Experimenting with Means to Store and Monitor IoT based Measurement Results for Energy Saving

M. Saari *, J. Grönman*, J. Soini *, P. Rantanen, and T. Mäkinen*
* Tampere University/Faculty of Information Technology and Communication Sciences, Pori, Finland
Mika.saari@tuni.fi

Abstract - **Nowadays, energy consumption and especially energy saving are important issues. The news of global warming has increased the need to save energy in many areas of our living community. Therefore, several research projects on various aspects have been established. This study is part of our ongoing project with the purpose of developing a means of reducing energy consumption in houses and apartments. In particular, the project aims to find ways to increase energy savings without compromising comfortable living. This requires the constant measuring of living conditions, which produces a huge amount of data that has to be monitored continuously and stored for later analysis. In this paper, we report our experiments to select appropriate tools for storing, monitoring, and visualizing data on living conditions. In our test arrangements, the data is gathered by IoT sensors in real locations. Different types of database systems and dashboard software have been used in the trials. Each of these is discussed from the IoT data processing perspective and is presented with examples of gathered data. Finally, we present the toolset that we considered a suitable choice for our context.**

Keywords - Internet of Things; IoT; database; visualization; sensors

I. Introduction

The news of global warming has increased people's awareness of energy consumption and energy saving. These important issues should influence our everyday decisions. The Internet of Things (IoT) has introduced a lot of tools to measure the environment and its state. In the academic world, several research projects from different aspects of energy consumption and energy saving have been established. We are focusing on this issue in our ongoing KIEMI ("Vähemmällä Enemmän – Kohti Kiinteistöjen Energiaminimiä", or "Less is More: Towards Energy Minimum of Properties" in English) project. The aim of the project is to develop proof-of-concept demonstrations and prototype applications that illustrate how cost-effective, open, and modular solutions could be utilized to improve the energy efficiency of existing, older buildings. [1]

The developed prototypes were placed in indoor spaces with the purpose of collecting environmental data, such as temperature and humidity. At the beginning of the project it became clear that the project partners were also interested in knowing the quality of the indoor air, for example the CO_2 and VOC (volatile organic compound) content. This kind of continuous measuring of living conditions produces a huge amount of data. The data must be stored for later analysis. In this research we report on our experiments to select appropriate tools for storing, monitoring, and visualizing the data.

The structure of this paper is as follows: In Section II, we review the related research. In Section III, we introduce three use cases for storing IoT data. In Section IV, we include a discussion and suggestions for future research on the topic and finally, Section VI summarizes the study.

II. Related Research

This section introduces the related studies in this research area.

In Finland, one of the major sources of energy consumption is housing. In fact, the heating of residential buildings accounts for up to 68% of housing energy consumption. Obviously, the main reason for this is our northern geographical location. This background of energy saving in the area of real estate and housing was handled in our earlier study [1]. In addition, the introduced prototype systems [2] were developed during the KIEMI project.

The prototype systems introduced in this study were developed for testing purposes using cost-effective, open, and modular solutions. Therefore, the prototype developing process could be called rapid prototyping. This kind of prototyping and the prototypes developed were handled in more detail in an earlier research paper [1].

In the IoT context dealt with in this paper, the collected stream of data is potentially useful for a large number of applications. The data coming from each connected device can be seen as collections of time series.

A time series can be specified as a sequential set of data points, typically measured over successive time periods. The mathematical definition is a set of vectors $x(t), t = 0,1,2,...$ where t represents the time elapsed. The variable $x(t)$ is treated as a random variable. The measurements taken during an event in a time series are arranged in chronological order. [3] [4]

There are several options for storing time series. Namiot [5] provides a survey of several data persistence options for time series data. He discusses a couple of traditional relational databases as well as many NoSQL solutions. Bader et al. [6] performed a survey and comparison of open source time series databases (TSDB). Their systematic search yielded a total of 83 TSDBs, which were compared using 27 criteria.

Di Martino et al. [7] compare the performance of three different types of database systems in processing time series data: the document database MongoDB, the column family database Cassandra, and the time series database InfluxDB. Musa et al. [8] conducted a performance comparison of InfluxDB and an object-relational database, PostgreSQL. Ramesh et al. [9] present data modeling schemas for Cassandra and MongoDB to store time series data. Rinaldi et al. [4] investigate how the data model used affects the performance of database queries, using InfluxDB in their evaluation.

III. TESTED DATABASES AND VISUALIZATION

This section presents three different use cases: RuuviTAG [10] with a time series database, a data gathering system with a cloud based document database, and an embedded relational database prototype system. The proposed ways to structure data are presented at the end of the section.

A. Time series database

A time series database is commonly used in IoT related systems. This section goes through a time series data table schema, the open source time series database InfluxDB, and finally data visualization with time series databases.

1) Time series databases

Time series data is the order of the values of a variable (e.g., temperature) at regular or set intervals (e.g., hourly). Thus, it is a sequence of discrete time data. For example, time-stamped information such as log files and IoT sensor measurements can be regarded as time series data. The measurements that make up the time series are organized on a timeline that reveals information about the underlying patterns. Organized data is important because there is a relationship between time and measurements and changing the order can change the meaning of the data. An example of a time series would be hourly temperature measurements at a specific weather station etc. [4]

During the rise of IoT technologies, a new trend in data storage has also emerged in the form of time series databases. A time series database (TSDB) is a database used specifically to manage and store values of variables that change over time. Time series data can be process measurements, triggers and alarms, system and application metrics, or revenue flows. They occur in all aspects of life, so the need to monitor, follow, process, and connect with them is enough. However, because the values of the variables are constantly changing, the database must meet specific requirements in order to process such a large amount of data. The write speed must be considerably high in order to track the large amount of data that is sent to the server every second or even milliseconds. In addition, the TSDB should have high capacity and potential scalability, as the amount of data increases rapidly over time. In addition, it provides quick access to real-time analytics, which is mostly used in trending machine learning applications. [4]

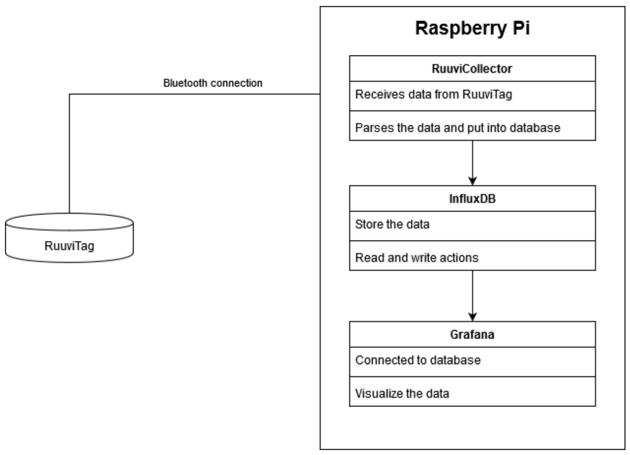

Figure 1. Open-Source sensing solution.

A TSDB can be designed using a standard SQL-based relational database, such as MySQL or PostgreSQL. Recently, NoSQL solutions have had success in designing TSDBs instead of relational databases. NoSQL databases have some advantages over relational databases: they are schema-free, with a simple DB design (no relational model is required); simple horizontal DB scaling in server clusters; easy support for availability. NoSQL databases use different data structures. Such an approach can benefit from the flexibility to represent the data and quick access to the data stored in the database, although this last point depends on the application and the structure of the data. [4]

The most effective TSDB at this time is InfluxDB [11].

2) InfluxDB

InfluxDB is an open source TSDB developed by Influx Data. It is written in the Go programming language developed by Google and is optimized to handle large write and read transactions. InfluxDB is intended for use in any application involving large amounts of time-stamped data. Examples of similar uses include DevOps monitoring, IoT sensor data, and real-time analysis. A time series database is, as its name implies, a database for storing time-stamped data, in which the data is stored and indexed according to the timestamp. In InfluxDB, data is stored in databases that contain measurements. It is possible to define retention policies for these databases. Retention instructions tell the database how long data will be retained and how many copies of that data will be retained in the cluster. The measurements contained in the databases can be compared to the tables in SQL-based databases, because at the conceptual level they are very similar to each other. [12]

Database data is processed using the Influx query language. The syntax of the language is expediently very similar to the SQL query language so that users that have experience with other SQL query languages can easily adopt its use. Database searches are performed using the SELECT statement, and it is possible to use other refinement criteria in addition to it, such as GROUP BY, INTO, or WHERE. The language also includes other useful functions for retrieving information. One database consists of at least two mandatory fields: a timestamp and a field. The timestamp is stored in the time column for each data point in the format RFC3339 and there can be only one for each data point. The field consists of a key value pair,

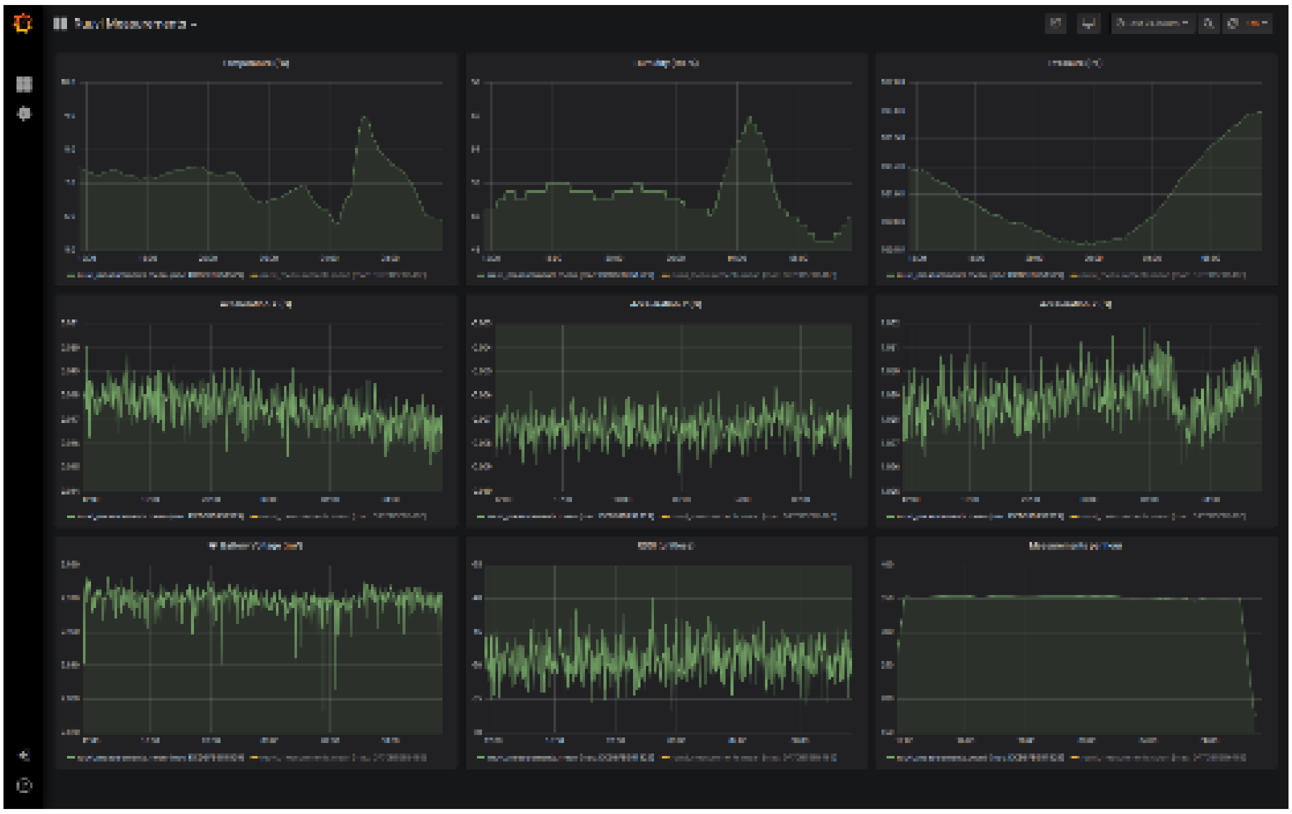

Figure 2. Example of a Grafana dashboard layout with charts.

where the column name acts as a key (Field Key) and the corresponding data points as values (Field Value). The keys are stored as strings, while values can be stored as either strings, truth value variables, or as floats or integers. The database can also contain tags. According to the fields, the tags consist of a key-value pair, where the column name acts as a key (Tag Key) and the corresponding data points as values (Tag Value). All tags are stored as strings. Although tags are not mandatory, their use is desirable mainly because, like fields, the columns are not indexed. As a result, database searches for tags are faster and are ideal for storing frequently retrieved data. All of the above together form a point. The point is thus in practice one line in the same series with the same timestamp. [12]

3) Data visualization

Data visualization is the presentation of data and data in the form of graphical components such as graphs, charts, histograms, gauges, geographic maps, etc. Data visualization helps users understand the meaning, structures, and correlation of data by displaying raw data more comprehensibly.

Grafana is a general-purpose open source visualization tool developed by Grafana Labs. It makes it possible to create different dashboards that allow time-stamped data to be visualized. Grafana runs as a web application and officially supports eight different data sources including Elasticsearch and InfluxDB, among others. Since version 3.0, Grafana has been able to install other data sources as plugins, but they are not officially supported. Dashboards play a key role in Grafana. A single dashboard is in practice an organized collection of different panels that together form a visual representation of the data collected. There can be one or more dashboards and it is possible to import or export them from the system as JSON documents. Panels are individual components that visualize data from a desired source, for example in the form of a graph. Each panel includes a Query Editor with slightly different functionality depending on the data source used. The Query Editor makes it easier to retrieve data from the data source used, and any changes you make to it in the panel Query are immediately reflected. Like dashboards, panels can be imported or exported from the system as JSON documents. [13]

Grönman et al. [14] present a low-cost and flexible solution for environmental sensing. The implementation is based on open-source components. The solution utilizes an ROS (Robot Operating System), RuuviTag sensor, and Raspberry Pi 3 computer to achieve a low-cost solution. By implementing a pilot project, they concretized the potential of their approach to environmental sensing. Potential application areas of this solution include the microclimate control of greenhouses and warehouses; for example, robots can also be used in environments where it is unsafe for humans to enter. Compact robots may also reach places that are inaccessible to humans. The robotics solution can utilize the information transmitted by IoT devices and sensors. RuuviTag is connected via Bluetooth to the Raspberry Pi and the collected data is transferred to the TSDB database. The stored data can be visualized by the visualization software.

In this case, they decided to choose three main software programs to take care of the functionality of the sensing solution. On the software level, the solution has three important programs: RuuviCollector, InfluxDB, and Grafana. First, the RuuviTag is switched to “RAW” mode, which means data comes straight from the sensor and no modification is made on the way. RuuviCollector uses a Raspberry Pi 3 computer with built-in Bluetooth to listen to the data coming from RuuviTag. RuuviCollector needs to parse the data into the correct format and then puts the data in the InfluxDB database. When the sensor data is stored in InfluxDB, then the connection between Grafana and the database needs to be made. Then Grafana simply reads the database and the user can create dashboards to visualize the data. The software and hardware connections are shown in Fig. 1.

Fig. 2. shows the Grafana user interface in the browser window. The user interface is designed to work in a web browser and can also be accessed from public or private networks. Information content can be displayed on the user interface. Fig. 2. also shows the RuuviTag measurements of temperature, humidity, pressure, acceleration x, acceleration y, acceleration z, battery voltage, RSSI, and counts of measurements per hour.

B. *Document database*

In this section, one developed prototype measurement system is presented. A document database test environment was installed and tested in a real environment, a detached house. The focus was to test the data flow and usability from the sensors to a cloud database service. In addition, during the test, a mobile application was developed to inform the user.

The document database was tested in a detached house where the main heating system is a fossil fuel (wood) fired water heater. The heating is managed by the resident and the wood burning is not automated. The house has central heating with water circulation, which is automated. The water circulation power depends on the outside temperature. The measurement system is illustrated in Fig. 3.

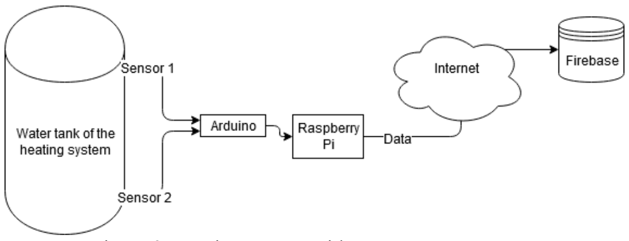

Figure 3. Heating system with measurement system.

The measurement system consists of sensors. The data from the temperature sensors is collected with an Arduino, a single board microcontroller that can handle several types of sensors. The Arduino is connected to the Raspberry Pi, which is small single-board computer (SBC) with a Linux Operating system (OS). The SBC can handle several Arduinos with wired or wireless connections. In the presented case we used a wired connection, but if a wireless connection is used the Arduino is able to work for long periods powered by battery.

The Raspberry Pi is commonly used in rapid prototype test cases due to its ease of use. The Raspberry Pi and the peripherals are off-the-shelf devices, which are low-cost and easily available. In this scenario, the Raspberry Pi was connected to the Internet with a WiFi connection. This connection is needed because the collected data is sent to the Google Firebase Realtime Database. The data includes a timestamp with date and time, and two temperature sensor values.

In the usage test, the measurement system collected 4.3 MB of data during about 10 months. The data was collected once every ten minutes and the database included over 40 thousand datapoints.

The data in the database could be handled in several ways. The Firebase Realtime database provides an interface for reading, modifying, and saving data. In this example case, the data was used with the TempApp mobile application. The TempApp is an Android application and is coded for testing purposes. The main idea was to test the data usage possibilities. The basic functionality of the TempApp application is to retrieve data on a mobile device and show it to the user.

The software framework of the measurement prototype consists of several software components. Fig 4. lists the software used with the devices.

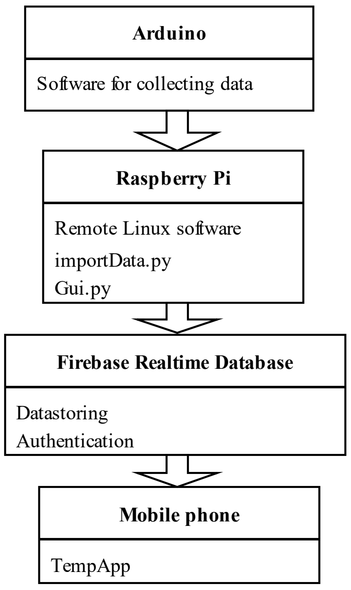

Figure 4. Hardware with software components.

The software in Raspberry Pi has several tasks: importData.py gets data provided by Arduino and sends the data to the database. Gui.py was developed for testing purposes to show that the data collected is acceptable. This was necessary during application development. Raspberry Pi runs with Linux OS and therefore the remote management capabilities provided by Linux were utilized. A remote desktop is one useful tool for the testing phase. Raspberry Pi also stores the data in a file, which can be retrieved easily. Fig 5. shows a bar graph of about one week of data.

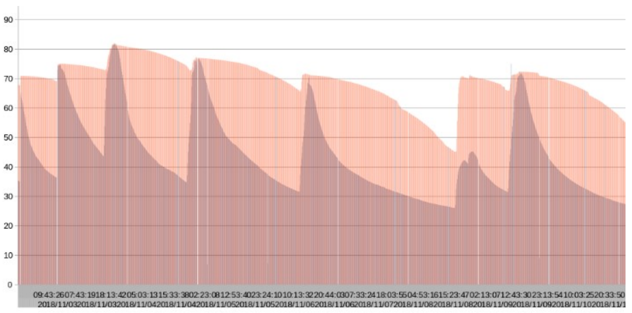
Figure 5. Collected data from measurement prototype.

The database handles data storing and offers the necessary interfaces for using the data. Furthermore, Firebase provides the authentication services that were utilized in the Firebase connections with Raspberry Pi and the TempApp mobile application. The mobile phone runs the TempApp software, the main purpose of which was to provide the user with information.

C. *Relational database*

In one of our experiments [2], we used the embedded relational database, SQLite, for persistent time-series data. The aim of the study was to test cost-effective, off-the-shelf components for measuring indoor air quality. In one of the cases, a sensor system was installed in an apartment building used for eldercare. In this section, we present a simplified schema of the database of the experiment and discuss corresponding schemas for document and time series databases, especially for MongoDB and InfluxDB.

Table 1 shows an excerpt of the data collected by the sensor system. The structure of the data follows the following relational schema - tables and their columns, primary keys (PK), alternate keys (AK), and foreign keys (FK):

sensor_nodes: {node_id^{PK}, description}
sensor_data: {data_id^{PK}, timestamp^{AK}, eco2, humidity, pressure, temperature, node_id^{FK}}

Table 1. Excerpt of data captured by the sensors in the experiment.

ECO_2	Humidity	Pressure	Temp. (°C)	Timestamp
541	57.96	1007.44	23.54	2019-09-12 T12:00:37.000+0000
600	57.88	1007.50	23.55	2019-09-12 T12:03:57.000+0000
646	57.82	1007.82	23.57	2019-09-12 T12:07:17.000+0000

The conceptual representation of the data is shown in Figure 6. There are two entity types, *Sensor Node* and *Sensor Data*. *Sensor Node* has two attributes, *ID* and *Description*, of which the second one is key. The key attribute of *Sensor Data* is *Timestamp*. The other four attributes represent the information collected at the timestamp, *ECO2* (CO_2 level), *Humidity*, *Pressure*, and *Temperature*. In the conceptual representation, the foreign key column of the relational schema has been replaced with the relationship type, *collects*. The primary key column of the *sensor_data* table is a surrogate key and therefore it has been omitted.

A record in a traditional relational database is a row, whose counterpart in *MongoDB* is a *document*. The structure of the document is composed of field and value pairs, being similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

From the conceptual schema in Fig. 6, we can derive different kind of schemas for the MongoDB document database. One of them is similar to the relational schema shown earlier:

sensor_node: {node_id, description}
sensor_data: {timestamp, eco2, humidity, pressure, temperature, node_id}

Like the rows in relational databases, the documents following the structure above are flat: the values of fields are simple – no documents or arrays. The associations between the documents are implemented by storing the key field of the *sensor_node* document in the *sensor_data* document, which corresponds to the principle in relational data structures.

Another way to structure data is to use only one type of document:

```
sensor_node: {
   node_id, description,
   sensor_data: [{
        timestamp, eco2, humidity, pressure, temperature}]
}
```

The document stores information on both the sensor nodes and the data they collect. The *sensor_data* is now a field of the *sensor_node* document. The field is an array of

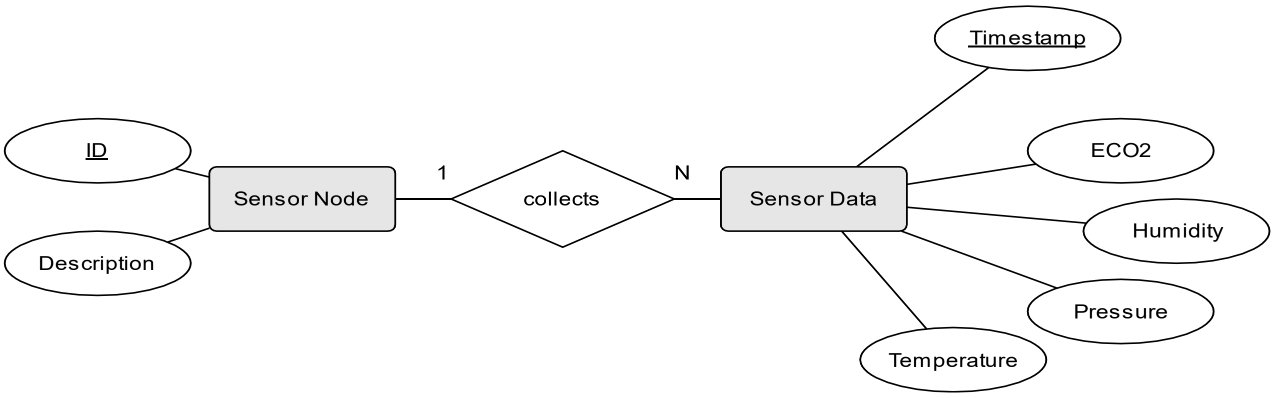

Figure 6. Conceptual schema of the database using prototype system.

documents based on the data collected by a sensor node. Alternatively, the information about the sensor node could be included in the *sensor_data* document as follows:

```
sensor_data: {
   timestamp, eco2, humidity, pressure, temperature,
   sensor_node: {node_id, description}
}
```

The InfluxDB *time series database* stores *measurements*, which are containers of *time*, a *field set,* and an optional *tag set*. *Time* contains timestamps, and the *field sets* and *tag sets* are collections of key-value pairs. The value of a *field* can be of string, float, integer, or Boolean type, while a *tag* value can only be string type. Unlike fields, tags are indexed. [8]

```
sensor_data: {
  time,
  field_set: {eco2, humidity, pressure, temperature},
  tag_set: {sensor_node}
}
```

Converting the relational structure of our experiment for a time series database is relatively straightforward, as shown above. As in the corresponding relational schema, the data structure contains *time*. The data collected by the system forms the *field* set. The data is classified based on the sensor nodes, which represent *tags*.

IV. Discussion and Conclusion

The "best choice" depends on several factors: data size, frequency of data saving, and usage of collected data. Therefore, each use case needs a different kind of data storage system. The previous section presented suitable ways to store data in our use cases.

The study points out several possibilities for future research: How to define a general data structure for storing data? This issue is touched on in the conclusion of use cases. Furthermore, the study raises a question about a technical issue: How to manage databases–cloud based or in one's own managed servers?

The study is part of our ongoing project, the purpose of which is to develop a means of reducing energy consumption in houses and apartments. The presented practical cases focus on the continuous measuring of living conditions, which produces a large amount of data. The data has to be monitored continuously and stored for later analysis.

The study has introduced three different use cases of storing and handling IoT data. Appropriate tools for storing, monitoring, and visualizing the data of living conditions have been presented. In our test arrangements, the data was gathered by IoT sensors in real locations.

Different types of database systems and dashboard software were used in the trials. As the conclusion of our study, we found the *time series database* with the general-purpose visualization tool for a suitable toolset for this kind of context.

Acknowledgment

This work is part of the KIEMI project and has been funded by the European Regional Development Fund and the Regional Council of Satakunta.

References

[1] M. Saari, P. Sillberg, J. Grönman, M. Kuusisto, P. Rantanen, H. Jaakkola and J. Henno, "Reducing energy consumption with IoT prototyping", Acta Polytechnica Hungarica, Volume 16, Issue Number 9, 2019.

[2] P. Rantanen and M. Saari, "Towards the utilization of cost-effective off-the-shelf devices for achieving energy savings in existing buildings", Proceedings of 2020 IEEE 10th International Conference on Intelligent Systems (IS20), Varna, Bulgaria, 26-28 June 2020. submitted.

[3] R. Adhikari and R. K. Agrawal, "An Introductory Study on Time Series Modeling and Forecasting," arXiv Prepr. arXiv1302.6613, vol. 1302.6613, pp. 1–68, Feb. 2013.

[4] S. Rinaldi, F. Bonafini, P. Ferrari, A. Flammini, E. Sisinni, and D. Bianchini, "Impact of Data Model on Performance of Time Series Database for Internet of Things Applications," in 2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), 2019, pp. 1–6.

[5] D. Namiot, "Time series databases," Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL2015), vol. 1536, pp. 132–137, 2015.

[6] A. Bader, O. Kopp, and M. Falkenthal, "Survey and comparison of open source time series databases," Lecture Notes Informatics (LNI), Gesellschaft fur Informatik, vol. 266, pp. 249–268, 2017.

[7] S. Di Martino, L. Fiadone, A. Peron, V. N. Vitale, and A. Riccabone, "Industrial Internet of Things: Persistence for Time Series with NoSQL Databases," Proceedings - 2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2019, pp. 340–345, 2019.

[8] E. Musa, G. Delač, M. Šilić, and K. Vladimir, "Comparison of Relational and Time-Series Databases for Real-Time Massive Datasets," in 2019 42th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2019, pp. 1065–1070.

[9] D. Ramesh, A. Sinha, and S. Singh, "Data modelling for discrete time series data using Cassandra and MongoDB," in 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), 2016, pp. 598–601.

[10] Ruuvitag, retrieved May 12, 2020, from https://ruuvi.com/files/ruuvitag-tech-spec-2019-7.pdf

[11] DB-Engines Ranking of Time Series DBMS, retrieved May 12, 2020, from https://db-engines.com/en/ranking/time+series+dbms

[12] InfluxDB 1.8 documentation, retrieved May 12, 2020, from https://docs.influxdata.com/influxdb/v1.8/

[13] Grafana, retrieved May 12, 2020, from https://grafana.com/grafana/.

[14] J. Grönman, M. Saari, J. Vihervaara and J. Viljanen, "An Open-Source Solution for Mobile Robot Based Environmental Sensing" in 2020 43th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2020, submitted.