database management systems (RDBMS). The next generation systems demand horizontal scaling by distributing data over autonomously addable nodes to a running system. For schema flexibility, they also want to process and store different data formats along the sequence factor in the data. NoSQL approaches are solutions to these, hence big data solutions are vital nowadays. But in monitoring scenarios sensors transmit the data continuously over certain intervals of time and temporal factor is the main property of the data. Therefore the key research aspect is to investigate schema flexibility and temporal data integration aspects together. We need to know that: what data modelling should we adopt for a data driven real-time scenario; that we could store the data effectively and evolve the schema accordingly during data integration in NoSQL environments without losing big data advantages. We need to build a middleware based schema model to support the temporal oriented storage of real-time data from sensors as hierarchical documents. We also need to adopt a schema for the data integration by using an algorithm based approach for flexible evolution of the model for a document oriented database, i.e, MongoDB. The proposed model must be logical, compact for storage and seamlessly evolving upon new data integration.

IoT ecosystems want general storage mechanisms having structural flexibility to accept different data formats arriving from a variety of sensory objects. The non-relational or NoSQL databases are schema-free and allow storage of different data formats without prior structural declarations. However for the storage we need to investigate the NoSQL models to design and develop besides flexibly preserving the big data timestamped characteristics for the massive real-time data flow during acquisition processes. Although all NoSQL databases have unique advantages, but document-oriented storage, as MongoDB provides, is considered robust for handling multiple structural information to support IoT goals. This rejects the relational structural storage and favours Java Script Object Notations (JSON) documents to support dynamic schemas; hence provide integration to different data types besides scalability features.

- Time-series

The Establish system collects a large amount of data from several sensors and/or IoT gateways and from other external interfaces. This significant volume of data should be recorded in a storage capacity that allows fast data processing, almost in real time. The most important volume of data is managed as a time-series structure. A **time series** is made of discreet measurements at timed intervals. The time series pattern is a write optimization pattern made to ensure maximum write performance throughput for a typical analytics application that stores data in discrete units of time, as is the measurement of the temperature over a specific time interval.

- Time-series in medical data

Healthcare monitoring systems measure physiological and biological body parameters, using BAN, of a patient's body in real-time. Because timely information is an important factor to detect immediate situations and to improve decision making processes, based on a patient's medical history, so considering temporal aspects are vital. Such sequence of values represent the history of an operational context and is helpful in a number of use cases where history or order is required during the analysis. This sequences of data flows in streams of different speeds and also needs proper management.

- Big data management frameworks

A big data management framework means the organization of the information according to the principles and practices that would yield high schema flexibility, scalability and processing of the huge volumes of data, but for which traditional RDBMSs are not well suited and becomes impractical. Therefore, there is a need to devise new data models and technologies that can handle such big data. Recent research efforts have shown that big data management frameworks can be classified into three layers that consist of file systems, database technology, and programming models. However in this article we shall focus upon database technologies only in context of the healthcare domain with real-time and temporal perspective.

- NoSQL database categories

NoSQL also be interpreted as the abbreviation of “NOT ONLY SQL” or “no SQL at all” . Based on the differences in the respective data models, NoSQL databases can be organized into following basic categories as: key-value stores, document databases, column-oriented databases and graph databases

- Document databases

These are the most general models, which use use JSON (JavaScript Object Notation) or BSON (Binary JSON) format to represent and store the data structures as documents for the data management. Document stores provide schema flexibility by allowing arbitrarily complex documents, i.e. sub-documents within document or sub-documents; and documents as lists. A database comprises one or more collections, where each collection is a named group of documents. A document can be a simple or complex value, a set of attribute-value pairs, which can comprise simple values, lists, and even nested sub documents. Documents are schema-flexible, as one can alter the schema at the run time hence providing flexibility to the programmers to save an object instances in different formats, thus supporting polymorphism at the database level.

These databases store and manage volumes of collections of textual documents (e.g. emails, web pages, text file books), semi-structure, as well as no structure and de-normalized data; that would require extensive usage of null values as in RDBMS. Unlike key-value stores, the document databases support secondary indexes on sub-documents to allow fast searching. They allow horizontal scaling of the data over multiple servers called shards. MongoDB, CouchDB, Couchbase, ReThinkDB, and Cloudant are some of the most popular document-oriented databases. Among these MongoDB is the most popular one due to its efficiency, in memory processing and complex data type features. The other databases such as Couchbase, ReThinkDB, Cloudant and CouchDB do not offer in-memory processing features; although the former three offer a list of data types. MongoDB query languages more like the SQL of RDBMS, so is easy to use for the programmers. MongoDB is good for the dynamic queries, which the other document-oriented databases lack, such as CouchDB or Couchbase. Besides this there are different object relational mapping middlewares available, to define out of the box multiple schemas depending upon the application requirements.

- MongoDB

MongoDB, created by 10gen in 2007, is a document oriented database for today’s applications which are not possible to develop using the traditional relational databases. It is an IoT database which instead of tables (as in RDBMS) provides one or more collection(s) as main storage components consisted upon similar or different JSON or BSON based documents or sub documents. Documents that

tend to share some of the similar structure are organized as collections, which can be created at any time, without predefinitions. A document can simply be considered as a row or instance of an entity in RDBMS, but the difference is that, in MongoDB we can have instances within instances or documents with in documents, even lists or arrays of documents. The types for the attributes of a document can be of any basic data type, such as numbers, strings, dates, arrays or even a sub-document.



Figure 18. Flexible storage architecture, optimising MongoDB for unique application demands

#### Flexible storage architecture, optimising MongoDB for unique application demands

MongoDB provides unique multiple storage engines within a single deployment and automatically manages the movement of data between storage engine technologies using native replication.

It allows to build large-scale, highly available, robust systems and enables different sensors and applications to store their data in a schema flexible manner. There is no database blockage, such as we encounter during alter table commands in RDBMS during schema migrations. However in rare cases, such as during the write-intensive scenarios in master-slave nature of MongoDB there may be blockage at the document level or bottleneck to the system if sharding is not used, but these cases are avoidable. MongoDB enables horizontal scalability because table joins are not as important as they are in the traditional RDBMS. MongoDB provides auto-sharding in which more replica server nodes can easily be added to a system. It is a very fast database and provides indexes not only on the primary attributes rather also on the secondary attributes within the sub-documents even. For the cross comparison analysis between different collections we have different technologies, such as aggregation framework, MapReduce, Hadoop, etc.

- Elasticsearch

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch can be used to search all kinds of documents. It provides scalable search, has near real-time search, and supports multitenancy. Elasticsearch makes all its features available through the JSON and Java API. Elasticsearch supports real-time GET requests, which makes it suitable as a NoSQL datastore,[7] but it lacks distributed transactions.

- Relational component

The database and data management layer in Establish is implemented using the relational database management system (DBMS). The characteristics for this type of knowledge and storage layer are the following:

- the data model implemented at the persistence layer of the system it is following at least the third-normal form for the database design, to reduce the duplication of data and ensure referential integrity.
- the system data model is following the Common Data Modelling with a generic Data Model which consists of generic types of entity like class, relationships, and others.
- the data at the knowledge and storage layer are accessed only through the business logic layer and independent from the business logic layer.
- the data model implemented at knowledge and storage layer it is properly documented. The documentation contains both the technical description of the data (e.g. entity-relationship diagrams, structures of databases, objects in databases, etc.), and the semantic description (association of data structures with business entities and their properties).
- the knowledge and storage layer must assure the integrity and accuracy of data (transaction integrity).
- 

**POSTGRES** is a powerful, open source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).

PostgreSQL supports a large part of the SQL standard and offers many modern features including the following –

- Complex SQL queries
- SQL Sub-selects
- Foreign keys
- Trigger
- Views
- Transactions
- Multiversion concurrency control (MVCC)
- Streaming Replication (as of 9.0)
- Hot Standby (as of 9.0)

PostgreSQL can be extended by the user in many ways. For example by adding new –

- Data types
- Functions
- Operators
- Aggregate functions
- Index methods

PostgreSQL supports four standard procedural languages, which allows the users to write their own code in any of the languages and it can be executed by PostgreSQL database server. These procedural languages are - PL/pgSQL, PL/Tcl, PL/Perl and PL/Python. Besides, other non-standard procedural languages like PL/PHP, PL/V8, PL/Ruby, PL/Java, etc., are also supported.



**MySQL** is developed by Oracle since 2010. In contrast to Microsoft SQL Server and Oracle, MySQL is Open Source. It is provided under the GPL License and is supported by major operating systems like Microsoft Windows, Linux and Mac OS X.

MySQL provides every feature a modern web application needs. In consideration of this fact MySQL is used as database management system. It is used by various big companies like Google, Facebook or CISCO<sup>22</sup>. It works seamlessly with many other applications including PHP or Apache web server and provides a lightweight, fast and scalable database. MySQL provides many modern SQL aspects like prepared statements, views, triggers and user-defined functions. To implement secure connections to the database SSL is supported as well as a build in JSON support is available. Additionally it provides Open Source tools for tasks like Database Design & Modelling, Database Administration or Database Migration. The MySQL environment provides every feature which is need to implement a modern web application and is therefore a good choice for a database management system.

The data storage can generally be divided to more traditional SQL databases and newer no-SQL approaches, which are typically connected with cluster computing. In Smart HVAC pilot, we combine these two approaches as per storage and computation needs of individual ESTABLISH components used in the pilot.

The components, which provides the IoT data collection (via LORA) and end-user UI (LORAWAN developed by IMA) exploit the Azure SQL database. Azure SQL Database is the intelligent, fully managed relational cloud database service that provides the broadest SQL Server engine compatibility, so we migrated our SQL Server databases.

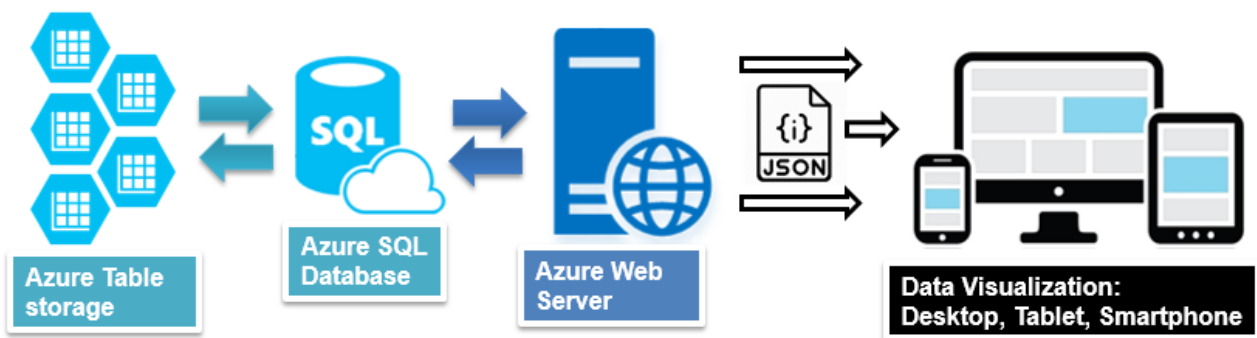


Figure 19. *Data management server and storage*

Data collection takes place automatically at the frequency you choose yourself. The measured values can be viewed as interactive graphs or numerical tables. If you need data for further use (calculations, statistics, analytics, etc.), you can simply export them directly to a CSV, XLS or PDF data file.

#### 4.1.2 Storage for Tracking of Professional / non-professional athletes with wearable sensors

To realize Turkish use case, two parts should be deployed Local Zone and Cloud Zone. Local zone includes some data analytics and data aggregation and manipulation steps, data acquisition from sensors. Data can be collected in local storage mechanisms or data can be stream immediately to Cloud Zone via IoT Gateway. There are Authentication and Authorization mechanisms on the data transfer and monitoring. A REST API will be implemented to data transfer between zones. The API will include Login and several pushData services.

Cloud zone will include some Big Data Analytics Components such as Cassandra, Machine Learning Libraries & Tools, Spark, in-memory databases, Kafka etc. The core data analytics will realize in this site of project. Of course there will be many open source technologies for the monitoring of the result and information delivery to related people such as Apache Web Server, Dashboards, Reporting tools etc.

Data storage layer of Establish solution for Tracking of athletes with wearable sensors pilot includes Cassandra as a NoSQL database.

## Apache Cassandra

Apache Cassandra is a NoSQL database that manages large volumes of structured data on a series of commodity servers. Cassandra is highly distributed, which allows for a high tolerance to node failure, as well as processing large datasets while remaining available to thousands of concurrent users, making it ideal for processing real-time transactional data. However, Cassandra does require its data to be at least partially structured.

### 4.1.3 Storage for Tracking Promoting independence of specific vulnerable groups

For the “**Promoting independence of specific vulnerable groups**” use case, the following SW components for data storage are used:

- A SQL database (PostgreSQL) to store
  - all those personal information related to the users (patients, caregivers)
  - Planned activities for recuperations programs
  - Data coming from IoT
  - Data coming from wearable
- A MQTT Broker user to temporarily store data coming from IoT (Libelium)
- A provider cloud storage for wearable data (FitBit)

A REST API is developed to expose all data managed by the application.

#### *Data collection*

Data collection is the process of retrieving the data from wearable, and from sensors for indoor or for outdoor parameters.

Data for wearable is the data retrieved from fit Bit devices. It is read by accessing FitBit cloud WEB API. ESTABLISH backend is calling the RESTFULL services exposed by FitBit API. Authorization is based on OAuth 2.0 specifications. (OAuth 2.0 official site, 2018)

A typical address is

GET [https://api.fitbit.com/1/user/\[user-id\]/activities/date/\[date\].json](https://api.fitbit.com/1/user/[user-id]/activities/date/[date].json)

The data is in JSON format like in the example presented below:

```
"activities-heart": [
  {
    "dateTime": "2018-11-21",
    "value": {
      "customHeartRateZones": [],
      "heartRateZones": [
        {
          "max": 81,
          "min": 30,

```



```

        "name": "Out of Range"
    },
    {
        "max": 114,
        "min": 81,
        "name": "Fat Burn"
    },
    {
        "max": 138,
        "min": 114,
        "name": "Cardio"
    },
    {
        "max": 220,
        "min": 138,
        "name": "Peak"
    }
}]]].....

```

Data for outdoor sensors is received from devices installed in the location established for the pilot. Data is first stored on a MQTT broker (is sent using WI-FI and MQTT protocol). (MQTT main page, 2018) From the MQTT broker it is accessed by the ESTABLISH backend, on the base of MQTT protocol. ESTABLISH backend is implementing a callback procedure, where the MQTT broker is publishing the data.

Data received from outdoor sensor has the format:

```

Message: {
  "id": "34604091",
  "id_wasp": "SCP5",
  "id_secret": "3D4CA2E80593E4E0",
  "sensor": "TC",
  "value": "1.6699999570847",
  "timestamp": "2018-11-21T17:33:43+02:00"
}

```

Data received from indoor sensors has the format:

```

Message: {
  "id": "34604000",
  "id_wasp": "GAS_WiFi",
  "id_secret": "6B3037057C10549D",
  "sensor": "TC",
  "value": "18.190000534058",
  "timestamp": "2018-11-21T17:23:58+02:00"
}

```

Data collected is the pre-processed as described in the next chapter.

### *Pre-processing*

Pre-processing is necessary in order to have a consistent and uniform set of data, expressed in common used measure units.

The pre-processing of data refers to:

1. Data filtering. Data outside normal values is filtered (considered as erroneous data) and not taken into account for further computations.
2. Data completion. Where data is not present (missing) it is filled in by the average value of last and next registered values.
3. Data transformation. Data computed like in the figure bellow:

### III. Modul de calcul – pentru fiecare poluant

1. Calcul masă volumică la momentul t pentru poluantul x –  $M_x(t)$

$$M_x(t) = 22,45 \cdot \frac{T_m(t) + 273,15}{273,15} \cdot \frac{1013}{P_m(t)}$$

2. Calcul concentrație măsurată la momentul t pentru poluantul x, exprimată în  $\mu\text{g}/\text{Nm}^3$  –  $M_x(t)$

$$C_x(t) = \left[ C_{m,x}(t) \cdot \frac{MM_x}{M_x(t)} \right] * 1000$$

Figure 20. *Data transformation formulas*

## 4.2 Knowledge layer - Analytics and processing

### 4.2.1. Advanced Analytics and Machine Learning

#### *Indoor air quality analytics*

The indoor air quality analytics system classifies the user behaviors and extract user pleasant index with some machine learning algorithms. The analyzed results are applied to air purifiers to clean the indoor air quality in home and office buildings. The algorithms such as the machine learning and data mining find out the indicators to improve the air quality by classifying the user life patterns and detecting the air pollution sources. It can improve the effectiveness and minimize the management cost of air purifiers by notifying the replacement cycle of filter parts, which are the core of the air purifier, through analysing the contamination degree of the environment. The data analysis will exploit machine learning classification algorithms such as SVM(Support Vector Machine), decision trees, etc for detecting user activities.

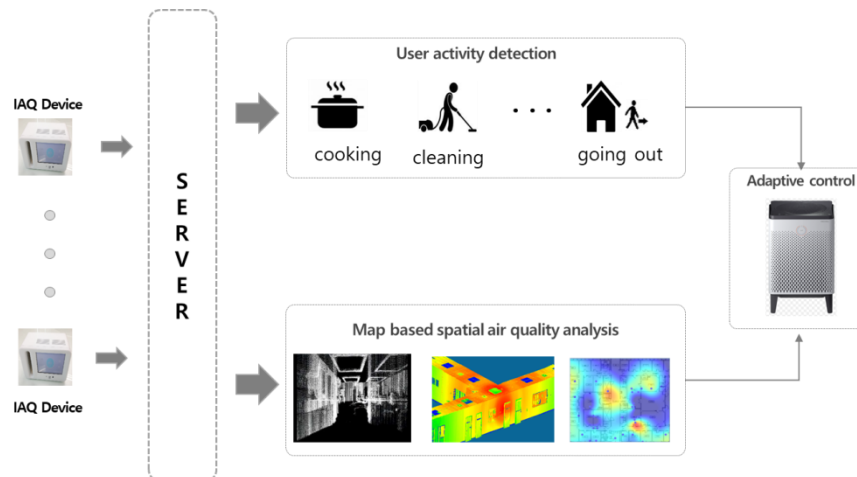


Figure 21. *Architecture of Indoor air quality analytics*

The improvement indicators of the air quality are used for the program to interconnect the intelligent management system of the air quality and the air purifiers. The field test can improve the reliability of the developed system for the commercialization in the practical applications

Advanced analytics will use algorithms to perform statistical analysis of stored time series about the patient behaviours and generate forecasts about these behaviours. Those algorithms will also generate suggestions for the future therapeutical programs and activities that are supposed to be assigned to patients by physical therapists. In this case we will use an **unsupervised machine learning** task by inferring a function that describes the structure of "unlabeled" data (i.e. data that has not been classified or categorized). Since the examples given to the learning algorithm are unlabeled, there is no straightforward way to evaluate the accuracy of the structure that is produced by the algorithm—one feature that distinguishes unsupervised learning from supervised learning and reinforcement learning. Another category of time-series data that will be analysed is the measurements of sensor values, both environmental and physiological, and that trigger notifications in the system on a certain occurrence. This analyse will generate forecasts of the critical occurrences that could happen in the future and will allow the physical trainers to better shape the therapeutical program of each patient or to avoid to assign activities that will fail to be completed. We will use supervised learning algorithms that will build

decision-trees describing the optimal sequence of activities, in time that could be recommended to a patient.

### *Cloud Platform for Data Fusion and Data Analysis from multiple sources*

The cloud platform for data fusion and data analysis based on big data and deep learning techniques intends to analyse the huge amount of heterogeneous information coming from the different municipal open data sources (traffic, air pollution, weather etc) and to extract its meaning in order to infer and produce easily understandable higher-level information about mobility and pollution to the applications for stakeholders, allowing them to send predictions, recommendations and alerts for a dynamic urban mobility management to the citizens and city authorities in real time.

This SW component is based on three fundamental pillars:

- A **data fusion approach** using semantic technologies to deal with the heterogeneity of the existing concepts in this domain knowledge and homogenise all of the incoming data and then, to store them in a structured form at the same level. So, the main semantic technology for this approach is Apache Jena, a Java framework which provides extensive Java libraries for helping developers to develop code that handles RDF, RDFS, RDFa, OWL and SPARQL in line with published W3C recommendations. Furthermore, Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies to store RDF triples in memory or on disk.
- A **big data analysis approach** to extract useful information from a huge data set and transform it into an understandable structure for further use. So, the main technology for this big data approach is Hadoop MapReduce, which is a computational model and software framework for writing applications which are run on Hadoop. These MapReduce programs are capable of processing massive data in parallel on large clusters of computation nodes.
- A **deep learning approach** to learn and be able to predict from unlabelled or unstructured data. Further details will be found in the final iteration of the deliverable *D5.4 – Report on techniques and algorithms for simulation*.

The inputs of the cloud platform for data fusion and data analysis are the historical data and the real-time data coming from the open data sources; while the outputs are the recommendations and predictions about mobility and pollution produced by itself.

### *Analysis and processing over a Traffic Simulation Platform*

This analysis and processing over a Traffic Simulation Platform intends to predict the evolution of traffic loads and pollution levels in a city from historical municipal open data allowing to the city authorities to make decisions on:

- the urban mobility planning based on the pollution levels,
- the contingency plans (protocols against the pollution), while simulating how such (new or already established) protocols could evolve along the time.

Or to perform different analysis about:

- the future predictions of the pollution levels,
- the management between the pollution levels and the traffic in the city.

This analysis and processing over a Traffic Simulation Platform consists of a SW sub-component for pre-processing data sources to be used by the Traffic Simulation Platform, and a SW sub-component for post-processing the results generated to predict the evolution of traffic loads and pollution levels in a city. Both SW sub-components are developed in Python and Bash to be integrated with the Traffic Simulation Platform.

Such Traffic Simulation Platform is based on SUMO (Simulation of Urban Mobility), that is an open source, highly portable, microscopic, multi-modal traffic simulation which allows to simulate how a given traffic demand which consists of single vehicles moves through a given road network.

The inputs of analysis and processing over a Traffic Simulation Platform are the historical data coming from the open data sources; while the outputs are the simulation results about traffic loads and pollution levels produced by itself.

### *Multi-modal Route Planner*

The multimodal route planning system is expected to support the optimization of citizen's individual transport. Being multimodal, it means that the system needs to contemplate different types of transport modes, or traverse modes, even in the same route.

The service will be available on different platforms, and it computes and suggests itineraries optimizing different criteria, such as the cost, the duration, the length, the environmental friendliness (CO<sub>2</sub> emissions) or the safety. Once the system provides a set of alternatives routes, the user will be able to choose the one that fits better with his/her requisites.

The multimodal route planning system developed is built upon OpenTripPlanner platform, which is an open source platform for multi-modal and multi-agency journey planning. It follows a client-server model, providing several map-based web interfaces as well as a REST API for use by third-party applications. OTP relies on open data standards including GTFS for transit and OpenStreetMap for street networks. OTP deployments now exist around the world and OTP is also the routing engine behind several popular smartphone applications. The base code used for OTP has been written in JAVA programming language, and it can be obtained from the GitHub of Open Trip Planner Project.

The inputs of the multi-modal route planner are the historical data and the real-time data coming from the open data sources; while the outputs are the calculated routes according to the user requests.

### *Data analysis for further knowledge about the influence of indoor air quality on personal conditions*

This data analysis SW component intends to extract new knowledge regarding influence of indoor air quality on personal conditions and/or workability through data pre-processing techniques such as checking data quality and extracting data features, and data analytics techniques such as correlation analysis and recognition of interesting events, e.g., unusual personal motion patterns and/ or states of personal discomfort.

Most of collected data are stored in PostgreSQL database and Azure table storage using NoSQL database. Due to EU GDPR (data protection requirements), user IDs, locations etc. are anonymized using random identifiers and hash tables; and the access to the data is only allowed for project researches and for scientific purposes.

### *Analytics for Tracking of Athletes Use Case*

In the Establish Cloud zone, there are some analytics on the data that is collected from environment, athletes and their trainings. To realize the analytics of data many machine learning techniques could be used according to data types. In Turkish use case the data is sensor data, so predictive analysis, classification and some clustering algorithms can be applied. The aims of the analytics can be listed as follow.

- Making predictions about athletes' performance.
- Determine wrong / correct training techniques
- Tracking athletes
- Determine impact of environmental factors on athletes' performance

### *Data analysis for further knowledge about decision support tools for behavioural choices and treatment options*

There are two types of data analysis performed:

- Real type analysis. Data received is analyzed, placed under Rule Engine or ML process, and important information retrieved.
- Long term analysis. Historical data is analyzed, placed under Rule Engine or ML process, and important information retrieved.

The presentation is in the form of charts and lists. For charts Grafana (Grafana, 2018) and Primefaces (Primefaces, 2018) charts are used.

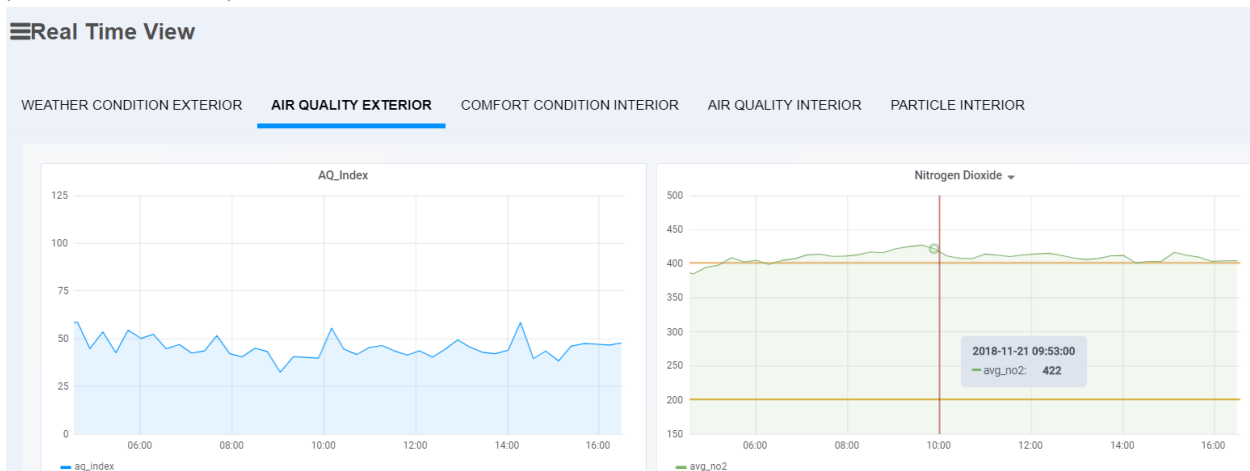


Figure 22. Real time analysis form

***For the use case “Promoting independence of specific vulnerable groups” also Machine learning algorithms are applied.***

For the purpose of this system, unsupervised learning methods are used. More specifically, “Multilabel classification” using k-means clustering algorithm applied on the time series of data collected from the input devices. (Pattanayak, 2017)

The value of k selected after an extended test, was 8. This mainly represents the following periods of day activity

1. Wake up
2. Morning
3. Between morning and noon
4. Noon

5. Afternoon
6. Evening
7. Late evening
8. Night

#### 4.2.2. Complex event processing

Complex event processing (CEP) is a method of analysing streams of data, received in real time, about things that happen (events). CEP systems combine data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events and respond to them as quickly as possible.

Complex Event Processor would take into account not only the data just received, but also the historical data (events that happened in the past and are somehow relevant). **Error! Reference source not found.** shows an overview of the structure of a CEP as it is currently understood.

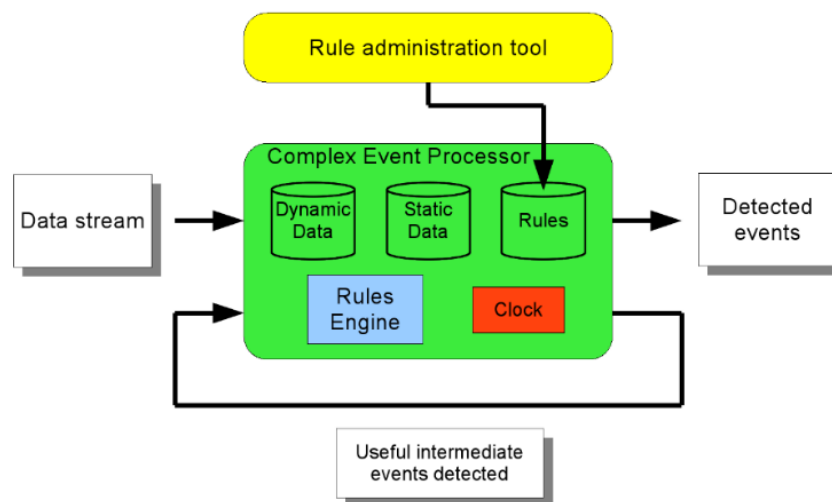


Figure 23. CEP overview

- *Dynamic Data* usually consists of the recent input data and possibly some useful intermediate derived data. It is stored in memory in order to apply the various rules on it. There is usually a window time frame to determine for how long the historical data must be kept.
- *Static Data* is some other data that describes the business context and is generally unmodified during the execution. Some examples could be data describing the infrastructure of an organization, unmodifiable parameters that depend on the country's laws, etc. *Rules* are expressions, usually with an if-then structure, written in some formal language. They are permanently being checked, in order to at some point determine the existence of a new event.
- The *Rule Engine* is the core component. Its task is determining when the rules match their condition and acting accordingly. Optimizing the amount of computations needed to achieve this task is usually not easy.
- A *Clock* is added to the diagram to stress the fact that some rules use time frames in their conditions, which means that the current time is also relevant for rule evaluation. *Useful Intermediate Events Detected* are auxiliary pieces of data that the CEP optionally stores temporarily (dynamic data) which are relevant in order to work with the rules. *Data Stream* is the input to the CEP. It often needs a pre-process in order to convert it to the CEP internal data



model. Usually a timestamp is associated to it. *Detected Events* is the output of the CEP. It is usually a series of time stamped data structures that describe some kind of event that is relevant in the business context. Other components will take it as input.

*Rule Administration Tool* is an optional external component that establishes the rules to be used by the CEP. It usually acts before the execution begins, though sometimes it is possible to update the rules on the fly, and therefore the behavior of the CEP evolves dynamically by applying new rules to the current historical dynamic data.

During the execution of the establish project, the following CEP technologies have been used:

- **Drools** is an open-source Business Rules Management System (BRMS) written in Java that provides, among other things, some powerful complex event processing features.
- **WSO2 CEP** is an open-source component that helps identify the most meaningful events and patterns from multiple data sources, analyze their impacts, and act on them in real time.

Complex Event Processing is about getting better information, in real time.



CEP is based on the observation that in many cases actions are triggered not by a single event, but by a complex composition of events, happening at different times, and within different contexts.

CEP is an approach that identifies data and application traffic as "events" of importance, correlates these events to reveal predefined patterns, and reacts to them by generating "actions" to systems, people and devices.

Examples can be listed about complex event processing like following;

- Regulatory constraints
- Fraud detection
- Aggregation
- CRM
- Intelligent Routing.

In the Tracking of athletes with wearable sensors pilot will be defined the rule sets about the collected data that is gathered from wearable sensors of athletes. Some actions will be triggered according to these rules.

#### 4.2.3. Event mediator

A Mediator is an object that coordinates interactions (logic and behaviour) between multiple objects. It makes decisions on when to call which objects, based on the actions (or in-action) of other objects and input. A mediator is best applied when two or more objects have an indirect working relationship, and business logic or workflow needs to dictate the interactions and coordination of these objects.

The mediator extracts the workflow from the implementation details and creates a more natural abstraction at a higher level, showing us at a much faster glance what that workflow is. We no longer have to dig in to the details of each view in the workflow, to see what the workflow actually is.

The event flow starts with the client sending an event to an *event queue*, which is used to transport the event to the mediator. The *event mediator* receives the initial event and orchestrates that event by sending additional asynchronous events to *event channels* to execute each step of the process. *Event processors*, which listen on the event channels, receive the event from the event mediator and execute specific business logic to process the event. It is important to note that the event mediator doesn't actually perform the business logic necessary to process the initial event, rather, it knows of the steps required to process the event. The event channels can be either message queues or message topics.

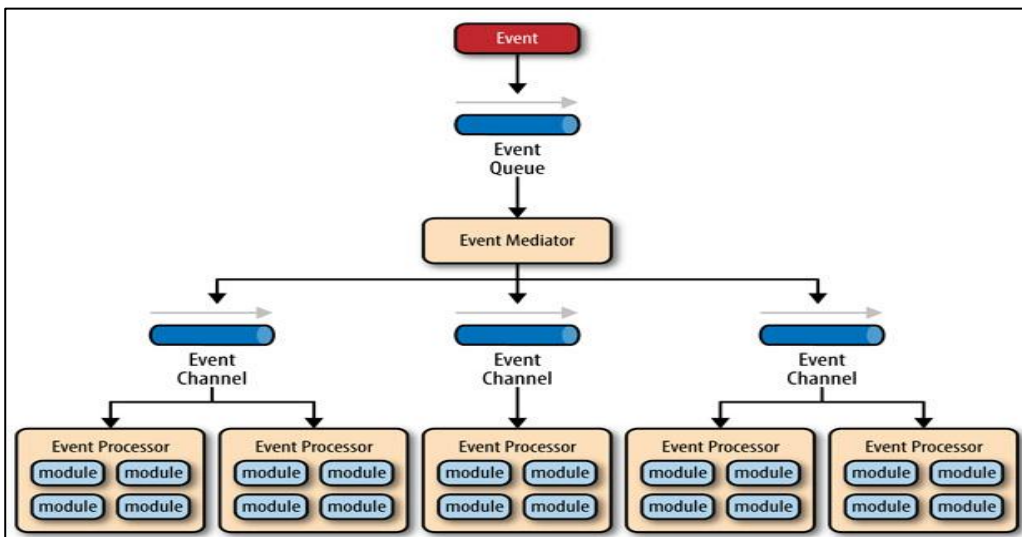


Figure 24. *Event processing*

For each initial event step, the event mediator creates a processing event, sends that processing event and *waits* for the processing event to be processed by the corresponding event processor. This process continues until all of the steps in the initial event have been processed.

### *Rule engine, for the use case “Promoting independence of specific vulnerable groups”*

Data received from all input sources, after pre-processing, is placed under the application of rule engines.

Rule engines are analyzing the input data, correlations between different parameters, and based on that, are creating notifications, alerts or recommendations.

**Edit Rules** ✕

RuleName:

Operator:

Vald1:

Vald2:

CreatedOn:

LastUpdatedOn:

ValidityStart:

ValidityEnd:

NumberOfRegistrations:

StartCronFormat:

EndCronFormat:

PatientId:

FitbitId:

EnvironmentDeviceId:

Severity:

Message:

Rule Type:

Operand1:

Operand2:

Figure 25. Rule definition form

Data si ora	CO2	NO2	PRES	TC	Masa volumica functie de temp.si presiune	CO2	NO2	PM10	PM2_5	AQI - valori medii orare		
	ppm	ppm	Pa	grd. C		mg/Nm3	ug/Nm3	ug/m3	ug/m3	NO2	PM10	PM2_5
5-11-2018 0:12	425.7612	0.283913	101742.9	9.81	25.21785515	742.87	517.89	74	45.35	517.89	74	45.3499985
5-11-2018 0:28	386.5583	0.27823	101745.9	9.5	25.18948467	675.22	508.09	48.04	41.14	508.09	48.0399971	41.1399994
5-11-2018 0:45	376.6127	0.261497	101740.1	9.33	25.1757688	658.21	477.79	50.06	39.86	477.79	50.0599976	39.8600006
5-11-2018 1:01	366.7154	0.266687	101751.3	9.24	25.16496112	641.19	487.49	44.05	41.18	487.49	44.0499992	41.1800003
5-11-2018 1:18	376.5644	0.277297	101746.4	9.04	25.14836248	658.84	507.22	48.28	38.58	499.70	52.89	41.22
5-11-2018 1:51	376.6609	0.274627	101738.1	8.8	25.12902512	659.52	502.72	71.66	42.14	496.66	52.42	40.58
5-11-2018 2:08	376.6127	0.273419	101725.5	8.58	25.11253564	659.87	500.84	55.95	42.46	495.21	54.00	40.84
5-11-2018 2:41	376.5644	0.270692	101710.2	8.36	25.09667819	660.20	496.15	52.07	46.24	498.88	54.40	42.12
5-11-2018 3:14	376.5644	0.273265	101687.4	8	25.07021289	660.90	501.40	57.79	49.93	501.67	57.15	43.87
5-11-2018 3:31	366.6672	0.255359	101669.5	7.98	25.07284845	643.46	468.49	76.75999	51.37	493.92	62.85	46.43
5-11-2018 4:21	366.6672	0.263202	101634.4	8.03	25.08596869	643.12	482.63	80.73	66.1	489.90	64.66	51.22
5-11-2018 4:54	356.77	0.27092	101641.2	8.07	25.08786152	625.72	496.75	2040.13	85.48	489.08	461.50	59.82
5-11-2018 5:11	366.6672	0.258362	101622.5	7.92	25.07908706	643.30	473.89	946.59	73.27	484.63	640.40	65.23
5-11-2018 5:44	366.7154	0.277087	101637	7.94	25.07728574	643.43	508.27	456.64	60.73	486.01	720.17	67.39
5-11-2018 6:01	366.6672	0.277736	101627.6	7.97	25.08229502	643.22	509.36	140.64	46.74	494.18	732.95	66.46
5-11-2018 6:17	366.6672	0.271088	101641.3	7.89	25.0717689	643.49	497.37	282.14	60.67	497.13	773.23	65.38
5-11-2018 6:50	366.6672	0.277737	101648	8.09	25.08795654	643.07	508.57	294.26	65.59	499.49	424.05	61.40
5-11-2018 7:07	376.5644	0.289064	101677.9	8.19	25.08951072	660.39	529.98	375.46	86.79	510.71	309.83	64.10
5-11-2018 7:23	376.5644	0.294852	101687.1	8.36	25.10237972	660.05	540.31	130.46	86.18	517.12	244.59	69.19
5-11-2018 7:40	376.6609	0.285911	101706.7	8.46	25.10646298	660.11	523.84	67.4	61.5	520.01	229.94	72.15
5-11-2018 7:56	386.4616	0.284025	101731.6	8.41	25.09585975	677.57	520.61	66.83	54.58	524.66	186.88	70.93
5-11-2018 8:13	386.5101	0.28773	101745.3	8.4	25.09159246	677.77	527.49	51.59	46.94	528.45	138.35	67.20

Figure 26. Rule application results

## 5 Acquisition / Interconnection Layer

### 5.1. IoT Backend Layer

#### 5.1.1. IoT Backend Data Management

Once the country pilots are done harvesting and pre-processing the data via their own processing methods, it will be transferred to the common data platform for further analysis and processing. This will be done using a gateway service of some sort, which will be defined on a per-pilot-basis. For example in the Finnish Indoor Air Quality Improvement at Schools pilot the gateway service used will most probably be Azure Data Factory, which is a cloud based data integration service that is available on the Azure Cloud Platform. Azure Data Factory makes the process of bringing the data from the pilot's own NoSQL Azure Table Storage to the common data platform not only significantly easier, but also automatic.

The data gathered can be divided into two types based on their structure. If the data is structured it will be brought into a tabular storage service such as Azure Data Warehouse. If on the contrary it is unstructured, it will be brought into a database that supports unstructured datatypes such as Azure Data Lake Store. If the non-pre-processed data is deemed valueless, it can then be discarded and only the pre-processed data kept.

The Romanian Use Case will use a proprietary connector in order to transfer the sensors data from the physical Gateway, Meshlium, to Cloud Platform. Having an easy-to-use interface, Meshlium allows the configuration of several cloud connectors.

After the data has been brought to a storage database of some sort, it can then be processed further. This will be done using the analysis services the Azure Cloud Platform offers such as HDInsight, Azure Machine Learning Studio and Azure Data Lake Analytics. These analytics services enable the obtaining of further insights from the data. Once the data has been processed it can then be utilized by the ESTABLISH Visualization Framework (EVIF) for visualizing and reporting. This is the most probable way of handling the backend data management, but these procedures have not been consolidated yet. The exact configuration of the data management will be verified when the deployment of the data platform approaches. Further detail on the procedures of data management on the common data platform can be found in deliverable 4.1 on the Data management platform architecture.

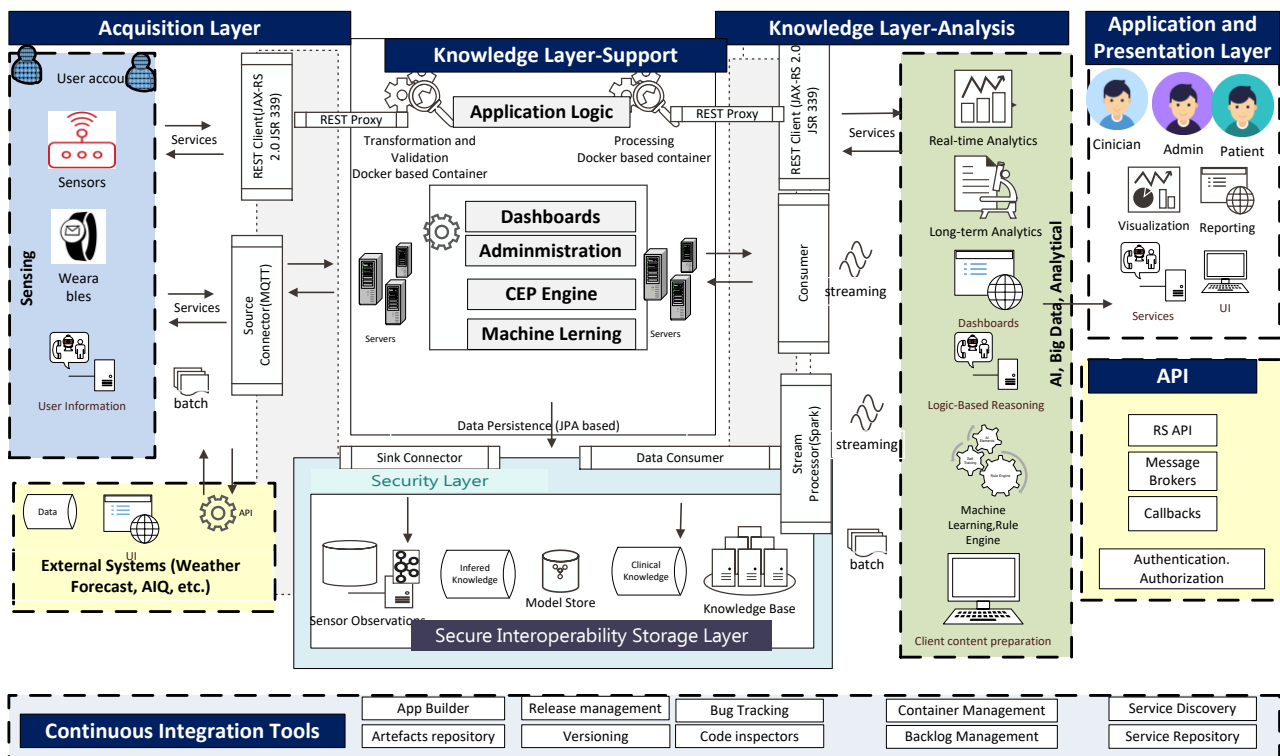


Figure 27. The ESTABLISH data flow mapped on logical architecture.

After receiving information from the sensors and processing it, the data is stored in special tables .

- A Data Table represents one table of in-memory relational data. The data is local to the .NET-based application in which it resides, but can be populated from a data source such as Microsoft SQL Server using a Data Adapter.
- A database trigger is special stored procedure that is run when specific actions occur within a database. Most triggers are defined to run when changes are made to a table's data. Triggers can be defined to run instead of or after DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.
- A stored procedure is a prepared SQL code that can save, so the code can be reused over and over again.

### 5.1.2. IoT Backend Device Lifecycle Management

Planning the lifecycle of the backend layer ahead is crucial. If the services and solutions are not shut down after the project, a proper lifecycle plan, which handles data lifecycle principles, should be included in the project plan.

The lifecycle management of the backend layer is discussed in detail in section 4.5 of the deliverable 4.1 on the Data management platform architecture

### 5.1.3. IoT Backend Protocol Adapters

As I was saying earlier, MVC a software design pattern where all the codes of a project are kept in Model, View and Controller while coding the project.

When sending information from the application to the solution backend a way of communication has to be established. There are three main ways for device-to-cloud communication arrangement from IoT Hub:

- Device-to-cloud messages for time series telemetry and alerts.
- Device twin's reported properties for reporting device state information such as available capabilities, conditions, or the state of long-running workflows. For example, configuration and software updates.
- File uploads for media files and large telemetry batches uploaded by intermittently connected devices or compressed to save bandwidth.

Each option has its unique elements of storage retrieval, maximum size capacity, communication frequency and compatible protocols. Below you can see a detailed comparison of each.

	<b>Device-to-cloud messages</b>	<b>Device twin's reported properties</b>	<b>File uploads</b>
Scenario	Telemetry time series and alerts. For example, 256-KB sensor data batches sent every 5 minutes.	Available capabilities and conditions. For example, the current device connectivity mode such as cellular or WiFi. Synchronizing long-running workflows, such as configuration and software updates.	Media files. Large (typically compressed) telemetry batches.
Storage and retrieval	Temporarily stored by IoT Hub, up to 7 days. Only sequential reading.	Stored by IoT Hub in the device twin. Retrievable using the <a href="#">IoT Hub query language</a> .	Stored in user-provided Azure Storage account.
Size	Up to 256-KB messages.	Maximum reported properties size is 8 KB.	Maximum file size supported by Azure Blob Storage.
Frequency	High. For more information, see <a href="#">IoT Hub limits</a> .	Medium. For more information, see <a href="#">IoT Hub limits</a> .	Low. For more information, see <a href="#">IoT Hub limits</a> .
Protocol	Available on all protocols.	Available using MQTT or AMQP.	Available when using any protocol, but requires HTTPS on the device.

Table: Comparison of device-to-cloud communication options

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-d2c-guidance>

## 5.2 IoT Gateway Layer

An Internet of Things (IoT) gateway<sup>23</sup> is a physical device or software program that serves as the connection point between the cloud and controllers, sensors and intelligent devices. All data moving to the cloud, or vice versa, goes through the gateway, which can be either a dedicated hardware appliance or software program. An IoT gateway may also be referred to as an intelligent gateway or a control tier. Some sensors generate tens of thousands of data points per second. A gateway provides a place to preprocess that data locally at the edge before sending it on to the cloud. When data is aggregated, summarized and tactically analyzed at the edge, it minimizes the volume of data that needs to be forwarded on to the cloud, which can have a big impact on response times and network transmission costs. Another benefit of an IoT gateway is that it can provide additional security for the IoT network and the data it transports. Because the gateway manages information moving in both directions, it can protect data moving to the cloud from leaks and IoT devices from being compromised by malicious outside attacks with features such as tamper detection, encryption, hardware random number generators and crypto engines.

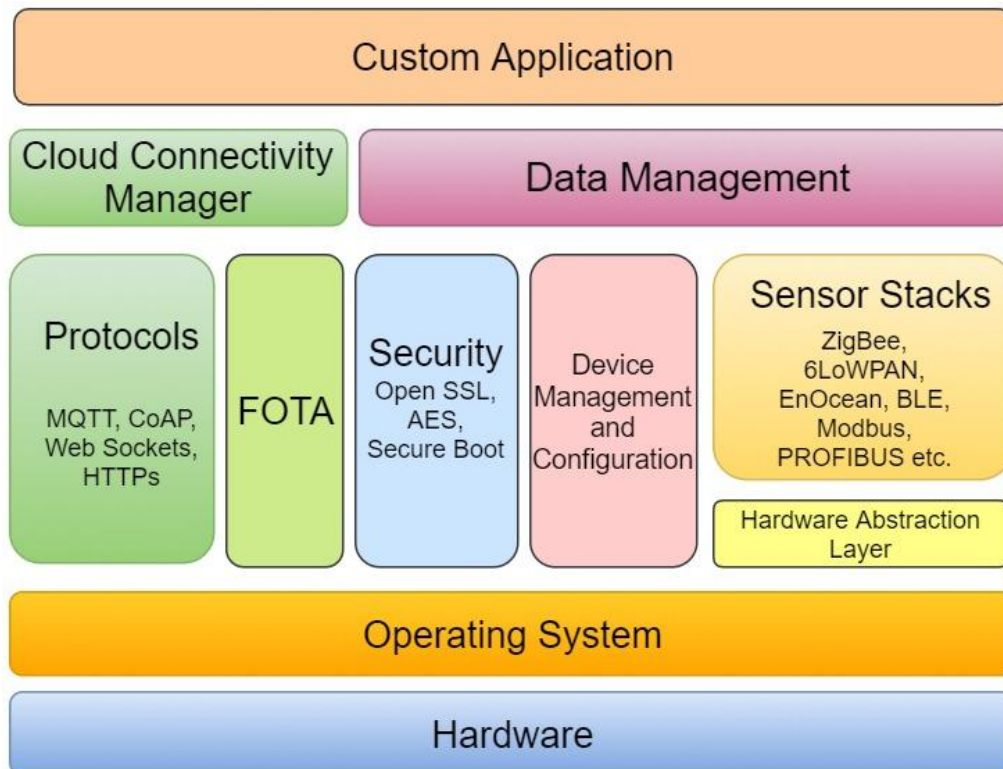


Figure 28. IoT Gateway Architecture

For the Romanian use case we refer to a physical device named Meshlium . One of the main applications of Meshlium<sup>24</sup> is being a gateway for Wireless Sensor Networks based on Waspnote (Waspnote Gas Board) and Plug & Sense devices (Smart Cities Pro). These are sensor nodes that can work with different communication technologies like WiFi, 4G or XBee among others.

Meshlium accepts POST and GET requests in any of its interfaces so Waspnotes are capable of sending frames, through GPRS, 3G, 4G or WiFi modules, via HTTPS requests. Meshlium, through HTTPS requests is capable of:



- Receive frames from 4G/3G/GPRS/GSM, WiFi or Ethernet via HTTPS.
- Parse these frames.
- Store the data in local Database.
- Synchronize the local Database with an external database.

### 5.2.1. IoT Gateway Data Management

Data Management includes data streaming, data filtering and data storing (in case of loss of connectivity with the cloud). IoT Gateway manages the data from sensor nodes to gateway and also the data from gateway to cloud. The challenge here is to minimize the delay to ensure data fidelity.

The gateway layer connects the edge layer (sensors and actuators) to the country pilot's own data platform. Together with the edge layer, it forms the physical part of the IoT framework. The gateway layer is a gateway device of some sort, for example a Wi-Fi/TCP device like in the Intelligent Air Quality Management System pilot or a processing unit like in the Finnish Indoor Air Quality Improvement at Schools-pilot.

The edge layer's sensors measure different environmental metrics, e.g. air humidity or physical activity of the subject, and creates data of these metrics. It then sends this data to the gateway device via some sort of protocol, most probably Wi-Fi or Bluetooth. The protocol used for the sensor to gateway traffic will be defined on a per-pilot-basis.

The sensors can also be arranged to communicate with each other like in the Intelligent Air Quality Management System pilot, in which the indoor air quality sensors communicate with the outdoor air quality sensors via radio frequencies. This way the data, which the outdoor sensors collect, can be merged with the data, which the indoor sensors collect, and thus a correlation between the two can be determined.

After the data has been transferred from the sensors to the gateway device via one of the aforementioned protocols, the gateway device has to connect to the pilot's data platform via some sort of gateway service, this too is pilot specific. In the Finnish Indoor Air Quality Improvement at School pilot there are two gateway services; one for sensor data and one for physiological data.

For data gathered by sensors tracking the air quality metrics, this gateway service will most likely be Azure's IoT Hub in conjunction with Azure Stream Analytics. Azure IoT Hub is a managed service that acts as a message hub between the gateway device and the pilot's database while Azure Stream Analytics is a processing engine that enables the examination of high volumes of real-time data. Using these services, constant real-time data can be gathered, analyzed and stored onto the database.

For data gathered by wearable and mobile devices, such as physiological data and user feedback, the gateway will most likely be an Android device, which connects to the database via Google's gRPC over the HTTP/2 protocol. The database itself will be hosted on a server based on Docker services, running on a virtual machine. Alternatively the data gathered by the wearable devices can be transferred to the server via an Android Wear application to gain access to raw measurement data.

The exact operations and processes of the gateway data management have not been consolidated yet and will be pilot specific. The responsibilities are communicated in the pilot by writing Data Management principles for parties involved in the project before starting the configuration phase. The principles are then discussed with all Pilot parties to ensure common understanding of the procedures.

The Data Management process specific to the Romanian use case consists in:

- The sensors nodes transmit the sensors measurements through 4G or WiFi connectivity to the Meshlium Gateway. Once in Meshlium, data is stored in a MySQL or Postgresql database that ensures local persistence of the sensors data. Regarding the Romanian platform, data forwarding to Cloud is done through a software component that serializes data to an MQTT broker. Similarly, uRADMonitor connects (via Wi-Fi) to a Cloud platform that allows data to be accessed via a REST interface.
- The integration with the common data platform can be achieved with the Azure IoT Hub service. This process can be performed by following the next steps:
  - Registering the Meshlium in Azure Portal, by annotating the connection string generated in Azure IoT Hub in the Meshlium configuration; basically, the obtained “connection string” from the Azure portal will certificate the Meshlium device as a valid sender of messages.
  - After validating the Meshlium Gateway, there are required a few configurations in the Configuration panel:
    - Number Requests: Number of requests to send per iteration.
    - Sync Interval: Time duration in seconds between synchronizing data batches.
    - Protocol: Choose the protocol to communicate with Azure IoT Hub. Valid protocols are MQTT (by default), AMQPS and HTTPS.
    - Log Level: Generate log messages. From fewer to more details, the levels are OFF, ERROR, INFO, DEBUG, REPORT. The default is OFF.

### 5.2.2. IoT Gateway Logic

Part of the analytics and logic can be shifted from the knowledge layer onto the gateway layer. In this type of computing, there’s a processing unit that the data passes through on the way from the sensors to the database. The processing unit can then for example filter useless or noisy information out from the data using various algorithms. This processing can reduce the size of transfers that are made between the sensors and the gateway.

In the Finnish Indoor Air Quality Improvement at School pilot, Azure Stream Analytics is a part of the gateway logic as the data is pre-processed before it is stored for the first time in the database. This way of handling data management is becoming more and more popular due to advancements in hardware technologies.

As for the Romanian Use Case, a Java based developed module will handle the Gateway Logic tasks; namely preprocess the sets of data in order to advance to the next phase - data visualization.

Data preprocessing is performed by using appropriate query functions, we presented below an example representing a simple data parsing task with storage capabilities.

```
SELECT
  *
INTO
  output
FROM
  input
```

cu

```
SELECT
  id,
  id_wasp,
  sensor,
  value,
  datetime
INTO
  output
FROM
  input
```

The following phases of data pre-processing are planned to be performed with the Java developed module from the backend, in order to achieve proper data to present further as an **AQI (Air Quality Index) visual instrument**. Thus, in order to adequately display the air quality measurements there are required to be performed the following data processing steps:

- data quality assurance management  
Consists of the validation conditions for **data correction** by comparison with specific intervals/criteria.
- data pre-processing  
Consists in the conversion of the measurements from the gas sensors.
- **comparison with the standard value**  
**For a comparison with the standard value the hourly average value needs to be estimated.**
- **use of AQI and value triggers** will be based on the latest report of UK COMEAP<sup>25</sup>.

### 5.2.3. IoT Gateway Protocol Adapters

Azure IoT protocol gateway adapters is a framework for protocol adaptation for high-scale two directional communications with the IoT Hub.

Azure IoT Hub natively supports communication over the MQTT, AMQP, and HTTPS protocols. In some cases, devices or field gateways might not be able to use one of these standard protocols and require protocol adaptation. In such cases, you can use a custom gateway. A custom gateway enables protocol adaptation for IoT Hub endpoints by bridging the traffic to and from IoT Hub. You can use the Azure IoT protocol gateway as a custom gateway to enable protocol adaptation for IoT Hub.

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-protocol-gateway>

Azure IoT protocol gateway enables devices to communicate with Microsoft Azure IoT Hub over MQTT and potentially other protocols. The protocol gateway provides a model for creating protocol adapters for different protocols. These adapters can be plugged into the execution environment through a 'pipeline' concept. The execution pipeline can also contain components (i.e. handlers) performing specialized processing of the data before it is passed to IoT Hub. The MQTT protocol adapter enables connectivity with IoT devices (or other clients) over the MQTT v3.1.1 protocol. The gateway bridges the communication to Azure IoT Hub using the standard IoT Hub client over the AMQP 1.0 protocol.

<https://github.com/Azure/azure-iot-protocol-gateway/blob/master/docs/DeveloperGuide.md>

IoT Hub allows devices to use the following protocols for device-side communications:

- MQTT
- MQTT over WebSockets
- AMQP
- AMQP over WebSockets
- HTTPS

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-protocols>

As mentioned, the Romanian use case WSN is based on Waspote (Waspote Gas Board) and Plug & Sense devices (Smart Cities Pro). There are 16 different wireless interfaces available (in the manufacturer portfolio) for Waspote including long range (4G / 3G / GPRS / GPRS+GPS / LoRaWAN / LoRa / Sigfox / 868 MHz / 900 MHz), medium range (ZigBee / 802.15.4 / DigiMesh / WiFi) and short range (RFID/NFC / Bluetooth 2.1 / Bluetooth Low Energy). The IoT devices deployed in the Rehabilitation decision support pilot are capable of sending frames, through 4G or WiFi modules via HTTPS requests.

### 5.3 Web Services

A web service is a technology that uses a set of protocols and standards that are used to exchange data between applications. Different software applications developed in different programming languages, and executed on any platform, can use web services to exchange data. Interoperability is achieved through the adoption of open standards. The OASIS and W3C organizations are the committees responsible for the architecture and regulation of Web services.

In the architecture of web services there are three parts: web service provider, the one that requests the web service and the publisher. The service provider sends a WSDL file with the definition of the web service to the service publisher. The one who asks for the service contacts the publisher and discovers who the provider is (WSDL protocol) and contacts the provider (SOAP protocol). The provider validates the service request and sends the structured data in XML format using the SOAP protocol. The XML file is validated again by the one requesting the service using an XSD file.

Most commonly standards: SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), REST (Representational State Transfer), XML (Extensible Markup Language), UDDI (Universal Description, Discovery and Integration), Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), or Simple Mail Transfer Protocol (SMTP).

A weather forecast web service from Open weather map have been used for the Optimized City and Mobility Planning Use case.

OpenWeatherMap<sup>26</sup> is an online service that provides weather data, including current weather data, forecasts, and historical data to the developers of web services and mobile applications. For data sources, it utilizes meteorological broadcast services, raw data from airport weather stations, raw data from radar stations, and raw data from other official weather stations. All data is processed by OpenWeatherMap in a way that it attempts to provide accurate online weather forecast data and weather maps, such as those for clouds or precipitation. Beyond that, the service is focused on the social aspect by involving weather station owners in connecting to the service and thereby increasing weather data accuracy.

The information retrieved from the web service is being stored in ElasticSearch in order to be used by other services as pollution prediction, CEP and Dashboards

## 5.4 OpenData

A huge amount of data is being generated by both public and private organizations. This data is stored beyond the reach of most people, secured in government or proprietary. The types and the depth of this data are growing as new and increasingly technological solutions and are implemented to solve the problems of the governments, businesses, and private citizens of smart cities. The majority of this data is going largely unseen and unused by limiting the number of people who can access to it. The solution to this is to make the data publicly available via an open government approach: An open data technology helps organizations manage and publish collections of data. It is used by national and local governments, research institutions, and other organizations that collect a lot of data.

The Optimized City and Mobility Planning Use case will mainly use data obtained from the open data of the VLCi Platform. It is based on Fiware, which is an open standard recommended by the European Commission for Smart Cities to ensure adaptation to the Internet of Things. The Main formats used by VLCi to expose the data are SHP, GML, WFS, WMS, KML, KMZ, CSV, JSON, JSON-LD, RDF XML/TURTLE /N3.

The open data is based on CKAN27. CKAN is a tool for making open data websites. CKAN is built with Python on the backend and Javascript on the frontend and uses The Pylons web framework and SQLAlchemy as its ORM. Its database engine is PostgreSQL and its search is powered by SOLR. It has a modular architecture that allows extensions to be developed to provide additional features such as harvesting or data upload.

Data sources selected are:

- Air pollution stations distributed throughout the city, to be able to know the pollution in the districts of the city
- Measuring Stations for Pollen
- Bike line
- Traffic cams
- Google Transit for public transport
- Parking
- Real-time traffic status
- Sense of circulation
- Intensity of bicycle
- Intensity of traffic
- Status of the Valenbici stations “public bike transportation”.

## 5.5 SOA Architecture

### “Decision support tools for behavioural choices and treatment options”

The whole system developed for the use case developed in Romania is based on the SOA architecture.

A **service-oriented architecture (SOA)** is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies. A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently, such as retrieving a credit card statement online.

A service has four properties according to one of many definitions of SOA:

1. It logically represents a business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers.
4. It may consist of other underlying services.

Different services can be used in conjunction to provide the functionality of a large software application, a principle it shares with modular programming. Service-oriented architecture integrates distributed, separately-maintained and deployed software components. It is enabled by technologies and standards that make it easier for components to communicate and cooperate over a network, especially an IP network.

## 5.6 SOA Architecture layers

The following main layers are considered in the SOA layered architecture

- Solution Security Layer
- Software infrastructure, IT Management, Monitor, Audit Layer
- Access Management and Load Balancing Layer
- Kernel Layer

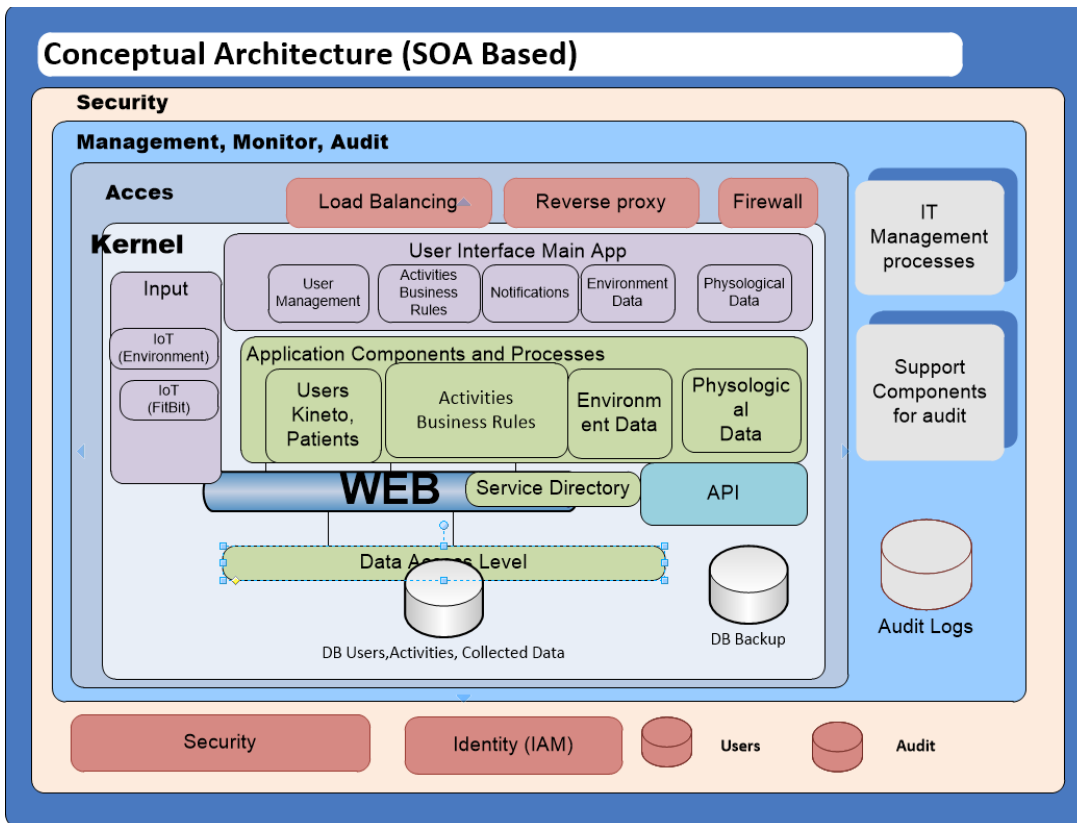


Figure 29. Soa based architecture



## 6 Constraints

### Optimized City and Mobility Planning

Operational constraints consist of the following areas:

#### *Open data Limitation*

Most of the information handled by this pilot is based on the data obtained from the open data of the city of Valencia. The quality of the results of the use case depends very much on the accuracy of the data obtained and the frequency of updating them.

After a first review of the data, we realized that they are not as accurate as we expected in addition to having a higher frequency of the desired update. For example, the information of the air pollution stations is updated daily, with such a low refresh rate, the usefulness of this information is compromised. There are other sets of data that are updated in real time, providing accurate information to obtain satisfactory results.

Valencia Open Data Platform is in continuous evolution and plans to incorporate new data sets that a priori seem very interesting for the Spanish pilot. From the Spanish consortium we are in continuous contact with the technicians of the municipality to keep track of the new data sets that will be incorporated in the short term to evaluate a future incorporation to the Spanish pilot

#### *Technical limitations*

As a result of the Spanish use case, two software components will be developed: a web site and an Android application.

#### *Web site limitations*

The website will provide the following functionalities: dashboards, routing and traffic simulation planning. The website will have two perspectives, one for the citizens and other for the city authorities.

Most of the functionalities will be responsive and available through different devices (PCs, Tablets and mobile phone). The dashboard won't be responsive because it is based on Kibana open source and in the current version, it does not provide this feature.

The web site will be optimized for the following resolutions:

- Smart Mobile Phone – designated mobile applications will be built
- Small devices (Tablets): resolution constraint is considered a minimum width of 768 px.
- Medium devices (Desktops): resolution constraint is considered a width equal or larger than 992 px.

The web site will be optimized to work under the following operating systems and clients (browsers):

- Windows 7 (and higher):
  - Internet Explorer 11 and Internet Explorer Edge and higher
  - Google Chrome 54 and higher
  - Mozilla Firefox 49 and higher

- MAC OS X 10.11 "El Capitan" (and higher):
  - Safari 8.0 and higher
- iOS 9 (and higher): default Safari installation
- Android Lollipop v 5.0 (and higher): default Google Chrome installation

### *Mobile application limitations*

The mobile application to be developed will display the route planner and the notifications of the pollution and traffic alerts. The application will be available only for the android devices. The application will be available through the "google play" app market store. To run the application, you will need an android version > 5.0 (Lollipop version) and connection to Internet.

## Developing smart HVAC systems that ensure a healthy indoor environment

### Smart HVAC systems that ensure a healthy indoor environment

All constraints in following text are considered at medium level usage.

### *Browser limitations*

- Operating system: Windows 7 and higher
- Internet Explorer 11 (Version: 11.726.15063.0)
- Google Chrome (Version: 54.0.2840)
- Mozilla Firefox (Version: 50.0.2)

### *Hosting environment limitations*

- Web Server
  - Processor: 2 CORE, 2.2 GHz per core
  - RAM: 4 GB
  - Hard disk capacity: 500 GB (not including backup)
  - Bandwidth: at least 10 MBps
  - Operating system: Windows 2012 Server
  
- Database Server
  - Processor: 2 CORE, 2.2 GHz per core
  - RAM: 4 GB
  - Hard disk capacity: 100 GB (not including backup)
  - Bandwidth: at least 10 MBps
  - Operating system: Windows

INSTANCE	CORES	RAM	STORAGE
B1	1	1.75 GB	10 GB

Figure 30. Azure App Service: Basic Service Plan

#### Basic

	DTUs	INCLUDED STORAGE	MAX STORAGE
B	5	2 GB	2 GB

Figure 31. Azure SQL Database: Single database model

- Mobile application operating systems
  - iOS 9 and higher (Safari browser)
  - Android 6.0 and higher (Google Chrome browser)

#### Users and sensors limitations

Not identified yet.

#### Intelligent air quality management system

Operational constraints were defined for two areas:

#### Browser limitations

The Establish Korean Pilot platform is based on Apache Tomcat framework designed to work on three levels: the database level, the application level, and the interface level. Any web browser that supports HTML5 and Javascript can be used as a client in the Establish Korean Pilot Platform.

The Establish Korean Pilot Platform will be optimized for the following resolutions:

- Smart Mobile Phone: resolution constraint is considered a minimum width of 1024 px.
- Small devices (Tablets): resolution constraint is considered a minimum width of 1024 px.
- Medium devices (Desktops): resolution constraint is considered a width equal or larger than 1024 px.

The Establish Korean Pilot Platform will run on any clients (browsers) supporting HTML5 and Javascript under the following operating systems:

- Windows 7 (and higher):
  - Internet Explorer Edge and higher
  - Google Chrome 54 and higher
  - Mozilla Firefox 49 and higher
- MAC OS X 10.11 "El Capitan" (and higher):
  - Safari 8.0 and higher
- iOS 9 (and higher)
  - Safari installation enabling HTML5 and Javascript
- Android Jelly Bean v 5.0 (and higher):
  - Google Chrome installation enabling HTML5 and Javascript

For each operating system and browser version, the resolution constraints presented above are valid.

### *Hosting environment limitations*

The components of the Establish Korean Pilot Platform can ensure full functionality and performance that is defined as having a maximum of 100 concurrent users on the platform. The concurrent users are defined as the number of users that can use a certain section of the platform at the same time. For this level of usage the following hosting cluster will be required:

- Web Server

Processor: 14 CORE, 2.3 GHz per core

Memory: 128 GB

Hard disk capacity: 512 GB

BANDWIDTH: at least 100 MBps

Operating system: Windows 10

- Database server

Processor: 12 CORE, 2.6 GHz per core

Memory: 64 GB

Hard disk capacity: 1TB

BANDWIDTH: at least 100 MBps

Operating system: Windows 10

## Promoting independence of specific vulnerable groups

### Rehabilitation decision support

Operational constraints of the platform consist of four areas:

- Security constraints
- Browser limitations
- Hosting environment limitations
- Sensors limitations

### *Security constraints*

Access to environmental data is permitted only using secured connections between sensors, MQTT broker, ESTABLISH backend, and ESTABLISH frontend

Access to data from wearable is permitted only using OAuth 2.0 protocol, and authentication in the owner account.

The results of applying Ai algorithms will not be used in automatic decisions. There will be notifications and advises, and a human operator is responsible for a decision.

### *Browser limitations*

This pilot resulted platform is based on JEE MVC (Model View Controller) pattern, designed to work on three levels: the database level, the application level, and the interface level, available as an Intranet/Internet solution on different devices (e.g., PC and tablets).

The rehabilitation decision support solution will be optimized for the following resolutions:

- Smart Mobile Phone. The application is accessed via a web browser
- Small devices (Tablets): resolution constraint is considered a minimum width of 768 px.
- Medium devices (Desktops): resolution constraint is considered a width equal or larger than 992 px.

The rehabilitation decision support solution will be optimized to work under the following operating systems and clients (browsers):

- Linux Ubuntu 18 (and higher)
  - Google Chrome 54 and higher
  - Mozilla Firefox 49 and higher
- Windows 8 (and higher):
  - Internet Explorer Edge and higher
  - Google Chrome 54 and higher
  - Mozilla Firefox 49 and higher
- MAC OS X 10.11 "El Capitan" (and higher):
  - Safari 8.0 and higher
- iOS 9 (and higher): default Safari installation
- Android Lollipop v 5.0 (and higher): default Google Chrome installation

For each operating system and browser version, the resolution constrains presented above are valid.

### *Hosting environment limitations*

The components of the rehabilitation decision support solution can ensure full functionality and performance on the following SLA (service-level agreement) commitments:

Response time for initial dashboard loading <3 seconds

Response time for page change, without loading data from server < 2 seconds

Response time for massive report loading (less than 10 000 records) < 30 seconds

Response time for Chart display < 5 seconds for each chart in a page.

- Low Level Usage:

Low level usage is defined as having a maximum of 40 concurrent users on the platform. The concurrent users are defined as the number of users that can use a certain section of the platform at the same time. For this level of usage the following hosting capabilities are required:

Processor: 6 CORE, 2.4 GHz per core

Memory: 8 GB

Hard disk capacity: 100 GB (not including backup)

BANDWIDTH: at least 10 MBps

Operating system: UNIX based (CentOS or Ubuntu) are recommended, Windows systems can also be used.

- Medium Level Usage:

Medium level usage is defined as having a maximum of 100 concurrent users on the platform. The concurrent users are defined as the number of users that can use a certain section of the platform at the same time. For this level of usage the following hosting cluster will be required:

- a. Web Server

Processor: 12 CORE, 2.4 GHz per core

Memory: 32 GB

Hard disk capacity: 500 GB (not including backup)

BANDWIDTH: at least 10 MBps

Operating system: Unix based (CentOS or Ubuntu) are recommended, Windows systems can also be used.

- b. Database server

Processor: 6 CORE, 2.4 GHz per core

Memory: 8 GB

Hard disk capacity: 100 GB (not including backup)

BANDWIDTH: at least 10 MBps

Operating system: Unix based (CentOS or Ubuntu) are recommended, Windows systems can also be used.

- High Level Usage:

High level usage is defined as having a maximum of 300 concurrent users on the platform. For this level of usage the following hosting capabilities must be scaled to a server cluster that will include:

- a. Load balancer:

Processor: 6 CORE, 2.4 GHz per core

Memory: 8 GB

Hard disk capacity: 100 GB (not including backup)

BANDWIDTH: at least 10 MBps

Operating system: UNIX based (CentOS or Ubuntu) are recommended, Windows systems can also be used.

- b. Three Web servers:

Processor: 8 CORE, 2.4 GHz per core

Memory: 16 GB

Hard disk capacity: 200 GB (not including backup)

BANDWIDTH: at least 10 MBps

Operating system: UNIX based (CentOS or Ubuntu) are recommended, Windows systems can also be used.

c. Two database servers:

Processor: 6 CORE, 2.4 GHz per core

Memory: 8 GB

Hard disk capacity: 100 GB (not including backup)

BANDWIDTH: at least 10 MBps

Operating system: UNIX based (CentOS or Ubuntu) are recommended, Windows systems can also be used.

### *Sensors limitations*

For environmental data, the following parameters will be measured, indoor and/or outdoor

Temperature

Humidity

Pressure

CO2

CO

NO2

O3

O2

Particles 1, 2.5, 10

The accuracy is mentioned in the description of components.

From wearable, the following parameters will be measured

Battery level

Heart rate

Steps

Sleep

The accuracy is mentioned in the description of components.

### *Indoor air quality improvement at school*

Operational constrains of the Finnish Pilot:

- Sensors and data collection: Microsoft Azure IoT Hub is capable of handling 400 000 messages/day. It can handle more than 500 000 devices, so there are practically no constraints in the context of planned pilot (See IoT Hub documentation).
- Physiological data collection: 10 concurrently active users per one instance of data collection virtual machine. Elastic scaling over multiple VMs via Kubernetes. No limit on the total number of users.

There are two separate mobile applications in the Finnish pilot, one for self-reporting air quality issues (e.g. symptoms, stress, etc.), and another for collecting physiological data (heart rate, etc.). Both mobile apps require Android. Minimum Android version for self-reporting app is 5.0 (API level 21), while for the physiological data collector it is 6.0.



## 7 Conclusions

This deliverable specifies the high level building blocks for the technical solutions that will be developed in the ESTABLISH project. The detailed design of each component will be performed in the corresponding tasks in subsequent work packages.

High level building blocks are situated in 4 layers: presentation layer, data acquisition layer, business logic layer and persistence layer. Various types of data are collected: physiological data including health and activity data, indoor and outdoor air quality data and data related to transportation. Microsoft Azure will be used for data storage and processing and for integrating data from multiple sources such as sensor networks and wearable devices and other systems. In addition to the services provided by Microsoft Azure for data analytics, Web applications and mobile applications will be developed for end users.

The deliverable gives an overview of the architectures used for environmental monitoring e.g. in healthcare domain. Furthermore, frameworks for developing specific parts of the architecture are analysed.

Development aspects are presented in addition to the logical components, including development level, staging level and production level, and the chosen approaches for frontend and backend development are discussed.

After describing the common non-functional requirements and pilot-specific constraints, are also presented the individual pilot-specific architectures, the components of the logical system architecture and also the collected sensor data are specified. As the pilots are still under specification, the final versions of the architectures will be presented in the second release of this deliverable.

## 8 References

- 
- <sup>1</sup> <http://www.tutorialsteacher.com/mvc/mvc-architecture>
  - <sup>2</sup> <https://db-engines.com/en/ranking>
  - <sup>3</sup> <http://www.tutorialsteacher.com/mvc/mvc-architecture>
  - <sup>4</sup> D3 visualization library : <https://d3js.org/>
  - <sup>5</sup> React : <https://reactjs.org/>
  - <sup>6</sup> <https://vuejs.org/> and <https://en.wikipedia.org/wiki/Vue.js>
  - <sup>7</sup> JSX : <https://jsx.github.io/>
  - <sup>8</sup> OAuth: <https://en.wikipedia.org/wiki/OAuth>
  - <sup>9</sup> <http://docs.grafana.org/>
  - <sup>10</sup> <https://www.upwork.com>
  - <sup>11</sup> <https://www.apple.com/swift/>
  - <sup>12</sup> <https://developer.android.com/studio/index.html>
  - <sup>13</sup> [https://msdn.microsoft.com/en-us/library/windows/apps/ff402526\(v=vs.105\).aspx](https://msdn.microsoft.com/en-us/library/windows/apps/ff402526(v=vs.105).aspx)
  - <sup>14</sup> [https://en.wikipedia.org/wiki/React\\_\(JavaScript\\_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library))
  - <sup>15</sup> <http://www.reactnative.com/>
  - <sup>16</sup> <https://www.nativescript.org/>
  - <sup>17</sup> <https://www.typescriptlang.org/>
  - <sup>18</sup> <https://www.xamarin.com/>
  - <sup>19</sup> <https://cordova.apache.org/>
  - <sup>20</sup> <https://phonegap.com/>
  - <sup>21</sup> <https://ionicframework.com/>
  - <sup>22</sup> <http://www.mysql.com/>
  - <sup>23</sup> <https://whatis.techtarget.com/>
  - <sup>24</sup> [http://www.libelium.com/downloads/documentation/meshlium\\_technical\\_guide.pdf](http://www.libelium.com/downloads/documentation/meshlium_technical_guide.pdf)
  - <sup>25</sup> <https://www.gov.uk/government/groups/committee-on-the-medical-effects-of-air-pollutants-comeap>
  - <sup>26</sup> <https://openweathermap.org/>
  - <sup>27</sup> <https://ckan.org>
  - <sup>29</sup> Grafana. (2018). Retrieved from <https://grafana.com/grafana>
  - <sup>30</sup> MQTT main page. (2018). Retrieved from <http://mqtt.org/>
  - <sup>31</sup> OAuth 2.0 official site. (2018). Retrieved from <https://oauth.net/2/>
  - <sup>32</sup> Pattanayak, S. (2017). *Pro Deep Learning with TensorFlow-A Mathematical Approach to Advanced Artificial Intelligence in Python*. Apress.
  - <sup>33</sup> Prime Faces. (2018). Retrieved from <https://www.primefaces.org>