"Software Engineering"
Course
a.a. 2018-2019
Template version 1.0
Deliverable #2

Lecturer: Prof. Henry Muccini (henry.muccini@univaq.it)

Dashboard Monitoraggio Ambientale

Date	23/12/2018
Deliverable	2
Team (Name)	5 Curly Brackets

Team Members		
Name & Surname	Matriculation Number	E-mail address
Marco Ovidi	247986	marco.ovidi@student.univaq.it
Matteo Di Paolo	243581	Matteo.dipaolo@student.univaq.it
Carlo Antonio Venditti	248254	Carloantonio.venditti@student.univaq.it
Dima Mhrez	247052	Dima.mhrez@student.univaq.it
Andrea D'Angelo	248817	Andrea.dangelo6@student.univaq.it

Project Guidelines
[do not remove this page]

This page provides the Guidelines to be followed when preparing the report for the Software Engineering course. You have to submit the following information:

- *This Report*
- *Diagrams (Use Case, Component Diagrams, Sequence Diagrams, Entity Relationships Diagrams)*
- *Effort Recording (Excel file)*

Important:
- ***document risky/difficult/complex/highly discussed** requirements*
- *document decisions taken by the team*
- ***iterate**: do not spend more than 1-2 full days for each iteration*
- ***prioritize** requirements, scenarios, users, etc. etc.*

Project Rules and Evaluation Criteria

General information:
- *This homework will cover the 80% of your final grade (20% will come from the oral examination).*
- *The complete and final version of this document shall be **not longer than 40 pages** (excluding this page and the Appendix).*
- *Groups composed of five students (preferably).*

I expect the groups to submit their work through GitHub

Use the same file to document the various deliverable.
Document in this file how Deliverable "i+1" improves over Deliverable "i".

Project evaluation:
Evaluation is not based on "quantity" but on "quality" where quality means:
- *Completeness of delivered Diagrams*
- *(Semantic and syntactic) Correctness of the delivered Diagrams*
- *Quality of the design decisions taken*
- *Quality of the produced code*

Table of Contents of this deliverable

Sommario

List of Challenging/Risky Requirements or Tasks

<In this section, you should describe using the table below the most challenging or discussed or risky design tasks, requirements, or activities related to this project. Please describe when the risk arised, when and how it has been solved.>

PLEASE FILL IN THIS TABLE AT EACH DELIVERABLE

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Identifying the right technologies	19/11	30/11	Testing performances on own computers or test servers
Learning how to implement NetData efficiently	19/11	11/12	Decided NetData would be too hard to implement efficiently with our time constraints.
Using new DB Technologies we're not familiar with	19/11	21/12	We studied a lot both the documentation of such technologies and how they could help our project.
Reinventing the system respecting our time constraints	11/12	23/12	We spent way more time on this deliverable than we did on the last.

A. Requirements Collection

A.0 Detailed Scenarios

Scenario 1

The sensor (we'll call it Sensor A) in the Hospital building keeps blindly sending short messages to the server, composed of its unique ID and the value it's sensing, to a certain port that is instructed to only accept signals of that kind [ID+Value]. The server keeps receiving these messages and checking if any of the values are above the threshold. In this scenario, it finds that none is: the situation is stable near Sensor A and thus no Warnings are generated.
The server then stores the values in the Time Series Database, and the Values Cache. The clients will get those values from there and process them to show them as simply as possible to the actual human Managers. [Continues with Scenario 2].

Scenario 2

Paul, the building manager, logs in the system using the credentials the admin Nora gave him. He was just hired, so he had no chance to change his password yet: he has to use the username and password that were generated by the system, based on his data. As soon as he logs in, the dashboard appears on his screen and each sensor he has control over is assigned to a certain value.
Paul only has to work three hours today (lucky him!), so he keeps the dashboard running and experiences no warnings or alerts during the entire period. Values keep getting updated real time but none of them is above the threshold, so all of them are green. Eventually, Paul logs off the system and goes home.

Scenario 3

Another sensor (Sensor B) is near the hospital right where a wildfire is located. It sends anomalous data to the server, that finds out something is wrong when checking the thresholds for temperature. The server then sends a request to Sensor B to up its update frequency and stores the Warning in a cache, while still sending data to the Time Series DB and the values cache. Sensor B starts sending data more frequently, and the server keeps the warning up for the client for as long as the values are above the threshold.

[Continues with Scenario 4]

Scenario 4

Erika begins her shift right after Paul. She's not just a building manager, but a Urban Manager, so she has a bigger area to inspect; when she launches the system, she notices a big warning saying that many values are above the threshold near the building and a wildfire might be starting.
Considering the values and their updates, Erika decides to call the firefighters to extinguish the fire.
When done, Erika checks that the value returned to their normal values and continues her shift.

Scenario 5

The company has just hired a new manager, Paul. Nora, the admin, must set up a new account for him, so she logs in the system as the administrator, and select the button "New User". Eventually, another password input is required to make sure she is the admin and she is currently active on the screen. Nora inserts the new manager's data into the system, and they are stored in the Relational DB. Nora receives a computer-generated username and password to give Paul, and he is now able to log in the system with these new credentials. Nora is done for the day so she logs off the system.

A.1 Functional Requirements

- Unusual values must be shown with appropriate colors based on their priority.

- Values over the defined threshold must be shown explicitly.

- There are 3 types of managers, related to the zones they have to monitor (building, area, city).

- Dashboard must show all sensors.

- Admin must be able to change the defined threshold values.

- The System must be able to detect sensors' failures and display a warning.

  If there are backup sensors for a measured property, the warning will have a lower priority.

- Different managers are given information with different levels of detail, based on a hierarchical relationship.

- The user must be able to select a zone to restrict the sensors displayed to only those in that specific zone.

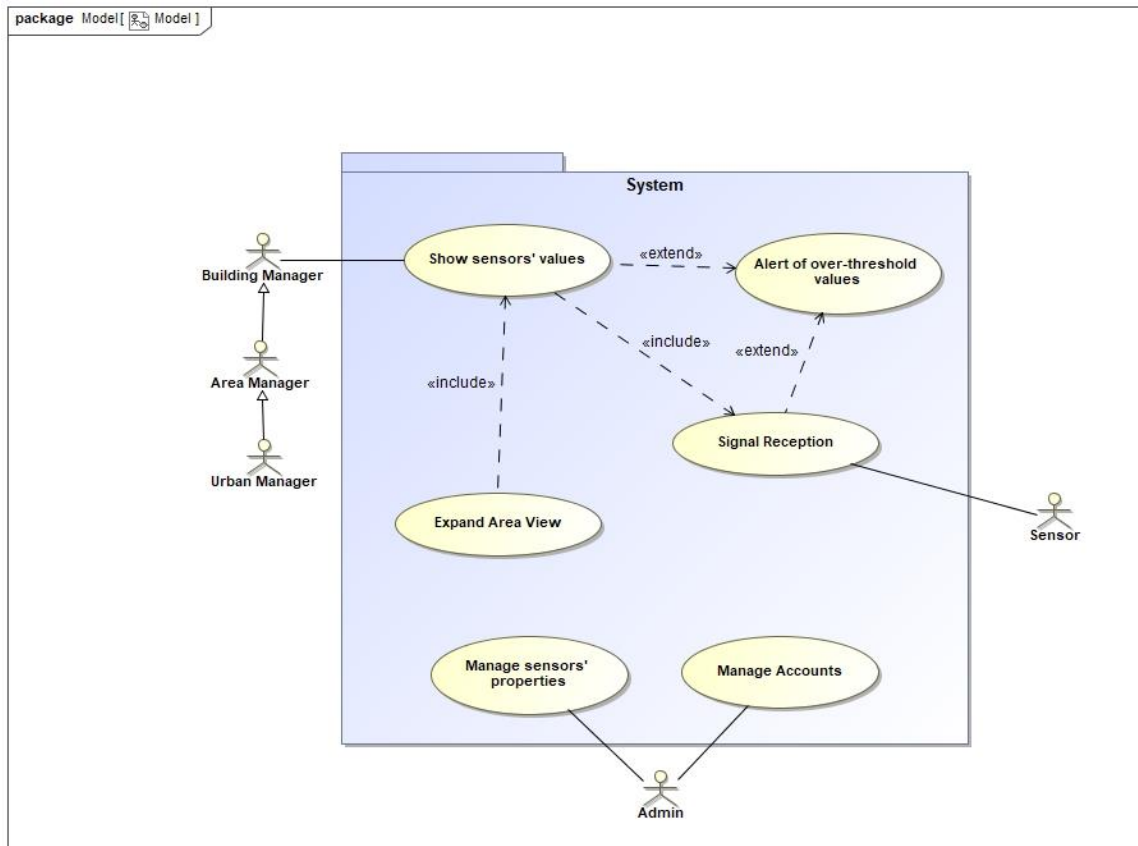- If a sensor detects unusual values, its frequency must be updated.

Figure 1: Use Case Diagram

Figure 1 represents the use case diagram of our model with a very high level of abstraction. We identified two types of users: the managers (Urban, Area, and Building Managers where there are generalization relations among them) and the admins. Managers access the dashboard for environmental monitoring while admins can change threshold values of sensors and add new managers.
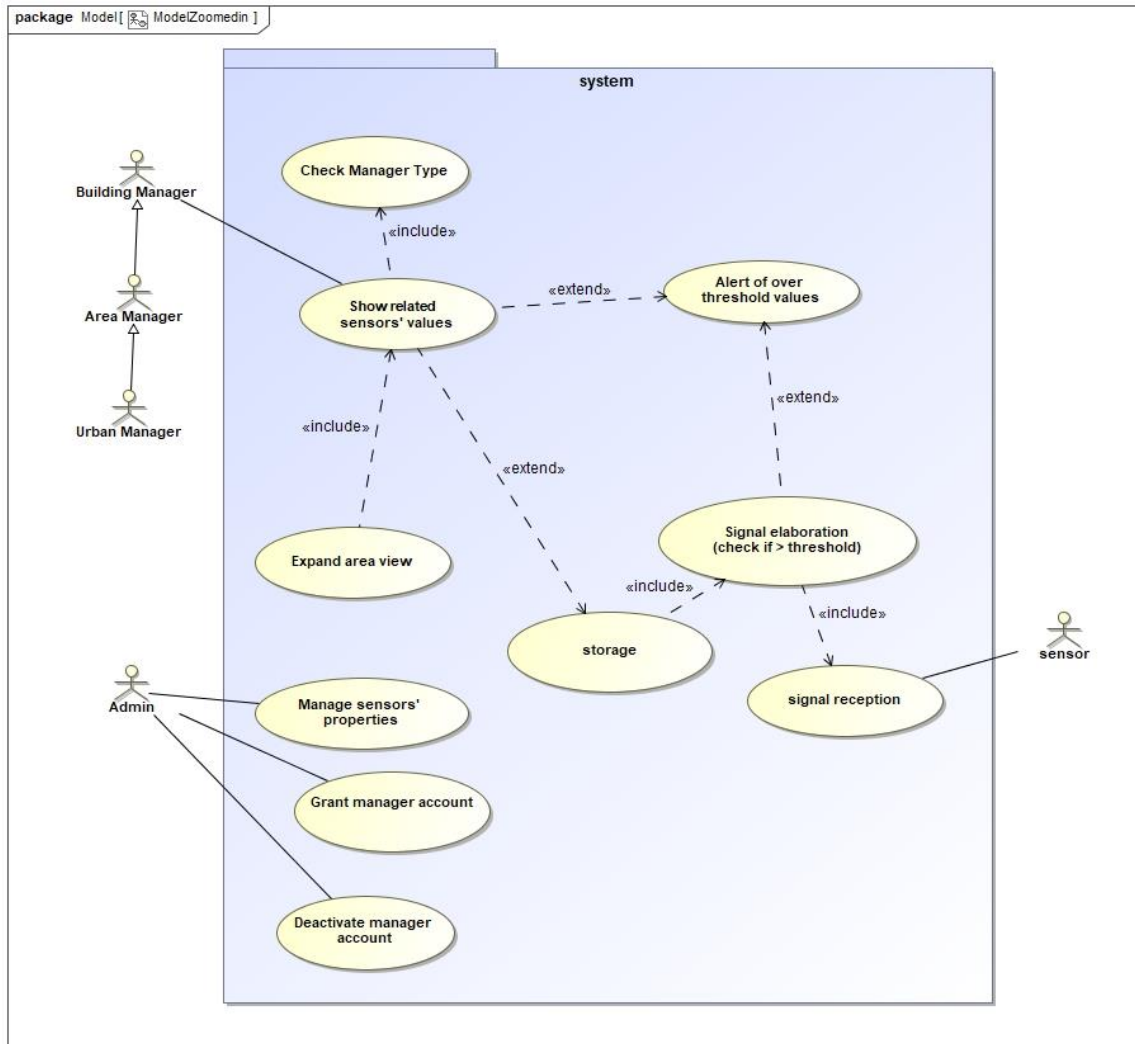
Figure 2: More detailed Use Case Diagram

Figure 2, is the second use case diagram. It shows in greater details the most complex use cases we found. Specifically, *"Signal Reception"* was split in *Signal Reception*, *Storage*, and *Signal Elaboration*, while a new Use Case called *Check Manager Type* was added as an include for *Show sensors' values*.

The tabular descriptions below are in reference to the Figure 2 diagram. They're saved as images for format purposes. The editable version of these tables can be found in the Diagrams directory.

USE CASE 1	Signal Reception	
Goal in Context	Signals from the sensors are received correctly.	
Preconditions	Sensors are installed and the system is running.	
Success End Condition	Values are received and processed.	
Primary actor	Sensor	
Trigger	No trigger	
DESCRIPTION	Step	Action
	1	Sensors send the detected values to the system.
	2	The system correctly receives and process those values.

RELATED INFORMATION	Signal Reception
Priority:	Very High
Performance	The system must be able to handle at least 150000 signals per minute.
Frequency	Every minute.
Superordinates	Signal Elaboration

USE CASE 2	Signal Elaboration	
Goal in Context	Understand if the values are usual or above the threshold.	
Preconditions	Signal Reception is working correctly.	
Success End Condition	The correct values are sent to the client.	
Primary actor	Sensor.	
Trigger	The system receives the data.	
DESCRIPTION	Step	Action
	1	The system receives the data.
	2	The sensor that sent the values is identified and its set threshold values are found.
	3	The newly received values are compared to the thresholds.
EXTENSIONS	Step	
	1a	Some values are over the threshold: Alert of values over the threshold.

RELATED INFORMATION	Signal Elaboration
Priority:	Very high
Performance	Must be able to elaborate at least 150000 signals per minute.
Frequency	Every time a signal is received.
Superordinates	Storage

USE CASE 3	Alert of over-threshold values	
Goal in Context	Make sure the Manager is alerted of the critical conditions	
Preconditions	Manager is logged in dashboard	
Success End Condition	Alert is given to manager in case of dangerous situations in a certain area	
Failed End Condition	System alerts Manager that it isn't able to detect dangerous situations	
Primary	Building Manager, Area Manager, Urban Manager	
Trigger	A value over threshold is detected	
DESCRIPTION	Step	Action
	1	A value over threshold is detected
	2	A warning is displayed
SUB-VARIATIONS		Branching Action
	1	One value type over threshold is shown ORANGE
	2	Two value types over threshold are shown RED
Priority:	Very High	
Frequency:	Every time that an unusual value is detected	

USE CASE 4	Storage	
Goal in Context	Signal values are correctly stored.	
Preconditions	Values have been received and handled correctly.	
Success End Condition	The system will know the last known values from that sensor.	
Primary actor	Sensor	
Trigger	A signal is received.	
DESCRIPTION	Step	Action
	1	The signal is received and handled.
	2	The signal is stored to complete the story of the values of a certain sensor.

RELATED INFORMATION	Storage
Priority:	High
Performance	Must be able to store at least 150000 signals per minute.
Frequency	Every time a signal is received.

USE CASE 5	Show sensors' value	
Goal in Context	System shows sensors' value to the manager	
Preconditions	Manager is logged in dashboard	
Success End Condition	The Manager is correctly shown current sensors' value	
Failed End Condition	Error message	
Primary	Building Manager, Area Manager, Urban Manager	
Trigger	Manager Login	
DESCRIPTION	Step	Action
	1	Manager logs in dashboard
	2	System evaluates current sensors' value and label it with different colours considering the treshold value
	3	System shows elaborated data to Manager
EXTENSIONS	Step	Branching Action
	1a	Values over the imposed threshold detected : Alert of threshold value

USE CASE 6	Expand area view	
Goal in Context	Manager can see a section of the area of expertise in detail	
Preconditions	Manager is logged in and dashboard is showing data	
Success End Condition	Manager can evaluate detail detected from sensors of an area	
Failed End Condition	Manager can't focus on a particular area	
Primary	Manager	
Trigger	Manager select an area of interest	
DESCRIPTION	Step	Action
	1	Manager logs in dashboard
	2	System shows entire area sensors' value
	3	Manager selects a section of area
	4	System shows details of selected area

USE CASE 7	Manage Sensors' Properties	
Goal in Context	Admin can be able to change threshold value fo each sensor	
Preconditions	Admin logged in dashboard	
Success End Condition	Admin have modified threshold values of sensors	
Failed End Condition	Error Message	
Primary	Admin	
Trigger	Admin clicks on settings panel in dashboard	
DESCRIPTION	Step	Action
	1	Admin logs in dashboard
	2	Admin opens setting panel
	3	Admin selects a sensor
	4	Admin change threshold value for the selected sensor

USE CASE 8	Check Manager Type	
Goal in Context	Understand which kind of manager has sent a request in order to show the related sensors.	
Preconditions	The system is up and running	
Success End Condition	The values of the correct sensors are shown to the manager	
Primary,	Manager	
Trigger	A manager logs in the system	
DESCRIPTION	Step	Action
	1	The Manager logs in the system
	2	System checks if Manager account exists, and which kind of manager they are.
	3	System returns a view with the values of the sensors that manager is allowed to control.

RELATED INFORMATION	Check Manager Type
Priority:	Medium
Frequency	Every time a manager logs in.

USE CASE 9	Grant manager account	
Goal in Context	Admin can eventually create a new manager account	
Success End Condition	A new manager account is created, and that person is now able to log in the system.	
Primary	Admin	
Trigger	Admin presses the related button.	
DESCRIPTION	Step	Action
	1	A new manager account is needed for another person joining the system.
	2	Admin presses the button and inserts the data for that specific person.
	3	That person is now able to log in the system with those credentials.

A1.1 GUI Requirements (da riempire a partire dalla Versione 2)

The functional requirements that are related to the GUI are:

- *Unusual values must be shown with appropriate colours based on their priority*
- *Values over the threshold must be shown explicitly*
- *Dashboard must show all sensors*
- *The user must be able to select a zone to restrict the sensors displayed only to those in that specific zone.*

Our GUI must be crystal clear and as immediate, evident and user-friendly as possible. For this reason, we added Clarity and Straightforwardness in our non-functional Requirements.

A1.2 Business Logic Requirements (da riempire a partire dalla Versione 2)

The functional requirements that are related to the Business Logic are:

- *Admin must be able to change the defined threshold values*
- *The system must be able to detect sensors' failures and warnings. If there are backup sensors for a measured property, the warning will have a lower priority.*
- *Different managers are given information with different levels of detail, based on a hierarchic relationship.*
- *There are 3 types of managers, related to the zones they have to monitor (building, area, city).*
- *If a sensor detects unusual values, its frequency must be updated.*

Our main concerns here are security and efficiency. They're always inversely proportional, and it is cardinal for us to find the right balance. They were both added to our non-functional requirements and explained in greater detail in the related section.

That means our system must:
- Check if the values are above the threshold
- Instruct the server to increase their frequency
- Store the data in the Values Cache or Warning Cache
- Store the data in the Time Series DB
- Be able to insert new sensors in the Relational DB
- Send the correct data to the clients
- Allow to insert or remove Managers

A1.3 DB Requirements (da riempire a partire dalla Versione 2)

There are no explicit functional requirements that relate to the DB.
Efficiency: The main (non-functional) requirement we have for our Databases is that they have to allow our system to handle at least 150000 signals per minute. We know the DBs are the slowest part of the system and we have to be especially careful in their management.
Consistency: It is also cardinal for us that the two DBs we use are consistent with each other. The system must also keep running in case of hardware fault, so we must keep backup DBs running.
Our Databases must store:
- Zones, including Areas, Buildings, Floors and Rooms.
- All sensors, with their ID.
- All signals received (Time Series)
- All managers account, with their name, informations, and role.

A.2 Non Functional Requirements

1. **Efficiency**

    We must handle *at least* 150000 signals per minute. Because of this, we chose to use two different DBs: a relational DB and a Time Series one. This choice was made because Time series Databases are extremely efficient for our purpose.
    It is important we query the relational Database as little as possible, because those accesses can decrease performances rather heavily if not optimized. In order to reduce unneeded accesses to the databases the system shall keep a cache in memory for nearly everything, this also makes the system more resilient and fault tolerant.
    But that isn't enough, we also had to optimize our server and communication methods: packets should be as small as possible in order to be processed faster and better. That is why, when a sensor is activated for the first time, it communicates with a different port sending all the initial information we need and storing them in the relational Database. From that moment onward, the sensor starts communicating with another port that only accepts simple packets containing the Sensor ID and the Value detected.
    This way we don't have to check if the packet is from a new sensor or an "old" sensor each time we receive one.

2. Usability

The GUI must be crystal clear and as immediate as possible. In our system, that translates to the manager being able to identify issues the moment they come up. Warnings must be explicit and unmissable.

Values must be updated real time and show clear, bright colors (green, orange and red) depending on the level of danger.

As this system is not meant for commercial use, **clarity must have priority over visual appeal**. Menus should be clear, not cluttered, and even a non-tech savvy manager must be able to use the system at its fullest potential with ease. We don't know whether the managers are engineers or just simple workers - therefore we have to keep it simple, fast, and efficient.

3. Security

Ill-intentioned people might break into our system and send false data to the server: this can cause erroneous or missed warnings and possibly greater risks. We must keep the connection between the sensors and the server as secure as possible, while also being able to guarantee the system's efficiency and fulfilling its requirements. Connection between Server and Client, or server and the two DBs, should also be secure, but assuring the correctness of our data takes priority. This comes down to choosing the right balance for efficiency and security: tests are necessary to understand which protocol is the fittest for our project.

4. Scalability

We assume the system is constantly growing, therefore new sensors could be installed rather frequently. Our system should be able to grow while maintaining the same efficiency. This is done by separating our Model (stored in the relational DB) from the signal handling, maintained in a Time Series Database. This non-functional requirement is also heavily tied to *Efficiency*. Having an efficient and scalable system, also means having a cheaper system both to set up and to operate.

A.3 Excluded Requirements

1) **Emergency Management.** The system does not provide any functionality to interact with emergencies (e.g., calling firefighters). It is only used to check the data the sensors are sending.

2) **Portability.** Our system is very specific and must work with a particular kind of sensors that are preconfigured for a certain behavior. It would be hard to code modules

that are generic for any data-collection software without lowing performances, so it won't be our goal for now.

3) **Legal Requirements.** We assume sensors are placed according to the law and no invasion of privacy is perpetuated, so we will ignore this for now.

A.4 Assumptions

1) We assume the sensors are already in place and their positioning is reasonable. A sensor placed near a fireplace would probably show unusual values when it is not needed.

2) "Alert" and "Warning" have very different meanings in our system. We have an alert when one type of parameter (e.g.: temperature) has a value that is over the threshold. A warning is issued when two or more types of parameters are above the threshold, and the manager must be alerted immediately.

3) We assume we have no budget limit, as it was not explicitly expressed in the requirements. Of course, we still tried to plan smart and without waste.

4) We assume sensors are configured to have a unique ID that does not already exist in our system, and that can be communicated to our server to a certain port.

5) We assume sensors are placed according to the law and no invasion of privacy is perpetuated.

A.5 Prioritization

Here is the list of the functional requirements in priority order:

1. Unusual values must be shown with appropriate colors based on their priority.
2. Values over the defined threshold must be shown explicitly.
3. If a sensor detects unusual data, its frequency must be updated.
4. There are three types of managers, related to the zones they have to monitor.
5. Dashboard must show all sensors.
6. Admin must be able to change the defined thresholds.
7. The system must be able to detect sensors' failures and warnings. If there are backup sensors for a measured property, the waring will have a lower priority.
8. Different managers are given information with different levels of detail, based on a hierarchical relationship.
9. The user must be able to select a zone to restrict the sensors displayed only to those in that specific zone.
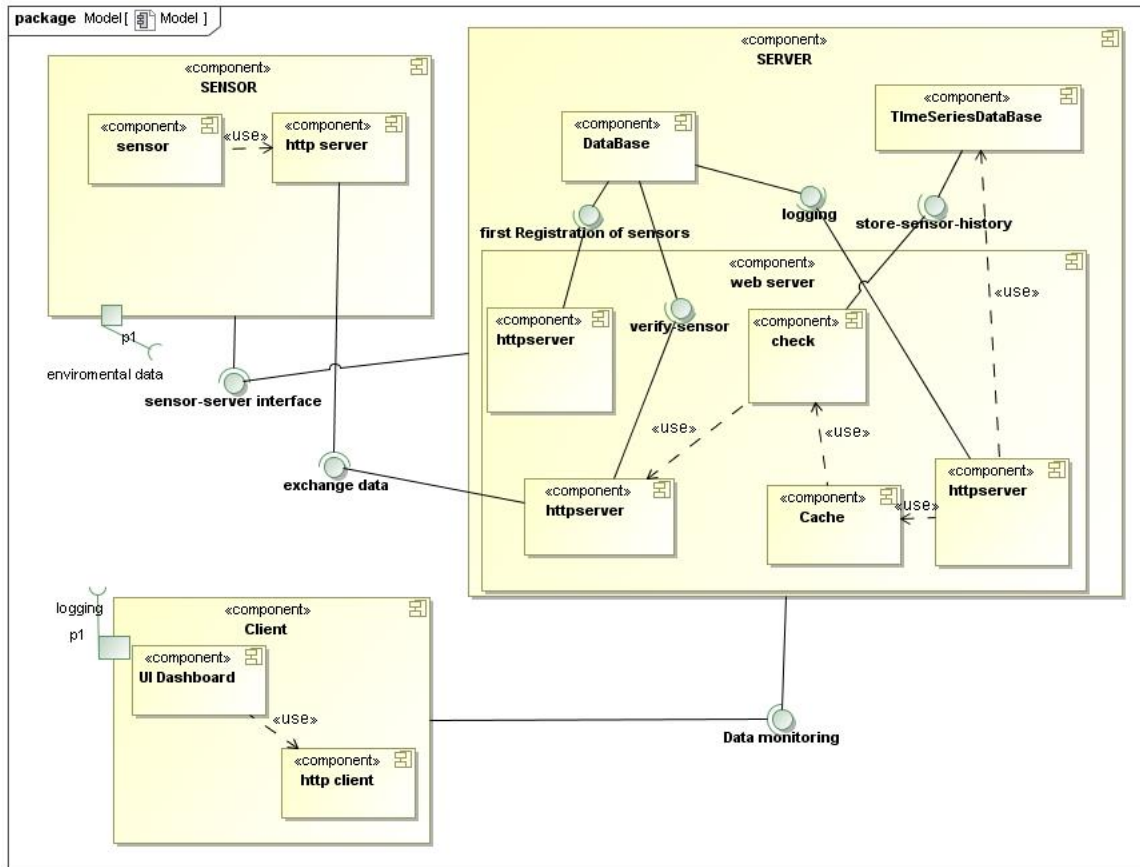
B.1 The static view of the system: Component Diagram

Figure 3: Component Diagram

The updated Component Diagram replaces NetData with an architecture of our own. Just like last time, the interactions are simplified to show the logical subdivision without too many technical details. The three main components are the Sensor, the Server and the Client. The Server manages and checks all of the signals before making them available for the Clients.

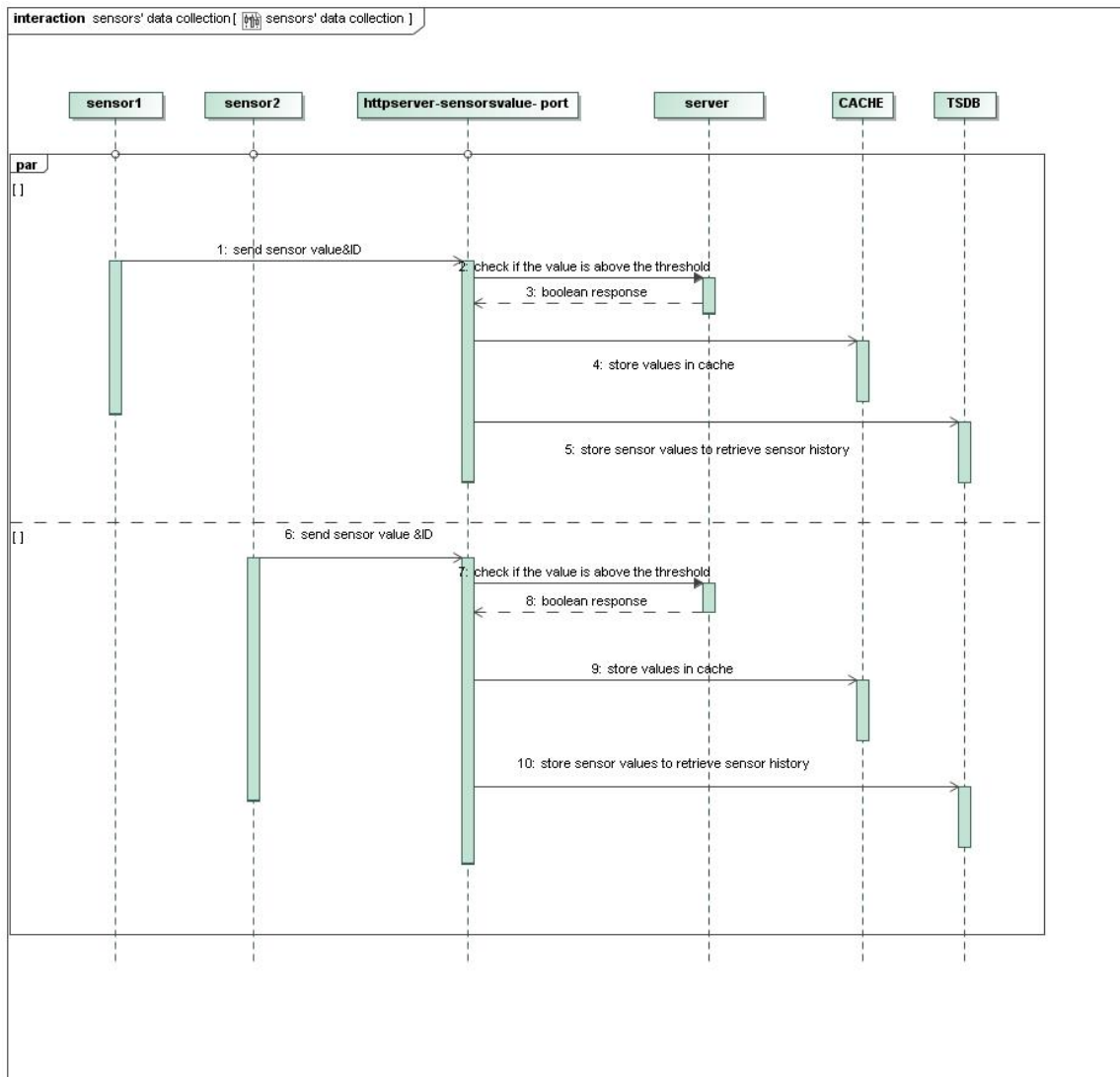## B.2 The dynamic view of the software architecture: Sequence Diagram

Figure 4: Sensors' data collection Sequence Diagram

This diagram shows the parallel data collection from the sensors, it states clear the importance in our system of the cache(s) and the other in-memory structures as they allow us to defer writes to the databases while still keep sending updated sensor values and status to the clients. We use different ports for receiving values and for registering new sensors. The use of HTTP allows fast prototyping and should suffice the performance requirements but in case of necessity can be easily replaced with plain TCP communication as it is loosely coupled with the rest of the system.
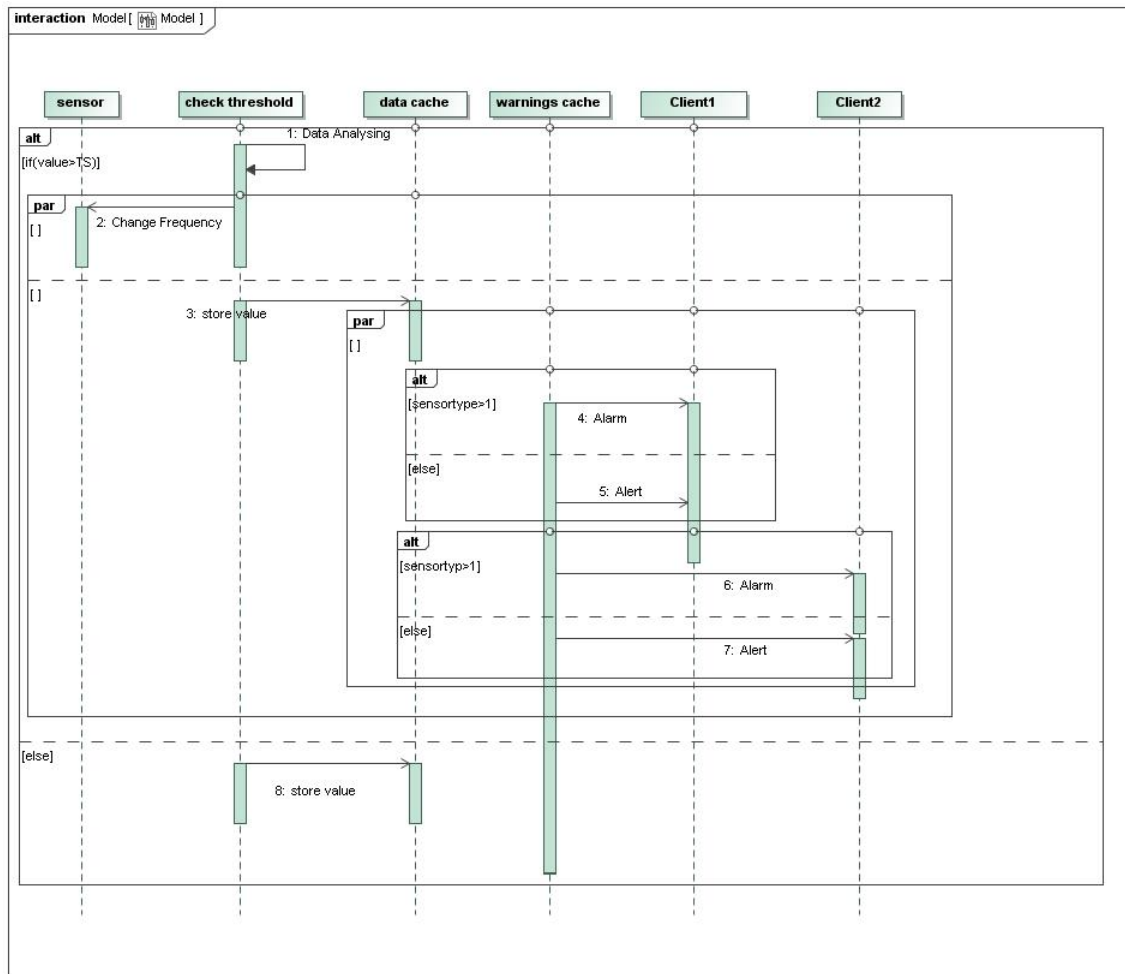
Figure 5: Data Processing Sequence Diagram

Here we show the parallel processing of the data received from the sensors and the streaming (along with eventual warnings and alerts) to the clients of users to whom the sensor is relevant (so the managers of zones containing that sensor).
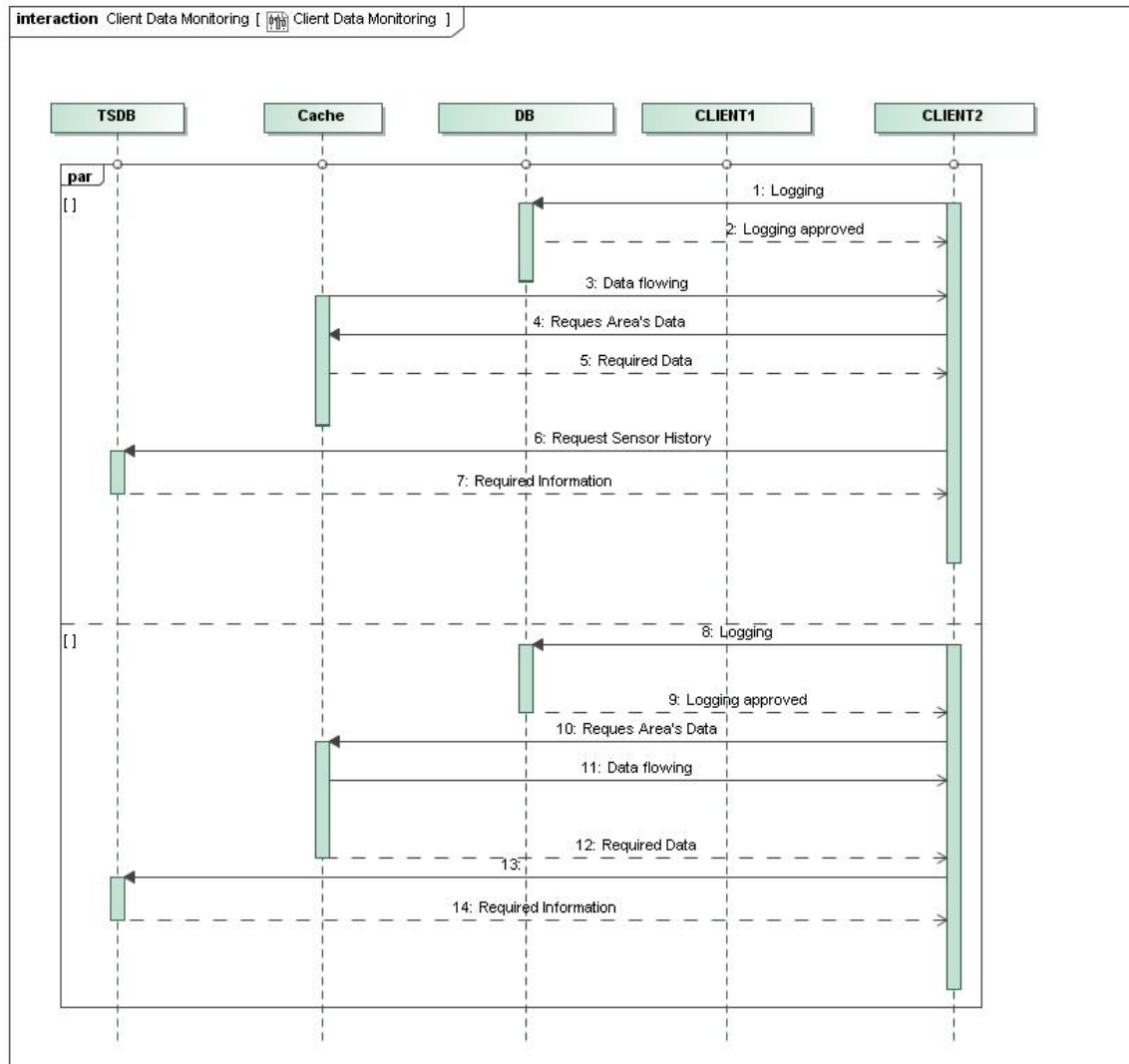
Figure 6: Client Data Monitoring Sequence Diagram

This diagram shows how the client receives data from the server. Once the manager is logged in, his dashboard displays and continuously updates status and values for all the sensors of his interest. If the manager wants to restrict the zone he's displaying, the client sends his request to the server, which from this moment on will only send the appropriate zone's values (until a new zone request is made from the client). The dashboard only shows the last known value for each sensor unless the user requests the values history for a specific sensor. In this case the server retrieves the data from the TSDB and sends it to the client.
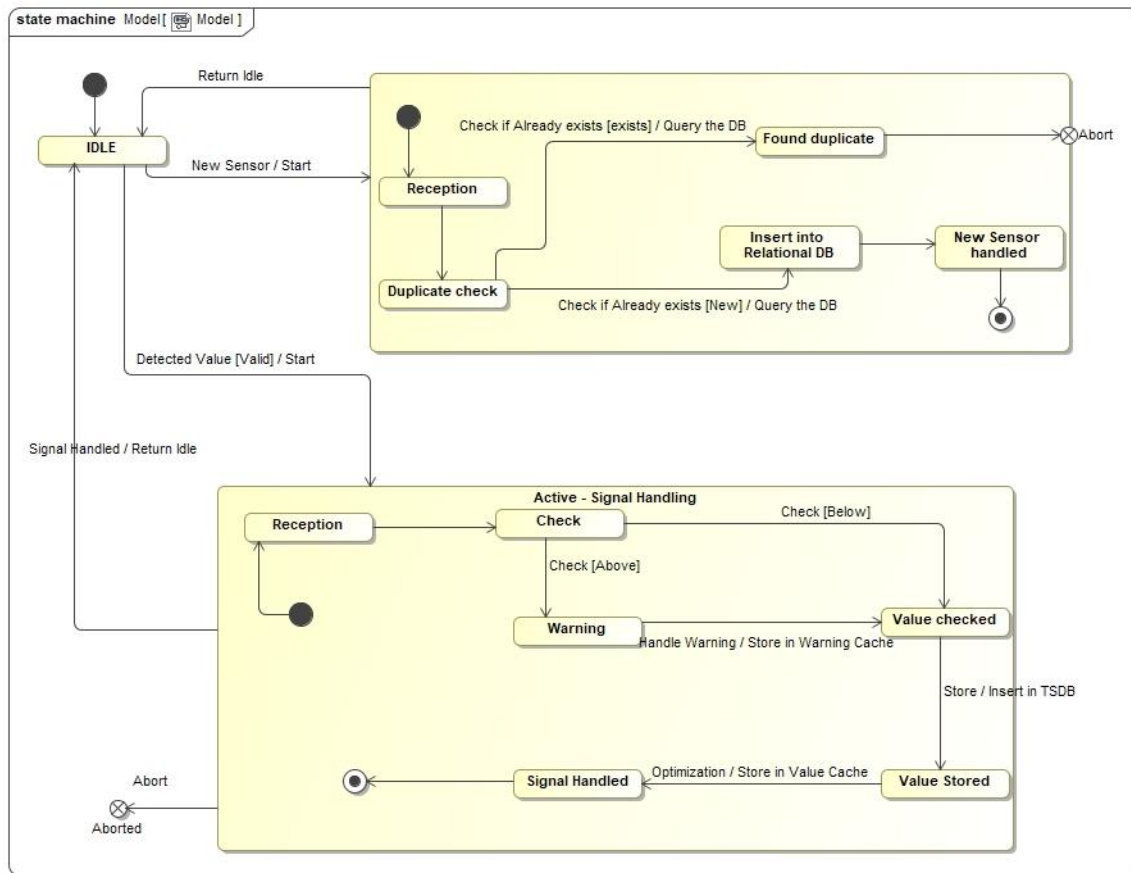
Figure 7: Server State Machine Diagram

This state machine Diagram depicts the server and its basic functions **that are related to the sensor**. First of all, the server behaves differently if he receives a "normal" Value packet or the packet from a new sensor that wants to be recognized. In the former case, the value goes through a check and then gets stored both in the cache and Time Series DB. This is basically a server-focused view of what was illustrated in Figure 4. If the server receives a New Sensor request, it checks if we already have a sensor with that ID in the relational DB and eventually inserts a new one.

Normally, a State Machine Diagram would also have internal operations that do not cause a transition from one state to another. Our server, however, operates a series of actions that cannot be paused, so it never stays in a certain state for too long.

C. ER Design

<Report here the Entity Relationship Diagram of the system DB>
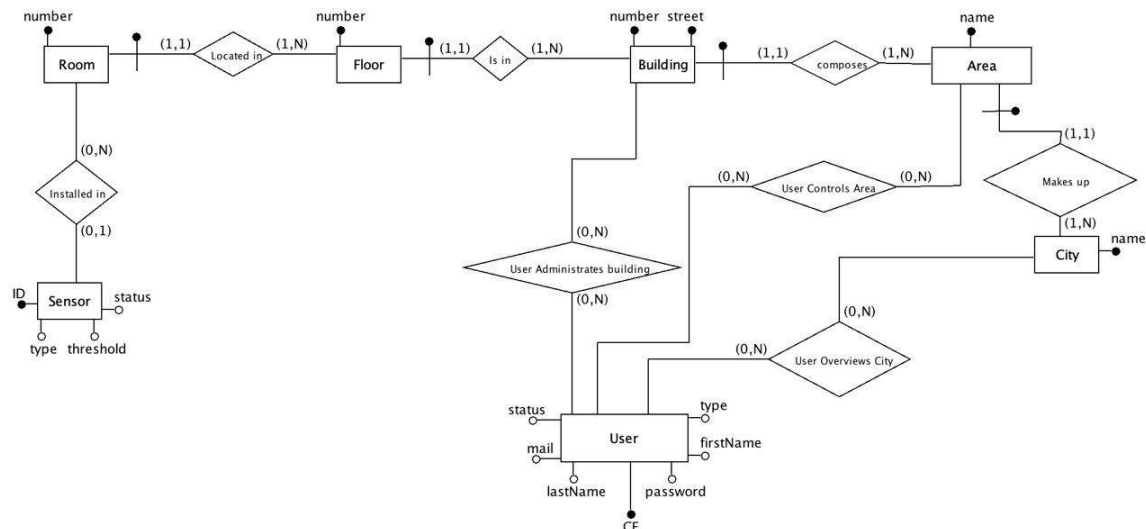

Figure 8: ER Diagram

The ER diagram has already been restructured and ready to be transformed into the corresponding relational model. The basic entity that constitutes the system is the sensor, while rooms, floors, buildings, areas and cities are elements of aggregations of sensors necessary for the location of the sensors as well as for the attribution of the different types of areas monitored to the various types of managers. For security reasons, user passwords will be stored in encryption to avoid unwanted access as much as possible. To speed up indexes and to make the system more robust, the keys identified in the ER diagram will be used as unique constraints at the implementation level and as IDs will be used auto-increment integer.

D. Class Diagram of the implemented System
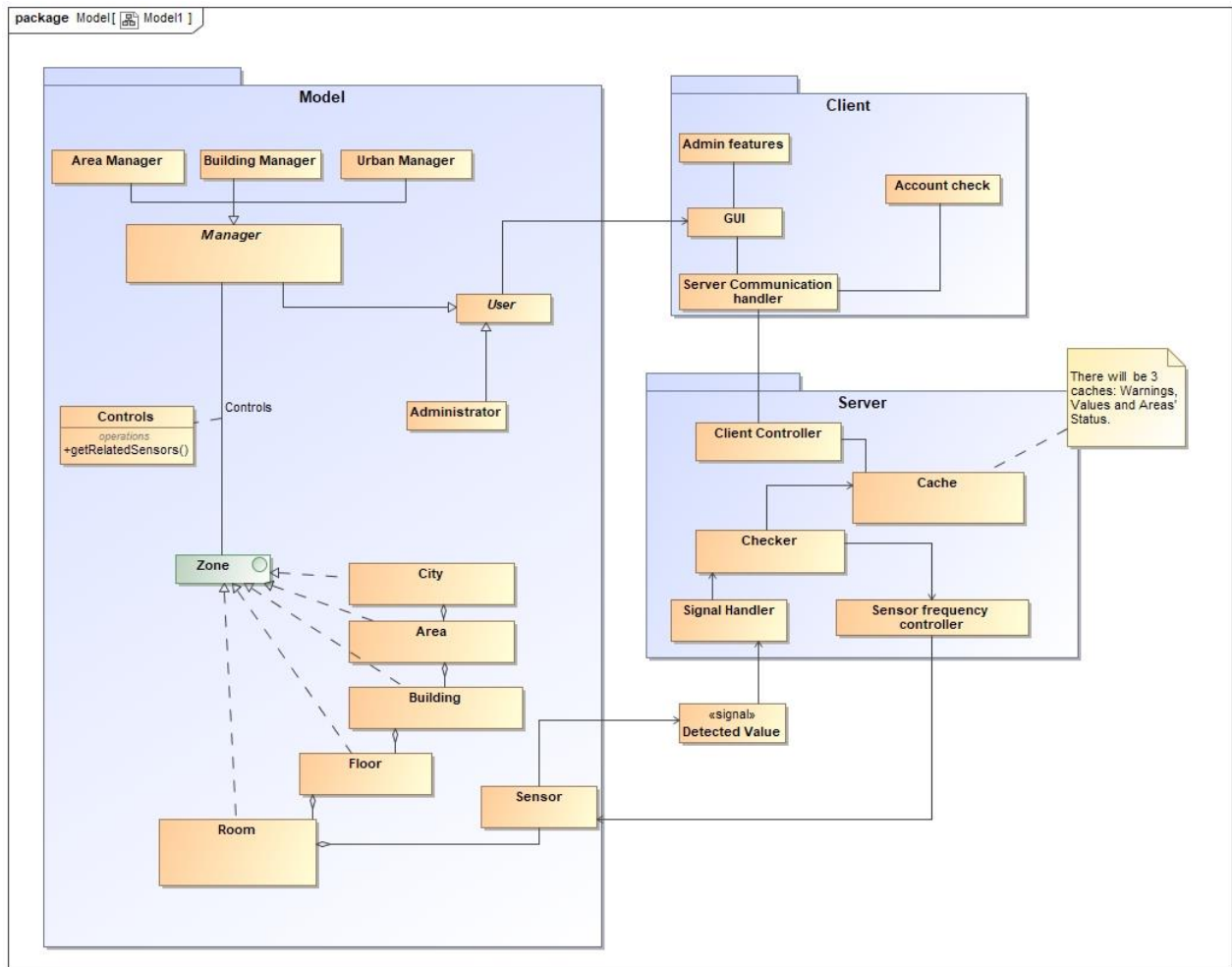
Figure 9: Overview of the Class Diagram

This figure is an overview of our Class Diagram. It does not represent attributes, operations, or other details as it is only meant as an aid to quickly grasp the basics of our system. Figure 12 will go in details about all of the classes presented here. Packages are very similar to the main components of figure 3, the component diagram.

The Class Diagram was made starting from the Use Case Diagrams and modeling each Use Case with the necessary classes to implement that feature.

Figure 10: Class Diagram
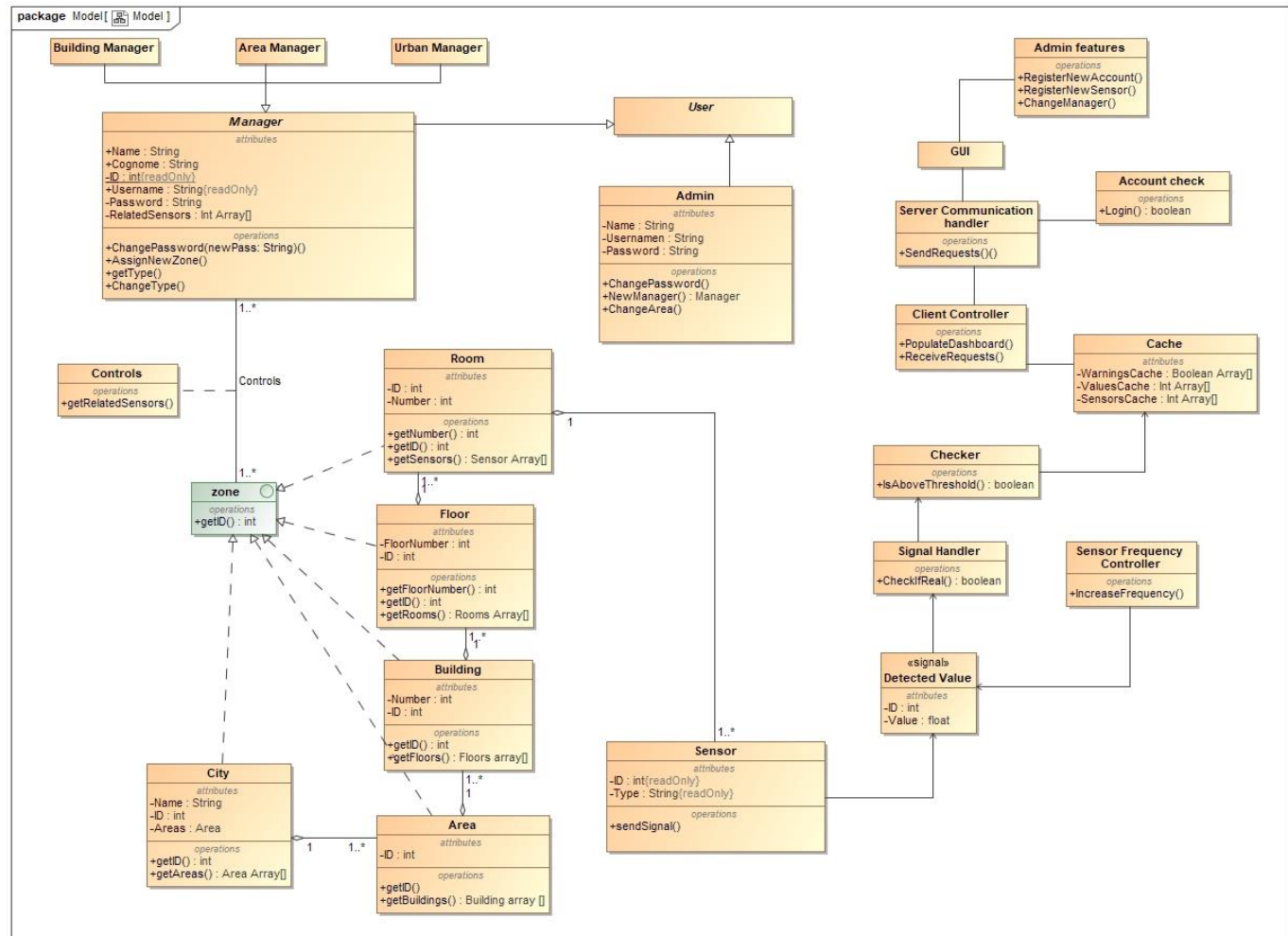
This is an in-depth view of our Class Diagram. It does not only show the Model part of our system, but the main components of the Control and View (GUI) segments as well.

The model is clearly very similar to the one illustrated in the ER Design.

It is also important to note that both Manager and User are abstract classes, thus cannot be instantiated. The class relationship "Controls" uses both the data from Manager and Zone to understand which sensors a certain manager is able to control. Sensor and Room have an Aggregation relationship because in our system, rooms only exist if at least one sensor is in them. As there are no empty rooms, we can say a room is composed of 1 or many sensors.
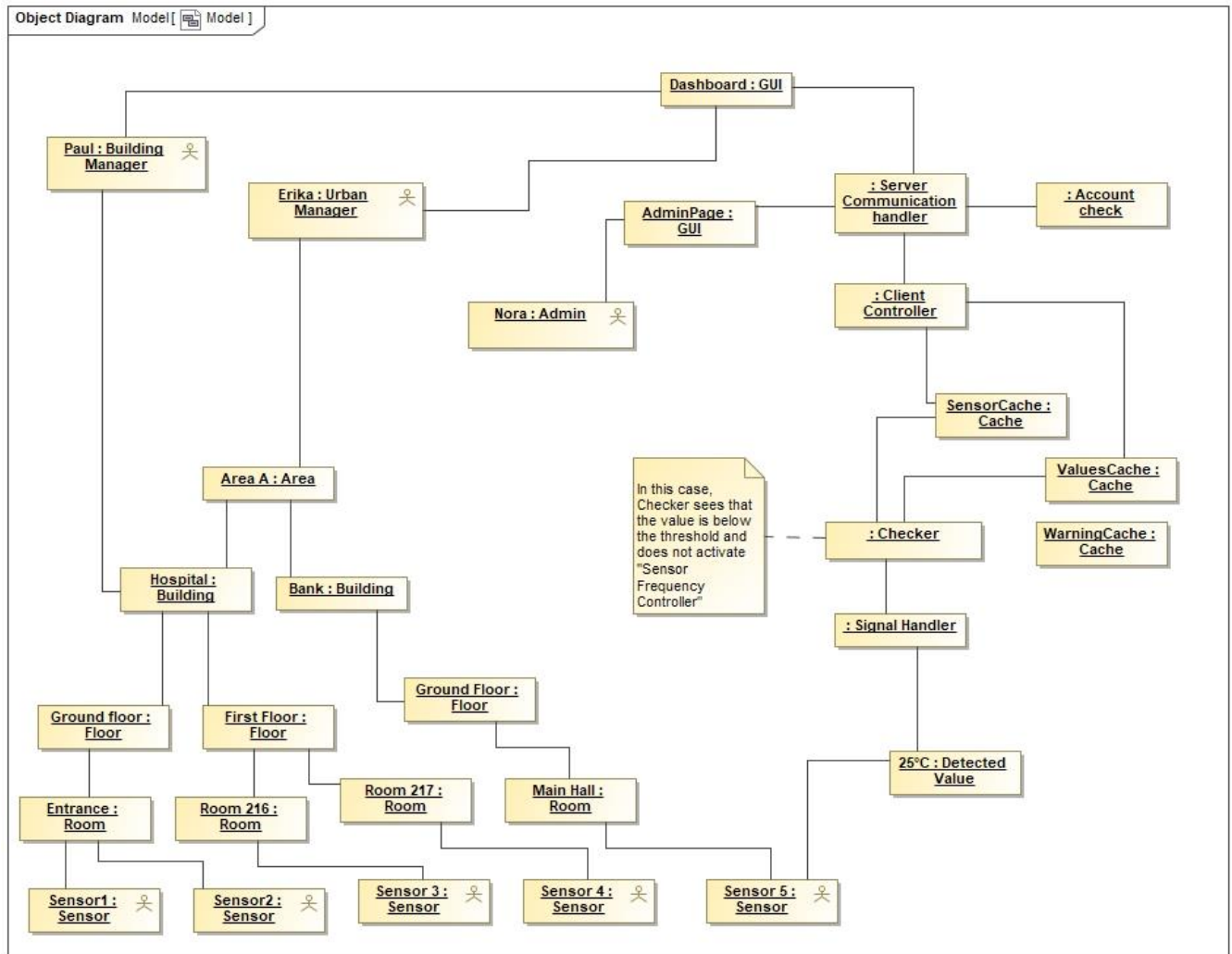
Figure 11: Object Diagram

Figure 13 represents an Object diagram, an instance of the Class diagram of figure 12. For coherence, we used the same names from the scenarios illustrated at the beginning of this documentation. So Paul is the Building Manager that has to control the hospital, Erika is still the Urban Manager, and Nora is the admin. As we can see the building is composed of many floors, which are composed of many rooms, and a room only "exists" in our system if at least one sensor is in it. For example, sensor 5 detects a temperature value of 25 degrees, and sent the signal to the Control part of the system that will handle it and eventually show it to the clients.

E. Design Decisions

1. NOT using NetData

   We were perfectly conscious of the risks related with our previous decision of using netdata, so we started as earlier as possible to test the needed technologies. While we had good results at first, the effort revealed to be unsustainable (at least with the time constrains we had). The workaround to the intrinsically synchronous nature (given we cannot query sensors for values) of netdata we thought could be done in a few hours may instead have required a lot of time due to the way netdata handles node.js plugins by default (we omit further discussion about a topic no more related to the project). After all the time spent on this was not entirely wasted as we learned a lot about the architecture and we perfected part of the design thank to the knowledge acquired.

2. Administrator User

   As the managers are meant to just control the sensors' values, we also chose to insert an Administrator as a distinct user beside the managers. One or more administrator users take care of tuning sensors threshold values, add new manager and grant permissions (e.g. assign the zones to the proper managers). This also simplifies the implementation of the GUI, moving all unneeded functions outside the dashboards of the manager users.

3. Time Series Database

   To maximize the efficiency of the system we decided to store the values of the sensors in a Time Series Database (TSDB), in the specific InfluxDB. This type of databases is specifically designed for the task of storing a great number of values sequentially (like the values received from a sensor). It is extremely fast for this kind of task and the management is easier than using a SQL database or many more common NoSQL databases. Of course, we also have a separate relational database where the systems stores all the structured information it needs to work properly (e.g. information and about sensors, zones, users and relationships between those).

4. In-memory Cache

   In order to allow quasi-real-time performance in the forwarding of the sensors' values to the clients, we decided to include various caches. Ideally our system should be able to perform all the operations needed without worrying about the databases being too slow or not working at all. So we included caches for sensors' values, zones' status, warnings (and alerts), currently loaded users' information

(e.g. relevant sensors and zones for a given user). All of those can be accessed and processed in a massively parallel fashion to collect data from sensors and deliver it to the client efficiently.

5. Java, Javascript and HTTP

Having spent too much time experimenting (mostly unsuccessfully) with new technologies to include in the project, we cannot take any more risk to run behind schedule. Because of this we all agreed that the best choice would have been to use technologies well known to everyone in the group. We decided to write the server side entirely in Java and a web client in JavaScript. All the communications in the system are done using HTTP, also from the sensors. This makes the development easier and faster, but it will certainly become a bottleneck if the sensors' number grows well past the requirements. To avoid problems, in the future, the HTTP server that collects data from sensors can be replaced to reduce overhead with nearly no modifications to the rest of the system.

F. Explain how the FRs and the NFRs are satisfied by design

A1.1 GUI requirements

GUI was created using HTML5, CSS and JavaScript.
The GUI has been implemented in a way that it can be accessed via the web, thanks to the implementation of a server created through Java and Java Servlet that can be reached remotely. This allows managers to view the dashboard at any time by knowing the server's IP address.

• Unusual values must be shown with appropriate colours based on their priority

For unusual values classified "Alert" (see assumption #2) we add a "orange" background color to the box(intented as a room box if on dashboard is logged in a building manager , intented as a building box if on dashboard is logged in an area manager and etc.) that contains the sensors that is over threshold . For unusual values classified "Warning" (see the same assumption) we do the same operation after have done risks evaluation but colouring the box with a "red" color.

• Dashboard must show all sensors

Every manager depending on his role have a different dashboard,operations performed by managers like login and data request are made by using an http client that sends/receives data from/to server using a GET request.

• The user must be able to select an area

On a relational DB (MySql database) are stored managers' permission so that every manager see only the areas assigned to him and with the right level of detail. A building will be able to choose between the various buildings assigned to him, an area manager among the various areas assigned to him and in the same way an urban administrator can select one of the cities attested to himwill specifically see a division for floors and rooms. Focusing on a room, he will be able to see specifically all the sensors availables in the single room. The same reasoning is applied to all types of managers who, however, will receive information grouped together with the growth of the vastness of the monitored infrastructure with the possibility of expanding to go into details.

A1.2 Business Logic Requirements

• *Admin must be able to change the defined threshold values*

 The admin, by connecting to the dashboard using the credentials, can check the thresholds of the sensors through a http call to the server and change the value if necessary, for each sensor the threshold level is stored in the relational DB.

• *The system must be able to detect sensors' failures and warnings. If there are backup sensors for a measured property, the warning will have a lower priority*.

 The server will receive messages from the various sensors through an http call. For each message received, it will verify the authenticity of the sensor by checking the relation DB in which the sensor identities are recorded. A "check" control module which resides on the server will check if it receives from each sensor (at least once a minute under normal conditions) a message and will compare the measured value that the sensor will send through the message with the maximum threshold value preloaded in a cache inside the server to be as efficient as possible in the comparison. The failure of a type of sensor will be shown in orange (considering an alert situation) if in the same environment there are other sensors that detect the same type of data otherwise in red (considering a warning situation)

• *Different managers are given information with different levels of detail, based on a hierarchic relationship.*

 A building manager will specifically see a division for floors and rooms. Focusing on a room, he will be able to see specifically all the sensors availables in the single room. The same reasoning is applied to all types of managers who, however, will receive information grouped together with the growth of the vastness of the monitored infrastructure with the possibility of expanding to go into details.

• *There are three types of managers.*

 In the relational DB each manager is assigned according to their role buildings or areas or cities to be monitored.

• *If a sensor detects unusual values, its frequency must be updated.*

 As the sensors can be configured, the request for data will be increased by the server up to a maximum of one detection every 5 seconds if the check module deems it necessary according to the analyzes carried out by the server

• A1.3 DB Requirements

It was decided to use a DB time series for storing the data collected by the sensors, which by its nature is suitable for storing this type of data and is optimized for handling time series data, that are arrays of numbers indexed by time (at datetime). For the management of roles, permissions and storage of active sensors in the system we decided instead to use a relational database (Mysql). Although slower than a NOSQL database since it is not used to a maximum if not in the login phase of the managers to recover access data and areas monitored by him, we have decided that it is the best technology for this purpose. To guarantee data preservation, we will certainly adopt backup systems for both databases in particular, the relational one because while the loss of old measurements present in the database time series (if not for analysis) does not represent a serious problem, the loss of the roles of the various managers can be serious and would prevent the access of the various managers. For security and efficiency, the latest detection measured by the individual sensors is stored in the server cache as well as in the DB to provide managers with less waiting time.

• Non-functional Requirements

1. *Efficiency*

The need to store many measurements per minute is guaranteed by a type of NOSQL server specifically designed for measurements called DBs time series. The server chosen in question is INFLUXDB which as reported in the official documentation *"Our influx-stress benchmark running on an AWS c4.4xlarge results in approximately 900,000 values per second on average."*

source: https://www.influxdata.com/blog/influxdb-1-1-released-with-up-to-60-performance-increase-and-new-query-functionality/
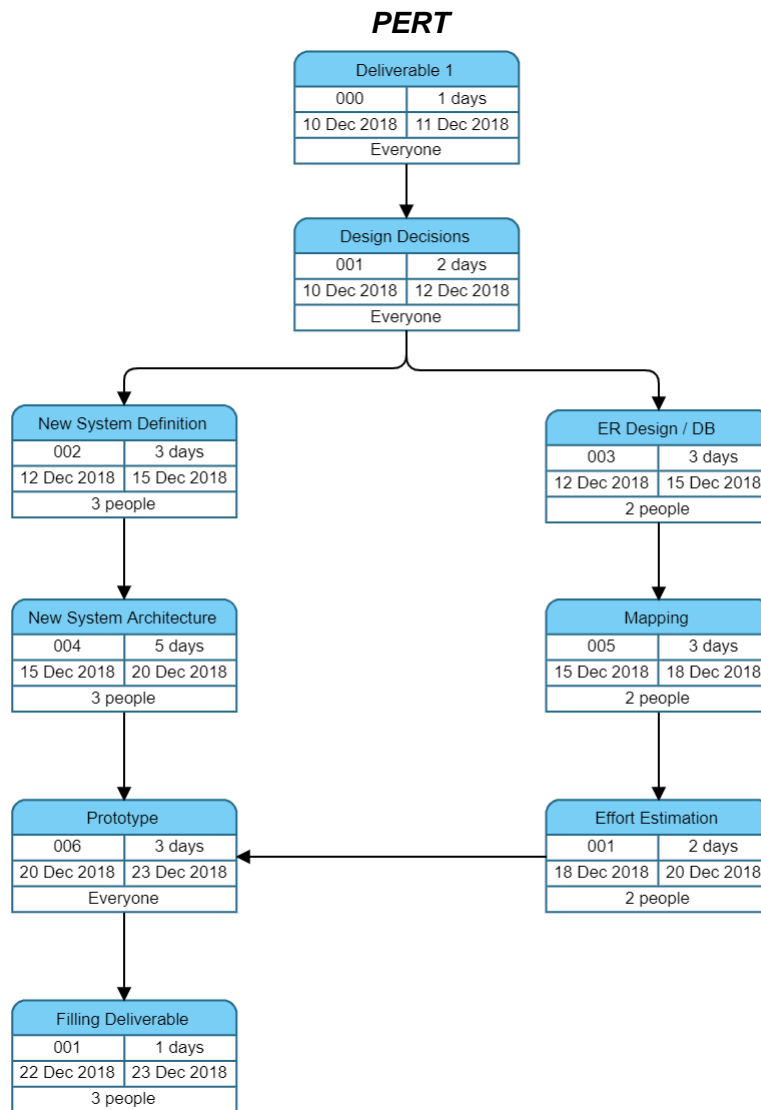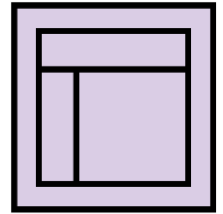
2. Security
We will use the https protocol to manage the communication between servers and users, in particular the access of an admin is very critical because criminals logging in as admin could lead to exorbitant levels sensors' thresholds so that catastrophic events can occur without the server being able to detect them and consequently without the managers being aware of it.

3. Scalability
The mysql DB will only store the identity of each individual sensor so we will have a tuple for each sensor that can be easily managed if the number of monitored sensors is increased. For storing the measurements detected by the sensors, up to 54,000,000 measurements per minute can be written to the NOSQL db.

G. Effort Recording

PERT

Deliverable 1	
000	1 days
10 Dec 2018	11 Dec 2018
Everyone	

Design Decisions	
001	2 days
10 Dec 2018	12 Dec 2018
Everyone	

New System Definition	
002	3 days
12 Dec 2018	15 Dec 2018
3 people	

ER Design / DB	
003	3 days
12 Dec 2018	15 Dec 2018
2 people	

New System Architecture	
004	5 days
15 Dec 2018	20 Dec 2018
3 people	

Mapping	
005	3 days
15 Dec 2018	18 Dec 2018
2 people	

Prototype	
006	3 days
20 Dec 2018	23 Dec 2018
Everyone	

Effort Estimation	
001	2 days
18 Dec 2018	20 Dec 2018
2 people	

Filling Deliverable	
001	1 days
22 Dec 2018	23 Dec 2018
3 people	

Time spent for the first deliverable: 55 hours

Time spent for DOING (first deliverable): 36.5 hours

Time spent for LEARNING (first deliverable): 18.5 hours

Note: Data is updated to 22/12/2018. Hours spent on the last day of the deliverable were not counted.

Time spent for the Second Deliverable: 110.5 hours

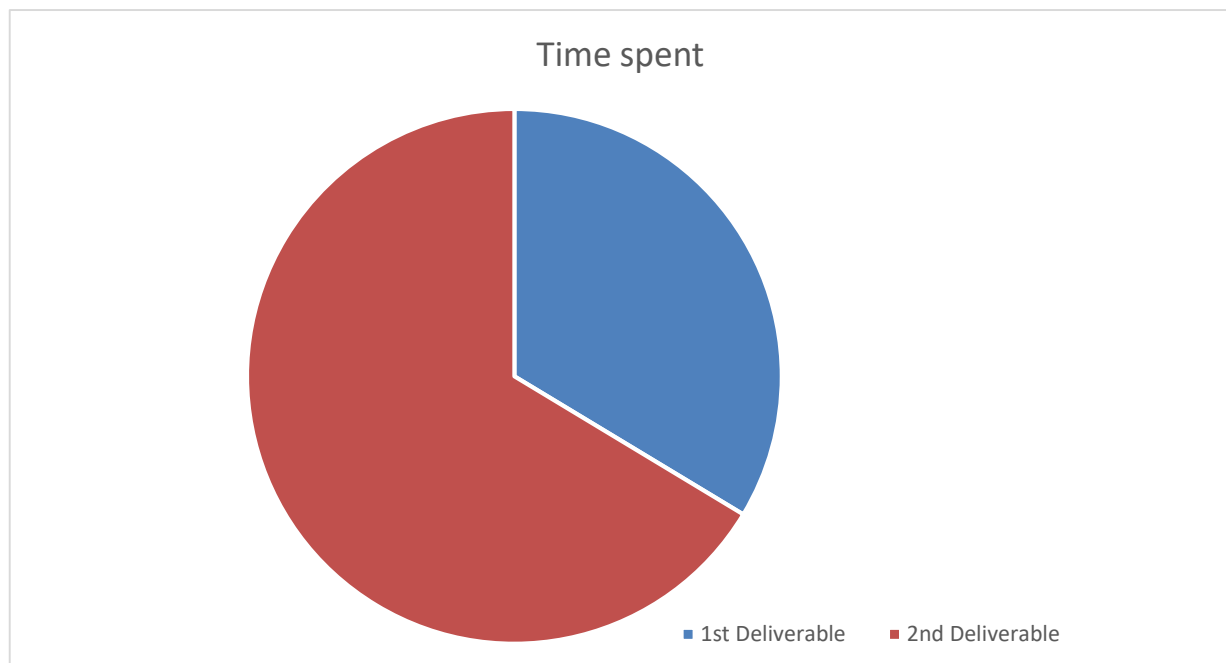*Time spent for DOING (Second Deliverable): 91.5 hours*

Time spent for LEARNING (Second Deliverable): 19 hours
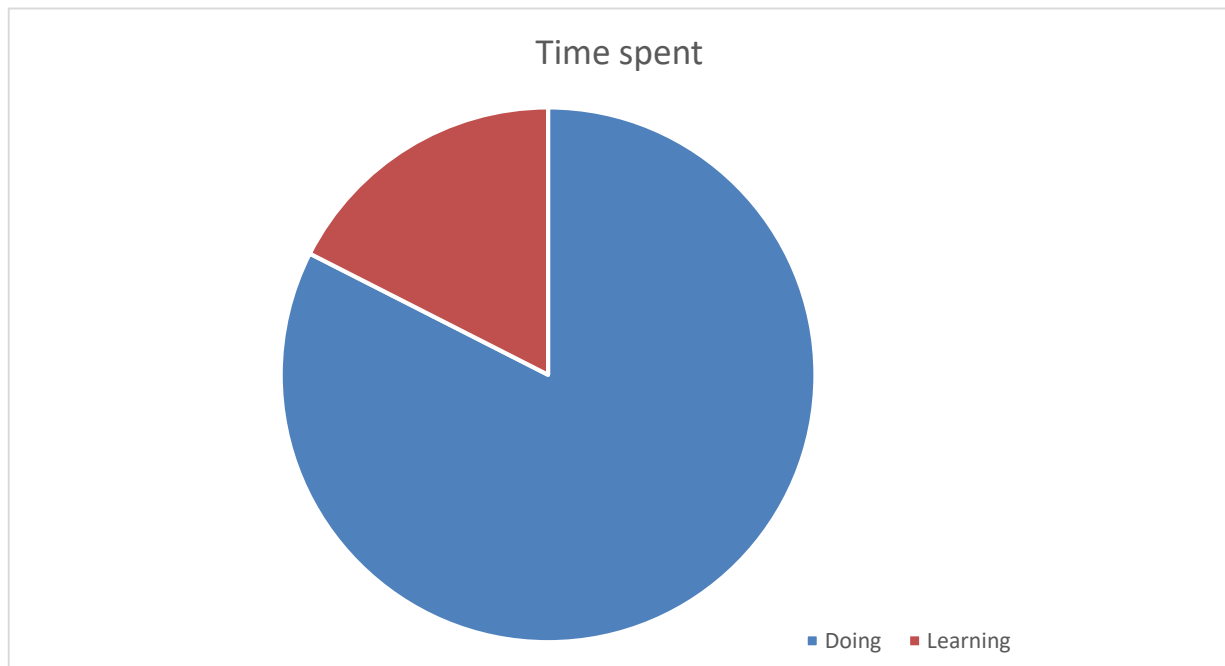
Time spent TOTAL: 165.5 hours

Time Spent for DOING (TOTAL): 128 hours

Time spent for LEARNING (TOTAL): 37.5

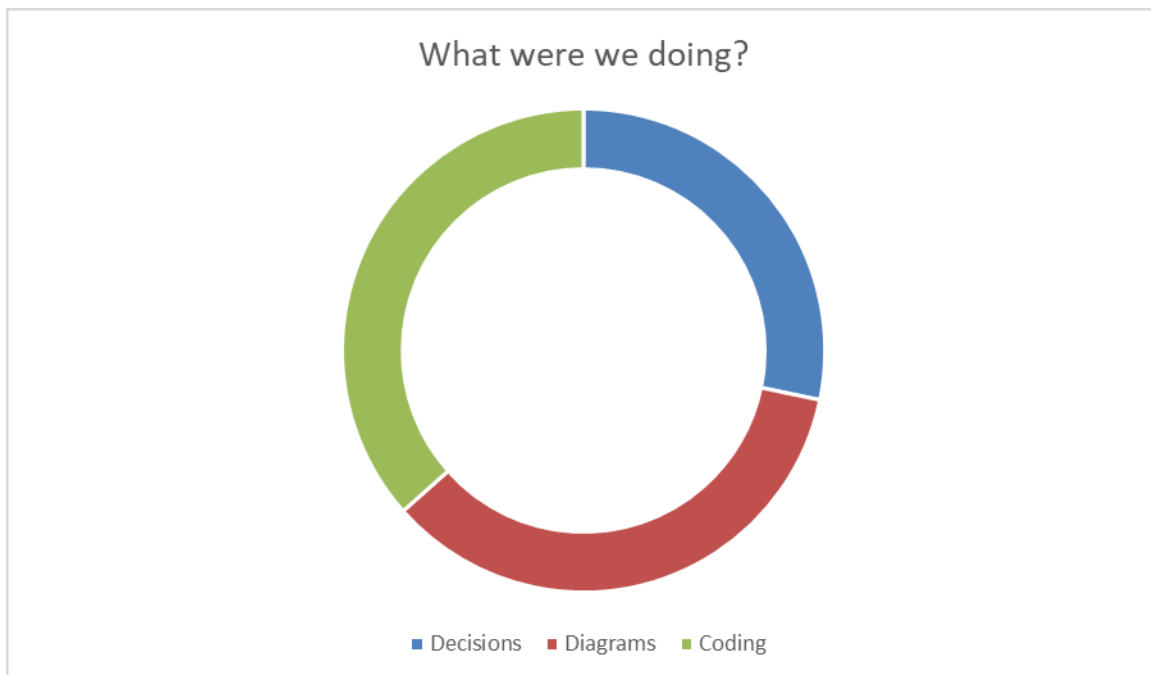**Time Partitioning**

Time spent

■ 1st Deliverable   ■ 2nd Deliverable

We spent way more time on the second deliverable, also due to the fact we chose not to use NetData and had to reinvent the system.

Time spent

We also spent way more on Doing than Learning. As mentioned, for the second deliverable we spent 19 hours on Learning and 89.5 hours on Doing.

What were we doing?

Half of the time spent on coding were used to code a Node.js plugin for NetData. When we realized the effort was going to be too much for a unsatisfying result, we chose to implement our own components. Overall, we spent a similar amount of time on Diagrams and Decisions. Class Diagrams were certainly the ones that took the most time, along with Sequence Diagrams.

Appendix. Code
REQUESTS SENDER

We have created a client application in java called RequestsSenders that simulate 150000 sensors.

When the application start, the class GenerateThreads creates 150000 instances of the class SensorSimulator.
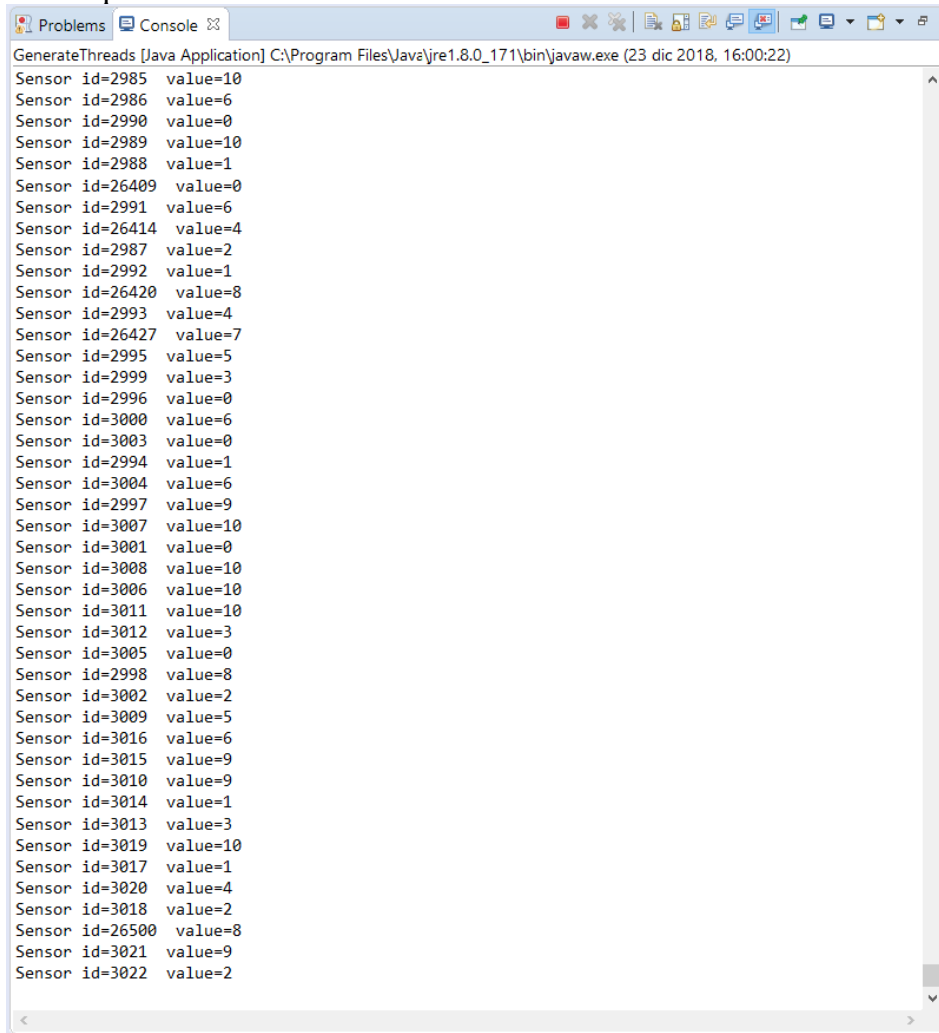
SensorSimulator sends an HTTP get request to the server(with its id and a random value), every 5000 milliseconds.

`"http://localhost:8080/Dashboard/RequestHandler?id="+id+"&value="+value`

This is the address for get request.

Note that in this case the server and the client run on the same machine.

The server response with a number that is the time that the sensor must wait to send the next request.
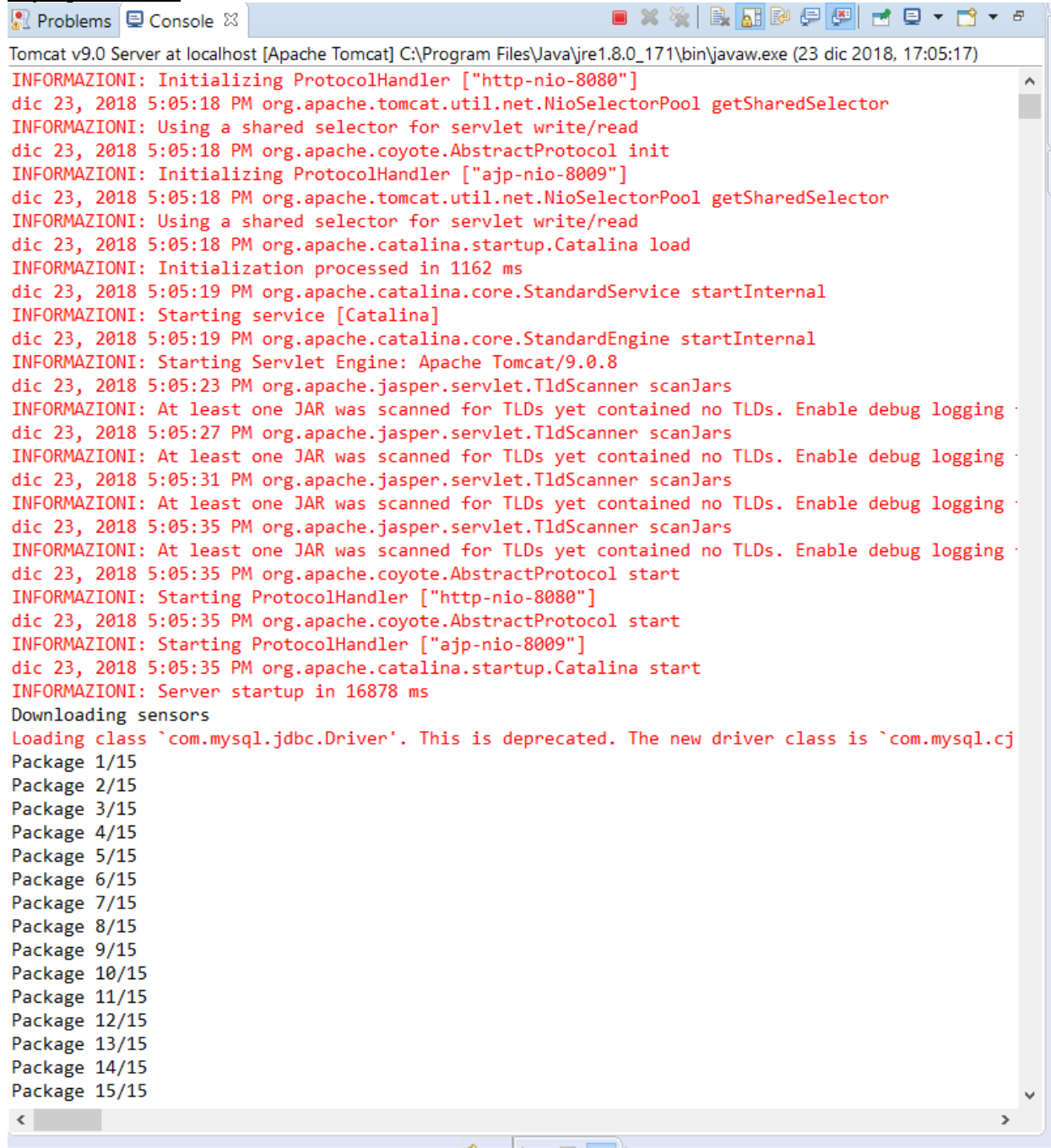
```
Problems    Console ☒                        ■ ✖ ✖ | ▤ ▥ ▣ ⊡ ▦ | ▧ ▥ ▾ ▭ ▾ ⊟

GenerateThreads [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (23 dic 2018, 16:00:22)
Sensor id=2985   value=10
Sensor id=2986   value=6
Sensor id=2990   value=0
Sensor id=2989   value=10
Sensor id=2988   value=1
Sensor id=26409  value=0
Sensor id=2991   value=6
Sensor id=26414  value=4
Sensor id=2987   value=2
Sensor id=2992   value=1
Sensor id=26420  value=8
Sensor id=2993   value=4
Sensor id=26427  value=7
Sensor id=2995   value=5
Sensor id=2999   value=3
Sensor id=2996   value=0
Sensor id=3000   value=6
Sensor id=3003   value=0
Sensor id=2994   value=1
Sensor id=3004   value=6
Sensor id=2997   value=9
Sensor id=3007   value=10
Sensor id=3001   value=0
Sensor id=3008   value=10
Sensor id=3006   value=10
Sensor id=3011   value=10
Sensor id=3012   value=3
Sensor id=3005   value=0
Sensor id=2998   value=8
Sensor id=3002   value=2
Sensor id=3009   value=5
Sensor id=3016   value=6
Sensor id=3015   value=9
Sensor id=3010   value=9
Sensor id=3014   value=1
Sensor id=3013   value=3
Sensor id=3019   value=10
Sensor id=3017   value=1
Sensor id=3020   value=4
Sensor id=3018   value=2
Sensor id=26500  value=8
Sensor id=3021   value=9
Sensor id=3022   value=2
```

Screenshot of console when the application is running

DASHBOARD

For the server side of the application we use a Tomcat webserver.
When server starts we must call the Start servlet that take sensor information from the MySqlDatabase.

```
Problems   Console ⌗                                         ■ ✖ ✖ | ▤ ▣ ▨ ⬚ ⬚ | ⬚ ⬚ ▾ ⬚ ▾ ⬚

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (23 dic 2018, 17:05:17)
INFORMAZIONI: Initializing ProtocolHandler ["http-nio-8080"]
dic 23, 2018 5:05:18 PM org.apache.tomcat.util.net.NioSelectorPool getSharedSelector
INFORMAZIONI: Using a shared selector for servlet write/read
dic 23, 2018 5:05:18 PM org.apache.coyote.AbstractProtocol init
INFORMAZIONI: Initializing ProtocolHandler ["ajp-nio-8009"]
dic 23, 2018 5:05:18 PM org.apache.tomcat.util.net.NioSelectorPool getSharedSelector
INFORMAZIONI: Using a shared selector for servlet write/read
dic 23, 2018 5:05:18 PM org.apache.catalina.startup.Catalina load
INFORMAZIONI: Initialization processed in 1162 ms
dic 23, 2018 5:05:19 PM org.apache.catalina.core.StandardService startInternal
INFORMAZIONI: Starting service [Catalina]
dic 23, 2018 5:05:19 PM org.apache.catalina.core.StandardEngine startInternal
INFORMAZIONI: Starting Servlet Engine: Apache Tomcat/9.0.8
dic 23, 2018 5:05:23 PM org.apache.jasper.servlet.TldScanner scanJars
INFORMAZIONI: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging
dic 23, 2018 5:05:27 PM org.apache.jasper.servlet.TldScanner scanJars
INFORMAZIONI: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging
dic 23, 2018 5:05:31 PM org.apache.jasper.servlet.TldScanner scanJars
INFORMAZIONI: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging
dic 23, 2018 5:05:35 PM org.apache.jasper.servlet.TldScanner scanJars
INFORMAZIONI: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging
dic 23, 2018 5:05:35 PM org.apache.coyote.AbstractProtocol start
INFORMAZIONI: Starting ProtocolHandler ["http-nio-8080"]
dic 23, 2018 5:05:35 PM org.apache.coyote.AbstractProtocol start
INFORMAZIONI: Starting ProtocolHandler ["ajp-nio-8009"]
dic 23, 2018 5:05:35 PM org.apache.catalina.startup.Catalina start
INFORMAZIONI: Server startup in 16878 ms
Downloading sensors
Loading class `com.mysql.jdbc.Driver`. This is deprecated. The new driver class is `com.mysql.cj
Package 1/15
Package 2/15
Package 3/15
Package 4/15
Package 5/15
Package 6/15
Package 7/15
Package 8/15
Package 9/15
Package 10/15
Package 11/15
Package 12/15
Package 13/15
Package 14/15
Package 15/15
```

We use the servlet `RequestHandler` for receving the sensor signal.
RequestHandler use a static method of the class Checker for control that the sensor value does not exceed the threshold.
Checker class access to the Cache for memorize the new value of the sensors.
The cache class has an array of Sensors width a size of 150000.
Each position of the array corresponds to a Sensor that has an id equal to the position of index. So a constant access to the array is guaranteed.
The access to the cache is controlled by a semaphore. And the operations of read and write are done by deep copy of objects for avoid access from references in other parts of the code.
At the same time we have a jsp for the dashboard to show to the user who accesses the server cache every second, updating the status of the sensors in the view.
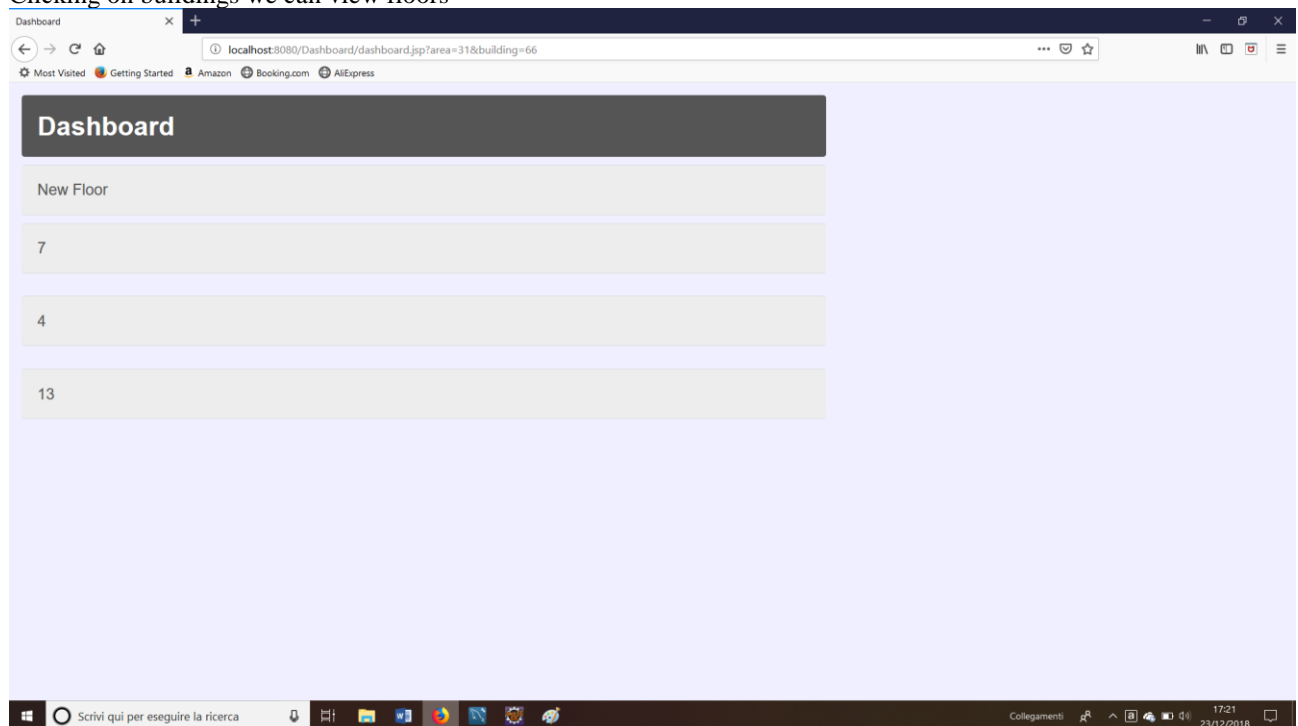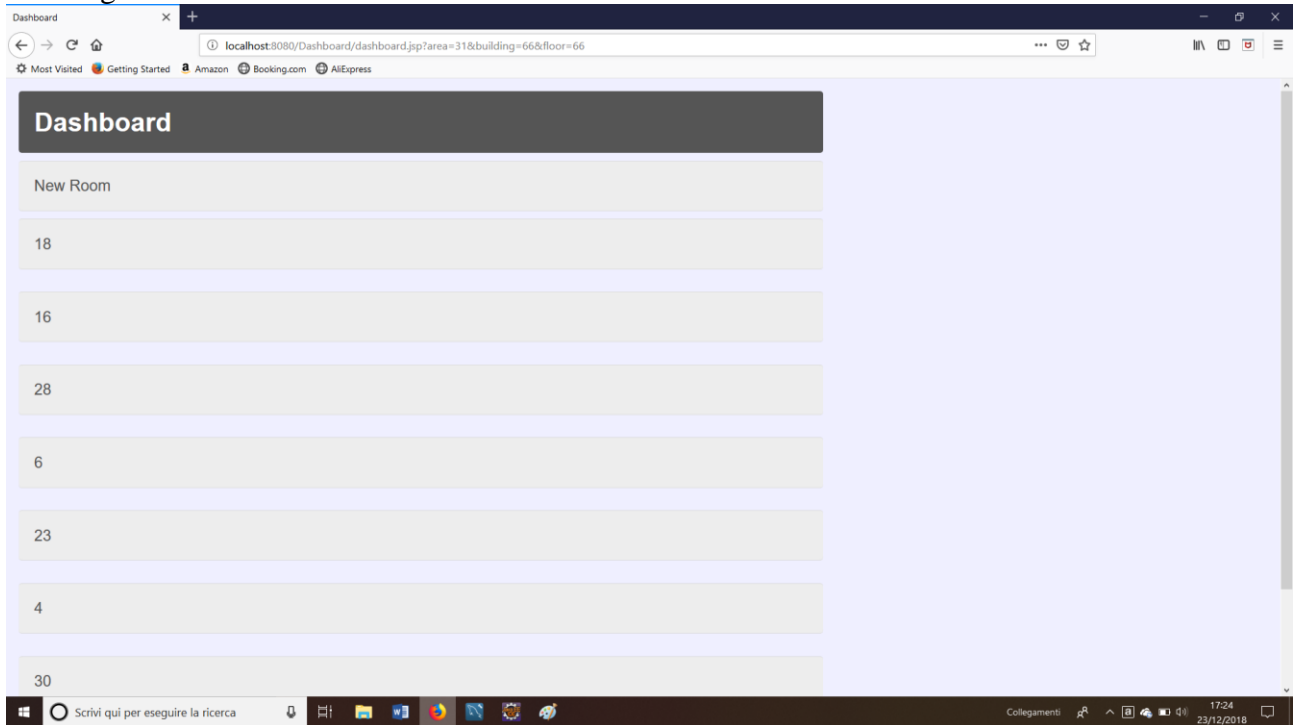Dashboard shows areas

Clicking on one area we can view buildings

Clicking on buildings we can view floors

Clicking on one floor we can view rooms

And finally clicking on one room we can view the sensor with their values that are updated each second