A practical framework to generate and manage synthetic sensor data

Zoltán Pödör[1], and Anna Szabó[2]

Abstract—**A huge number of sensors are around us and they generate different kinds of data. Data owners, e.g. the companies need IT environments and applications to handle these datasets. The collected data often contain sensitive information about the operation of the companies and the production processes. Therefore, artificial sensor data are strongly needed in the development and testing phase of these applications.**

In this paper, we introduce a complex application with three main modules to manage synthetic sensor data. The first component is the data generator module, which is capable of creating synthetic sensor data according to the user-defined distributions and parameters. The second module is in charge of storing the generated data in a flexible relational database, developed by us. The third component ensures the filtering and the visualization of the collected or generated data. A common interface was created to bring together the components and to provide a unified interface for the users. The adequate user management was an important aspect of our work. Accordingly, four different user types and authorities were defined.

Index Terms—**synthetic sensor data; data generation; database for sensor data; data visualization**

I. Introduction

Nowadays various types of sensors are available to measure various things around us. They can be applied in our everyday lives to map the attributes of our environment or to measure our health conditions. In addition to this, they can be part of a smart home or smart devices. Apart from these possibilities, sensors can be utilized in industrial environment as well. According to the challenges of IoT and Industry 4.0 an expanding number of the companies and factories apply sensors [1] [2]. These devices measure different kinds of quantitative and qualitative attributes connected to the production, including the actual manufacturing processes and other related attributes (e.g. current consumption) [3]. Collected data can help us to improve the efficiency of the product, to reduce the cost, the amount of waste and to increase the income and the profit. Companies often require special, individual software solutions and applications connected to their data processing and displaying [4].

Security issues - connected to IoT and Industry 4.0 -, such as privacy, access control, information storage and management and the reliability of data management software are the main challenges [5] [6]. Besides the security and privacy questions, the reliability of the developed software is an important problem [7].

Bugs in the code or an incorrectly implemented analysis method can cause different kinds of problems, e.g. a huge loss for the company. Different aspects of software testing are important parameters of developing software that are free from bugs and problems [8]. Test data are the basic elements of the reliable and well-qualified software testing methods [9]. In many cases, the collected data can be sensitive, especially in an industrial environment. For example, the machine and the product data of a given company or the personal data of the workers can be considered sensitive. In many cases, the person or the company who ordered the software cannot disclose the sensitive data to the developers, only in the form of transformed, encrypted data [10].

There are two main ways of creating artificial data in order to handle this problem. First one is data masking [11] when real data are replaced with generated data with a high, measurable level of similarity. The name of these artificial data is semi-synthetic, or hybrid data. Another type of artificial data is the full-synthetic data that is created by an algorithm, and it is usually used for test datasets of production or operational data [12]. In this paper, we focus on the problems of functionality testing.

Synthesis of data is a simple simulation with the primary aim of generating data according to a given model [13]. The elementary simulations can be: (1) sequences that generate various increments; (2) randomizers that create random values using any well-known distributions; (3) mathematical functions that describe the form of generated data; (4) noisers which are generators of lists with missing values, range filters, etc. In this paper we focus on the randomizers with some general distributions. The details and descriptions of different kind of data generator applications are introduced in Section II.B. These and the above mentioned solutions and applications focus only on the data generating method, without the possibility of included, direct and efficient data storing and data processing. The data handler applications target special kind of visualization and analysis possibilities [14], including the pre- and post-processing methods. They usually do not have the ability to create artificial sensor data. They usually use outer databases, which store the actual used data and they focus only on data visualization and data analysis. The uniqueness of our solution is that these three important components (data generator, database and data handler) are connected into one complex application with an appropriate user management according to the stored datasets.

[1] Zoltán Pödör is with Eötvös Loránd University, Faculty of Informatics (e-mail: pz@inf.elte.hu)

[2] Anna Szabó is with Eötvös Loránd University, Faculty of Informatics (e-mail: h6co1g@inf.elte.hu)

DOI: 10.36244/ICJ.2022.2.7

For that reason, on one hand, we offer a Python-based solution to create artificial sensor data according to different conditions (range, measurement frequency, measurement unit, accuracy, etc.) to help the testing period of the software development process. The generated sensor data is stored in the database module that is an integrated part of our application. The data generator was implemented to test the functionality of the developed software with special distributions of the generated synthetic sensor data. On the other hand, our solution is a complex framework with a database to store the generated data and a visualization surface to check, visualize the generated data. Another big advantage is the common surface above the modules that contains all available services of the application, and it grants the services only to the authorized users from the artificial sensor creation through the data generation to storing and visualization. The data visualization and the basic analysis modules allow to process data, from different, outer sources, but stored in the application's database.

Our main aim is to support different kinds of data-based application development and testing phase with artificial sensor data samples. The problem of sensitive data can be handled by the included data generation module during the software development period. Our application can be used in different areas. The integrated database and visualization modules provide the opportunity to handle the data that is either artificially generated or comes from an external source. The module-based structure provides easy expandability in terms of the number of the modules and their structure while the user-friendly interface provides easy handling and use for non-IT professionals because it does not require any special prior knowledge. In the future we plan to include other types of databases and to develop the visualization and analysis modules next to the data generator module.

II. THE APPLICATION COMPONENTS

The application contains three main modules (Fig. 1.). First one is the flexible database to store the generated artificial data with all connected properties, like timestamp, unit, location, devices etc. Another database was created to store separately the data of the different users.

Second one is the data generator module that is responsible for the generation of the artificial sensor data according to the user-defined parameters and distributions.

Third part is the filtering and visualization module that intends to select and visualize the appropriate data. This module gives the opportunity to create some basic statistical properties of the generated data besides the visualization which can help us to compare the artificial data with the real data.

These three modules are embedded into a web-based interface to provide a unified user interface, easy application handling, and user-friendly system for all potential users. The common surface includes all available services of our application but only for the authorized users.

Because of the data security difficulties, user management was an important aspect of our work. Four different types of users were created. The tool admin can handle the sensors (create, delete and modify them), the data generator can use and set the data generator module, the data handler uses the analysis and visualization module while the user admin manages the data (name, e-mail address, roles etc.) of the system's users and registers new members. In the next chapters we will introduce these components in detail.

A. The database

Not only the sensors but also the systems which can handle them (store, analyse the collected data, display the raw and the

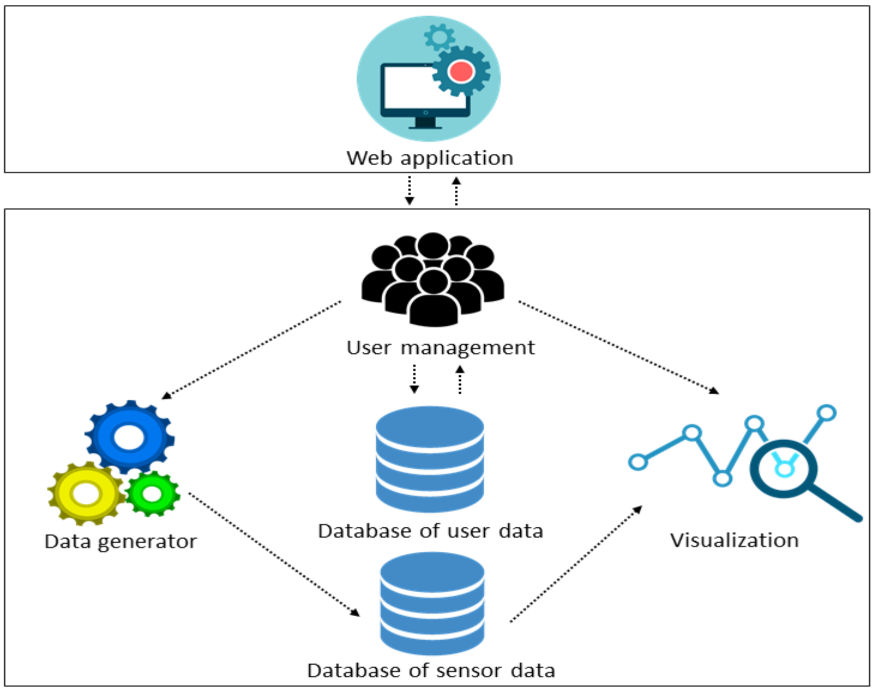


Fig. 1. Structure of the application

processed data) are important. A basic element of such a system is an efficient, reliable and flexible database to store the collected sensor data. Relational and non-relational databases have a lot of advantages and some disadvantages too and both of them offer the opportunity to store the collected sensor data [15]. The structure of a non-relational database is not fixed (unstructured), which means that there are no relation schemas, connections, and the form and the structure of data can be easily modified [16]. The IoT and the sensor based applications often use non-relational databases, because they have a lot of advantages, e.g. the free structure of the database [17]. One of the advantages of the relational databases that they are capable of performing more complex queries and filtering [18] [19]. Our main aim is to create and store artificial sensor data to help the testing of the functionality of the developed software and not to serve big queries. Because of the above-mentioned reasons, we decided to create and use a flexible relational database in MSSQL environment. It is the first module of our complex system, which can store the data derived from the data generator module.

The structure of the database can be seen in Fig. 2 shown as an entity-relationship diagram. The following tables were defined:

- ***Sensor_types*** table: general characterization of the different sensor types defined by a unique identifier (*id*) and each type has a practical name (*name*). *Max_value*s and *min_values* define the possible minimum and maximum values measured by the given sensor type. The *measurement_length* defines the length of one measurement and the *measurement_accuracy* defines the measurement precision of the actual sensor type.
- ***Sensors*** table: it describes the parameters of a given sensor, which belongs to a certain type. Each one of the sensors is defined by a unique identifier (*id*) and has a practical name (*name*). The *alarm_low_boundary* and the *alarm_over_boundary* attributes define the lowest and the highest measured values which might be practically correct. The *frequency* defines the regularity of the measurement. Furthermore, the year of production (*year_prod*) and the date of calibration (*calib_date*) are stored in this table.
- ***Measurement_types*** table: describes the parameters of the measured data. Each piece of data has a unique identifier (*id*) and has a practical name (*name*). The unit of the measured values is stored as well (*unit*).
- ***Measured_values*** table: it contains the measured sensor values, which are defined by two attributes: the *date* and the *sensor.id*. The attribute *measured_value* stores the measured sensor values. The attribute *valid* is a binary parameter with only two values, 1 and 0. Value 1 shows that the measured value is correct, which means that it is between the *sensors.alarm_low_boundary* and *sensors.alarm_over_boundary* values from Sensors table, and the *error_rate* attribute is NULL. Value 0 shows that the measured value is incorrect. In this case, the value of *error_rate* is the value of signed distance between the measured value and the given boundary. The values of attributes *valid* and *error_rate* are created automatically by a trigger.
- ***Devices*** table: describes the parameters of the devices which contain the sensors. Each device is defined by a unique identifier (*id*) and has a practical name (*name*).

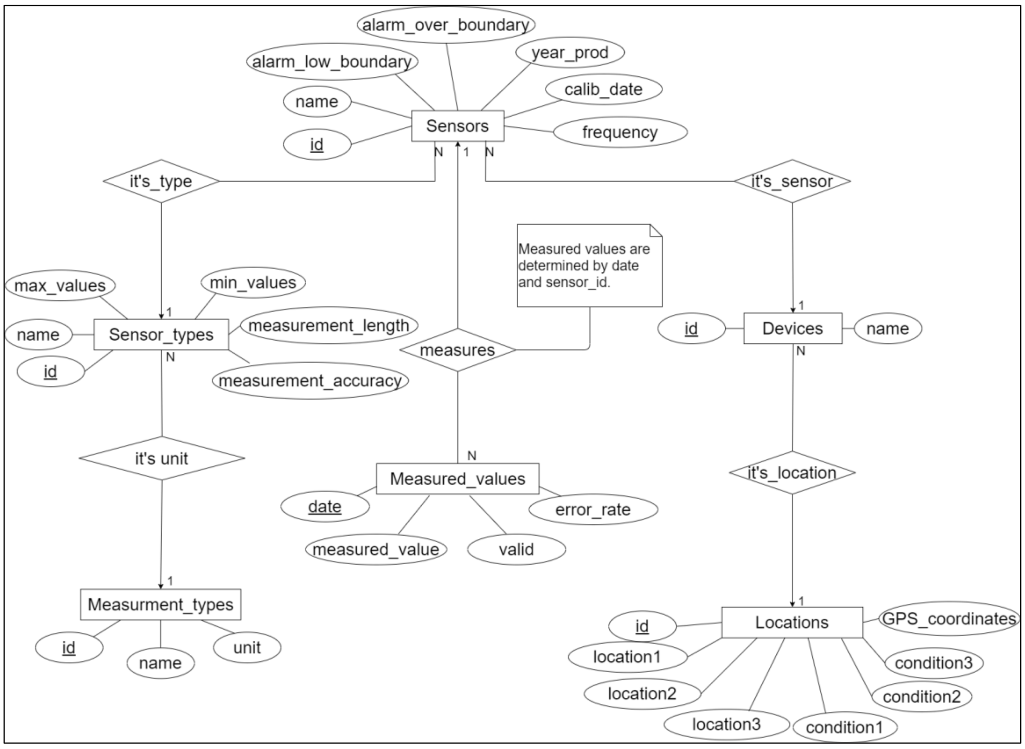


Fig. 2. Entity-relationship diagram of the database structure

- ***Locations*** table: describes the parameters of the location of the devices and sensors. Each place and location is defined by a unique identifier (*id*). Attributes *location1*, *location2* and *location3* store the details of the location, e.g. the name of the location or company. The attribute *GPS_coordinates* contains the accurate location in GPS format. *Condition1, condition2* and *condition3* provide an opportunity to store information about the conditions of the location, e.g. if the sensor or the device is an indoor or outdoor gadget.

The next connections were defined between the tables:

- Each of the sensors has one and only one sensor type. In other words, there are sensor families, with some well-defined general parameters in which we can define a concrete sensor, which belongs to the given sensor family. This connection is a one-to-many relationship, because one sensor family can contain more sensors and one sensor always belongs to only one family.
- Each of the sensor types includes the unit and the type of the measurement. It is a one-to-many relationship, because a given sensor type measures only one thing, but one type of measurement can measure more than one sensor type.
- The measured values are connected to the sensor which measured them. This is a one-to-many connection, because a measured value always belongs to only one given sensor, which measured it, but one sensor can measure more than one data.
- All sensors belong to a device, which contains the given sensor. This connection is a one-to-many relationship, because a device can contain a lot of sensors, but one sensor is always in one device.
- It is important to store the conditions, e.g. the place of the sensor. It is a one-to-many connection, because a given device is always in one place, but more than one device can be in one place.

Our database is suited for storing any measured sensor data or storing the data, which are generated by the application's second module, the data generator. Of course, in the future, if it is necessary or practical, we can replace our relational database with another type of database.

B. The data generator

This module supports the development and testing phase of the applications, which use sensitive data, because it generates artificial data according to the user-defined parameters. These artificial data are similar to the original but sensitive data. There are some solutions for generating synthetic or artificial data, but they usually have other, special goals with their solutions. For example, it is possible that they are made for a special task. Zimmering et al. [20] created a novel method for a generation process of data to compare the selected machine learning methods. Tam et al. [21] have developed an automatic process for creating input files (connected to building sensors) to a fire model simulation. Norgaard et al. [22] proposed a supervised generative adversarial network architecture to create synthetic sensor data connected to health monitoring.

In some synthetic generation, tools need a sample of the real-data as an input. They learn from the original dataset, and they generate the new clone datasets based on this information. CTGAN [23] is an open-source project from MIT which is a collection of Deep Learning-based Synthetic Data Generators. Synsys [24] is a system written in Python and it uses Hidden Markov Models to generate sensor event sequences and it accepts an existing dataset as input and generates a similar synthetic dataset.

There are some solutions which do not need a sample dataset but need a schema to describe the real-data. We have taken to examples to show it. Log-synth [25] generates data in accordance with a given schema. It contains only two distributions which are the normal and random walk. The user can define starting time, frequency and the parameters of the given distribution in the schema file. Iosynth [26] is similar to log-synth, but it allows for more flexibility in terms of defining the type of data to be generated (in a schema file). Furthermore, it has more implementations of distributions to choose from. We can choose either fixed interval sampling, normal or exponential distributions. Both solutions can create JSON files as an output.

The above-mentioned data generators are simple solutions, which implies that they do not have either an included database, user management or a web-based surface to handle the whole application. They do not give the opportunity to create artificial sensors and devices connected to the data synthetization.

Our main aim is to generate artificial sensor data to support the testing phase of software and their functionalities which were developed to handle the original sensor data. It was an important aspect of creating this application not to use real data in the generation method, only their, like Iosynth and log-synth.

The main part of the generator module was implemented in Python language. A public Github project, which was developed by a Korean developer team, was integrated into our module to help us to create data. It is called Mandrova which means "make it" in English, but it means "make sensor data" in the context of sensors [27]. Developers can generate values with many kinds of distributions with the help of this project, but only three of them were used:

- normal distribution with the mean (μ) and the standard deviation (σ) parameters:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

- exponential distribution with lambda parameter:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, where\ x \geq 0 \\ 0,\ where\ x < 0 \end{cases} \tag{2}$$

- gamma distribution with α and β parameters:

$$f(x) = \begin{cases} \frac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}, where\ x > 0 \\ 0, \qquad where\ x \leq 0 \end{cases} \tag{3}$$

These three types were chosen in our current solution, because they are among the most general distributions [28]. The surveyed data generators, which use the distribution of the data,use less or the same distributions as our application. Log-synth [25] contains only normal distribution and random walk. Using Iosynth [26] we can choose normal or exponential distributions and fixed interval sampling. We are planning to build in other distributions in the future to extend the possibilities of this module.

The generation of the values is based on four basic parameters. At first, an existing sensor must be chosen for which the values will be generated. All sensors (which are stored in the database, except for those that have not been connected to a sensor type) are listed and can be selected. The second criterion is the type of statistical distribution with appropriate parameters. The user also has to decide how much data should be generated. Finally, a timestamp has to be given, which is the date of the latest created data. Before the actual data generation process, some important information is automatically queried from the database. The table ***sensor_types*** guarantees the following attributes of the chosen sensor: *measurement_accuracy*, *max_values* and *min_values*. *Measurement_accuracy* defines the accuracy of the stored data. If this record is missing, the rounding value is one by default (as one decimal). When the generated value exceeds the *max_values* or it is less than the *min_values* parameters, the generated data's new error indicator values are 9999 or -9999 (it depends on whether the value is over the maximum or under the minimum). This notation helps us to determine and handle "measurements" which are certainly wrong. If both *min_values* and *max_values* fields are empty in the database, all incoming data are accepted. If only one of them is missing, all the generated values are correct which are under or over the existing limit.

In accordance with the above-mentioned conditions, the generator module's algorithm has three main steps. The first one is to generate the artificial data based on the user-defined distribution and connected parameters. The second one is the checking method, to decide whether the generated value is valid or not according to the parameters *min_values* and *max_values*. If it is necessary, the algorithm changes the generated values to the error indicator values. The last step is the data rounding on the basis of the parameter *measurement_accuracy*. After that, the module stores the new records in the database.

C. Filtering and visualization

Data filtering and visualization are the most important basic tasks of data handling and they are essential for further data analysis [28]. The third module's visualization part is a JavaScript-based component and it gives the opportunity to handle and visualize the generated datasets and to analyse them later in Python language following further development. Currently, the module has two main functions: filtering data by different aspects, like timestamp, sensors, places etc. and visualizing and creating basic statistical properties of datasets. This second function allows a basic comparison between the generated and the original, real data.

A user-friendly interface was created to filter and visualize the stored data. It was an important aim not to develop this module only for IT specialists. Therefore, a huge number of automatized solutions were built in to help users.

Users can filter by devices, sensor types or sensors and enter a starting and an ending date. The device selection is optional, but if a device has been chosen, only those sensor types and sensors are available which are connected to the selected device. Otherwise, all stored sensor types are available to users. Sensor type selection is required because our goal was to visualize only those sensors which have the same type and for example the same unit. Obviously, the user must choose at least one sensor. The last two parameters, the timestamps are optional. After setting these parameters, the application queries all records from the database according to the selected parameters. Our aim was to visualize more than one sensor from one sensor type, but we had to handle the problem of different timestamps of different sensors. To solve it, an algorithm was created and implemented. For visualization, we had to use a two-dimensional table, which stores the values that the user wants to display. The first column stores timestamps after which there is one column for each sensor. This table is loaded up with data by the above-mentioned algorithm whose main task is to check if the selected sensors made measurements at a given time or not. The algorithm uses all the timestamps when any of the selected sensors (in the given time period) made a measurement. Timestamps are stored twice, in two arrays. In the first one, we store all the available dates in chronological order. All of them are stored in the second one too but they are grouped by sensors. The algorithm iterates over the first array and compares the elements to the other array's timestamps. If they are equal, it means that a measurement was made at this time by the current sensor, and we store this measured value in the given sensor's column. Otherwise, we insert a null value into the appropriate sensor in the given timestamp. Such a table can be seen below (Table I.). The timestamps are in the first column, and it can be seen in the others that the three sensors (named S1, S2, S3) did not make measurements at the same time, so there are null values in these fields.

Table I
An example for the uploaded two-dimensional table

date	S1	S2	S3
2021-11-02 14:00:00	null	17.03	null
2021-11-02 14:05:00	20.17	null	10.58
2021-11-02 14:10:00	null	23.11	null
2021-11-02 14:15:00	22.05	null	9.87

For displaying the selected data, Google Charts was used which is an API that helps developers to display data on diagrams simply. The easiest way to use it is to load some libraries and embed a JavaScript code in the application [30].

The application has a user-friendly web-based interface to provide a uniform display for the using of all three main modules and their functionalities. The available application menus depend on the actual user's authorities. This solution ensures that users can only use those functions which are available to them. To guarantee this, user management is an important part of the application.

III. USER ADMINISTRATION

Data security and the adequate user authorities are basic requirements for such an application [31]. According to this, our software can be used only by registered users. The attributes of the users are stored in an independent database. It consists of two tables. One is for the users and the other is for the different roles. A user can have more than one role (Fig. 3).

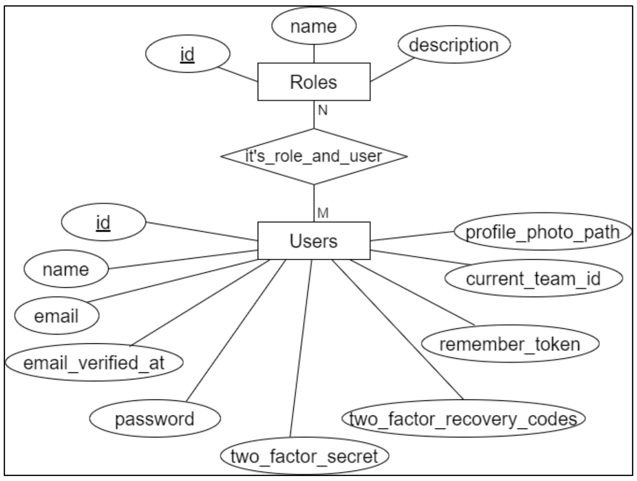


Fig. 3. Users and roles

Four different roles are defined. These roles are separated from each other, and they are related to well-defined tasks. First is ***tool admin*** who manages the data of measurement types, sensors, sensor types, devices and locations. It is the tool admin's responsibility to modify the attributes of a given element, e.g. the alarm boundaries of a sensor, the year of the production, the date of the calibration and the measurement frequency.

Due to the relationship between the database tables, there may be anomalies after a modification or creation, and we have to handle them. A special menu was created for this. The application can identify and inform the tool admin about the next anomalies:

- There is no measurement type connected to a sensor type,
- A sensor is not connected to a sensor type,
- A sensor is not connected to a device,
- A device does not have a location.

The application can identify these anomalies, and it creates a list about the problems for users to correct them. The interface of this menu is shown in Fig. 4.

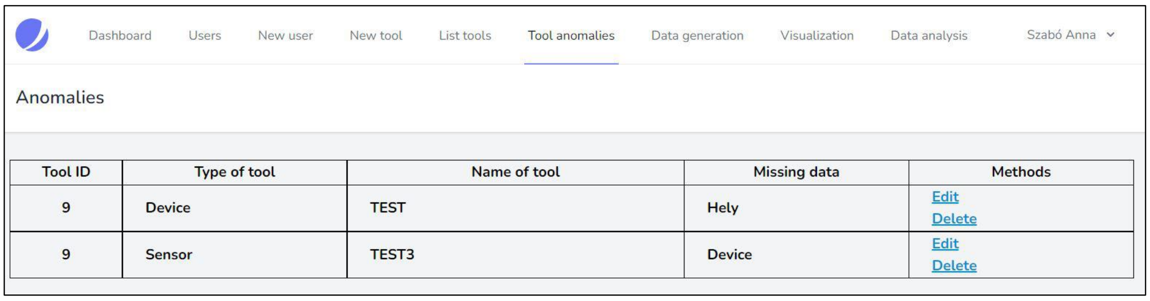
The second type of user is ***data generator***, who can generate the artificial sensor data by specifying various parameters. The details are in Section II B.

Data handler manages the visualization interface. This role gives the opportunity to give some filtering conditions. Afterwards, the diagram, which is created by the program based on the given conditions, can be viewed by the user.

The last user is the ***user admin***, who manages the data of the users and registers new users. User admin can handle the personal data of the users, like name, e-mail address and it is his/her responsibility to set the authorities of the users. The current surface of the application corresponds to the logged user's authorities. It means that a given user can reach only those functions that come under his or her authority. For example, a data generator user can only use the functions connected to artificial data generation, but not other functions, like visualization, sensor and user management. Of course, a user can be assigned multiple roles who can reach and use all related functions in the application.

IV. A DEMONSTRATION USE-CASE

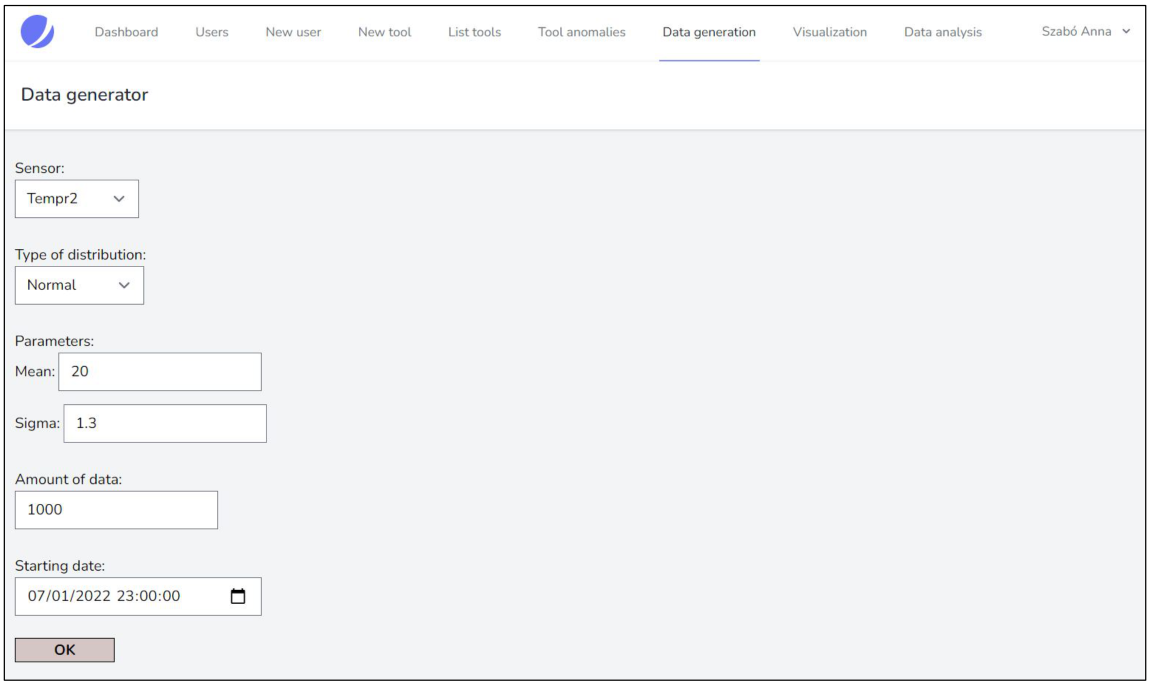
In this section, we would like to show the functionality of our application from the data generation to data filtering and visualization. To do this, different kinds of users were defined with the appropriate roles and authorities. At first, we generated some artificial values with our data generator module. Before generating, some test elements (sensors, sensor types etc.) were inserted into our database. In this test, one sensor, named Tempr2 was selected, which is a thermometer, and the other parameters were entered. We chose the normal distribution and 1000 values were generated where the starting date was 2022. 01. 07 23:00:00. (Fig. 5). At the end of the process, a message tells us if the generation was successful or not. We manually checked the generated data in the database.



Anomalies

Tool ID	Type of tool	Name of tool	Missing data	Methods
9	Device	TEST	Hely	Edit Delete
9	Sensor	TEST3	Device	Edit Delete

Fig. 4. Anomalies



Data generator

Sensor: Tempr2

Type of distribution: Normal

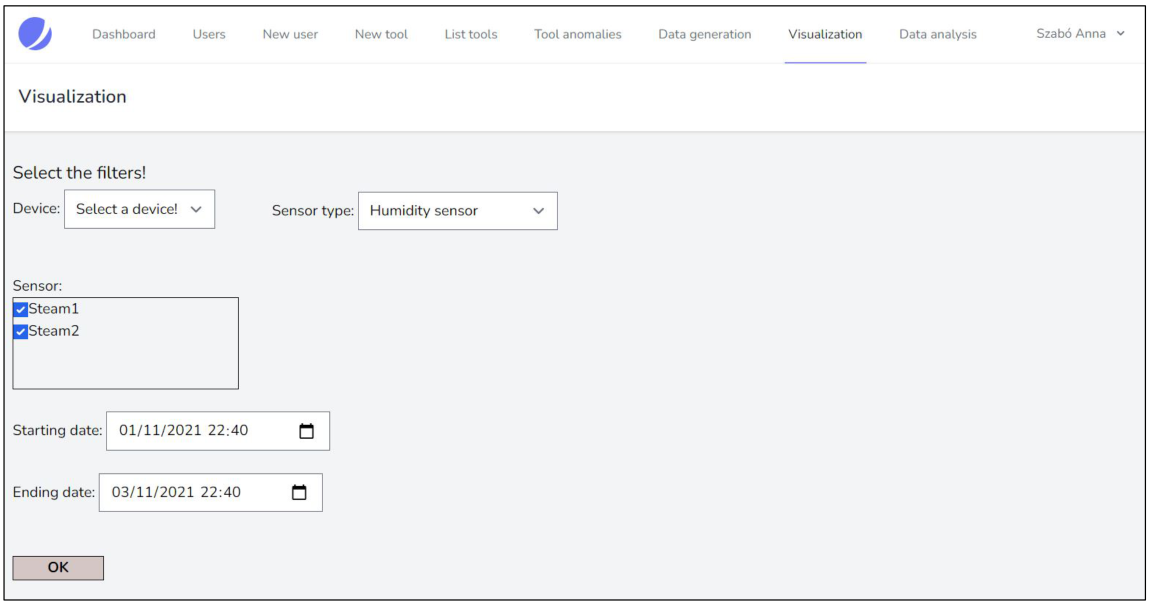
Parameters:
Mean: 20
Sigma: 1.3

Amount of data: 1000

Starting date: 07/01/2022 23:00:00

Fig. 5. Interface of a test data generation

After a huge number of values had been generated, we started to test the visualization component. As it can be seen in Fig. 6, humidity sensor type was selected and two sensors appeared in the box which means that there are two sensors in the database connected to the chosen sensor type. We also entered the starting and the ending date, and then the line diagram was created by the program as shown in Fig. 7. Besides, we tested the filter under the diagram that also worked properly as we could easily change the interval of the dates. In addition, some random values were selected from the diagram and we checked in the database if they matched, and we experienced that all of them were the same at the given times.



Visualization

Select the filters!

Device: Select a device!   Sensor type: Humidity sensor

Sensor:
✓ Steam1
✓ Steam2

Starting date: 01/11/2021 22:40

Ending date: 03/11/2021 22:40

Fig. 6. Filtering before visualization

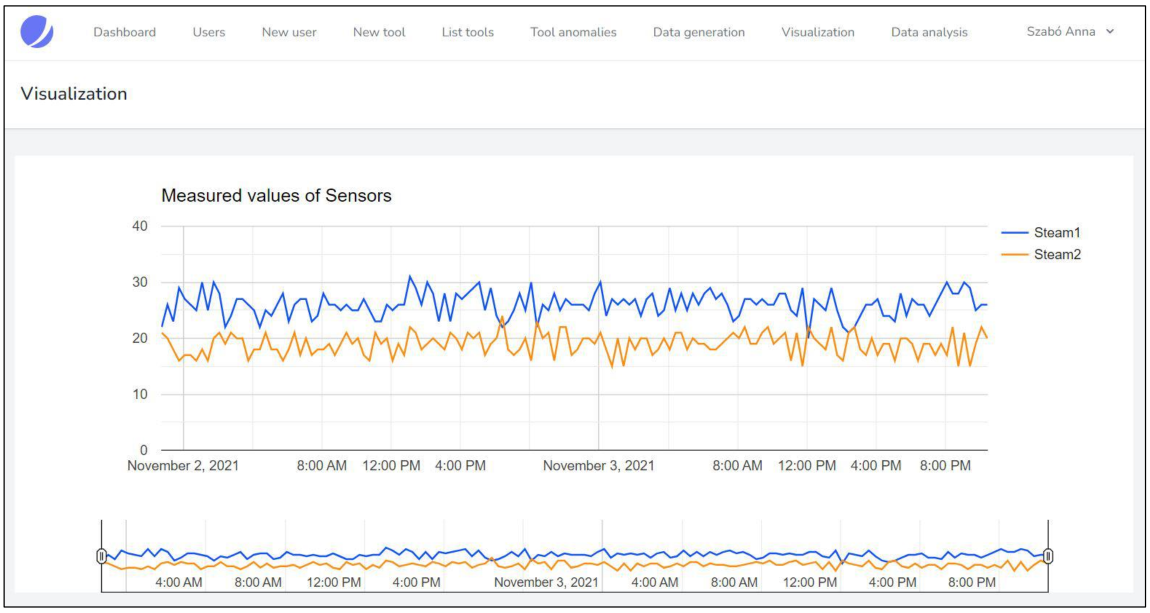


Fig. 7. Line diagram of measured values

V. CONCLUSION

In this paper we created and implemented an artificial sensor data generator as part of a complex sensor data handling application. The application includes a flexible database structure (designed by us) to store the data, the above-mentioned data generator module and a visualization module to filter and visualize the selected data. The main idea behind the generator module is that there are a huge number of applications which are developed to handle sensitive, industrial sensor data. This module is able to create artificial sensor data sample to support the development and testing phase of the software and their functionalities. A common web-interface guarantees access to the whole application.

User management was an important part of the software. Accordingly, the four defined roles and the connected authorities define the available application interface and functions for a given user. The connected data is stored in a separated database.

The examined similar data generators are simple solutions since they do not have included database and user-friendly surface. The parametrization and the use of them is possible only in command line or in special environment. User administration is the other specialty of our solution that defines the access to the artificial devices, to the sensor, and to the generated data.

The architecture and the module-based design of the application allows for later expandability and further steps of development. There are three possible distributions in our current generator module, but we can expand this circle with other important distributions to broaden the range of possibilities of this module. The visualization module currently contains simple analytic functions to create some basic statistical parameters and we are planning the expansion in this direction. The range of the user roles can be expanded too in the future. The current version of our framework – thanks to the well based database structure, user administration and module structure - is a good basis for the future making it possible for us to extend the sphere of the modules and their functionalities.

ACKNOWLEDGMENT

The research was supported by the project No. 2019-1.3.1-KK-2019-00011 financed by the National Research, Development and Innovation Fund of Hungary under the Establishment of Competence Centers, Development of Research Infrastructure Programme funding scheme.

REFERENCES

[1] G. Dalmarco, F. R. Ramalho, A. C. Barros, and A. L. Soares, “Providing industry 4.0 technologies: The case of a production technology cluster,” *The Journal of High Technology Management Research*, vol. 30, no. 2, 2019, DOI: 10.1016/j.hitech.2019.100355.

[2] A. Koncz, and A. Gludovatz, “Calculation of indirect electricity consumption in product manufacturing,” *International Journal of Energy Production and Management*, vol. 6, no. 3, pp. 229–244, 2021, DOI: 10.2495/EQ-V6-N3-229-244.

[3] S. A. Hashmi, C. F. Ali, and S. Zafar, “Internet of things and cloud computing-based energy management system for demand side management in Smart Grid,” *International Journal of Energy Research*, vol. 45, no. 1, pp. 1007–1022, 2020, DOI: 10.1002/er.6141.

[4] J.-Q. Li, F. R. Yu, G. Deng, C. Luo, Z. Ming, and Q. Yan, “Industrial Internet: A Survey on the Enabling Technologies, Applications, and Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1504-1526, 2017, DOI: 10.1109/COMST.2017.2691349.

[5] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, “Security of the internet of things: Perspectives and challenges,” *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, 2014, DOI: 10.1007/s11276-014-0761-7

[6] K. R. Sollins, “Iot Big Data Security and Privacy Versus Innovation,” *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1628–1635, 2019, DOI: 10.1109/JIOT.2019.2898113

[7] A. Garg and A. Arora, "Software Reliability–A Review", *International Journal of scientific research and management*, vol. 4, no. 7, 2016.

[8] H. Tahbildar and B. Kalita, "Automated Software Test Data Generation: Direction of Research," *International Journal of Computer Science &amp; Engineering Survey*, vol. 2, no. 1, pp. 99–120, 2011, DOI: 10.5121/ijcses.2011.2108

[9] M. Esnaashari and A. H. Damia, "Automation of software test data generation using genetic algorithm and reinforcement learning," *Expert Systems with Applications*, vol. 183, 2021, DOI: 10.1016/j.eswa.2021.115446.

[10] M. A. Calles, "Protecting Sensitive Data," in *Serverless Security,* Berkeley, CA, USA: Apress, 2020, pp. 257-283, DOI: 10.1007/978-1-4842-6100-2_10.

[11] A Net 2000 Ltd., "Data Masking: What You Need to Know" 2016.

[12] N. Laskowski, "What is synthetic data? - definition from whatis.com," SearchCIO, 12-Feb-2018. [Online]. Available: https://www.techtarget.com/searchcio/definition/synthetic-data. [Accessed: 09-Mar-2022].

[13] S. Popic, B. Pavkovic, I. Velikic, and N. Teslic, "Data Generators: A short survey of techniques and use cases with focus on testing," *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pp. 189–194, 2019, DOI: 10.1109/ICCE-Berlin47944.2019.8966202.

[14] O. Embarrak, Data Analysis and Visualization Using Python, Apress Berkeley, CA, 2018 DOI: 10.1007/978-1-4842-4109-7

[15] K. Fraczek, M. Plechawska-Wojcik, "Comparative Analysis of Relational and Non-relational Databases in the Context of Performance in Web Applications," in BDAS 2017 DOI: 10.1007/978-3-319-58274-0_13

[16] B. Anderson and B. Nicholson, "SQL vs. NoSQL databases: What's the difference?," *IBM*, 15-Jun-2021. [Online]. Available: https://www.ibm.com/cloud/blog/sql-vs-nosql. [Accessed: 19- Jan-2022].

[17] S. Kontogiannis, C. Asiminidis and G. Kokkonis, "Comparing Relational and NoSQL Databases for carrying IoT data", *Journal of Scientific and Engineering Research*, 2019.

[18] A. Gopani, A. Choudhary, S. Bhattacharyya, and S. Goled, "10 most used databases by developers in *2020," Analytics India Magazine*, 12-Jan-2022. [Online]. Available: https://analyticsindiamag.com/10-most-used-databases-by-developers-in-2020/. [Accessed: 19-Jan-2022].

[19] "Engines ranking," DB. [Online]. Available: https://db-engines.com/en/ranking [Accessed: 19-Jan-2022].

[20] B. Zimmering, O. Niggemann, C. Hasterok, E. Pfannstiel, D. Ramming, and J. Pfrommer, "Generating artificial sensor data for the comparison of unsupervised machine learning methods," Sensors, vol. 21, no. 7, 2021, DOI: 10.3390/s21072397.

[21] W. C. Tam, E. Y. Fu, R. Peacock, P. Reneke, J. Wang, J. Li, and T. Cleary, "Generating synthetic sensor data to facilitate machine learning paradigm for prediction of Building Fire Hazard," *Fire Technology,* 2020, DOI: 10.1007/s10694-020-01022-9.

[22] S. Norgaard, R. Saeedi, K. Sasani, and A. H. Gebremedhin, "Synthetic Sensor Data Generation for Health Applications: A Supervised Deep Learning Approach," in Annual International Conference of the EMBC, Honolulu, HI, USA, 2018, pp. 1164-1167, DOI: 10.1109/EMBC.2018.8512470.

[23] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. "Modeling tabular data using Conditional GAN". 2019, DOI: 10.48550/arXiv.1907.00503.

[24] J. Dahmen and D. Cook, "SynSys: A Synthetic Data Generation System for Healthcare Applications," *Sensors*, vol. 19, no. 5, 2019.

[25] Tdunning, "TDUNNING/log-synth: Generates more or less realistic log data for testing simple aggregation queries.," *GitHub*. [Online]. Available: https://github.com/tdunning/log-synth. [Accessed: 09-Mar-2022].

[26] Rradev, "Rradev/iosynth: Iosynth is IOT device/sensor simulator and synthetic data generator.," *GitHub*. [Online]. Available: https://github.com/rradev/iosynth. [Accessed: 09-Mar-2022].

[27] Makinarocks, "Makinarocks/Mandrova: An Awesome Synthetic Sensor Data Generator for Python3," *GitHub*. [Online]. Available: https://github.com/makinarocks/Mandrova. [Accessed: 02-Mar-2022].

[28] C. Forbes, M. Evans, N. Hastings, and B. Peacock, *Statistical Distributions*, 4th Edition, Hoboken, New Jersey, USA: John Wiley & Sons, Inc., 2011

[29] S. K. Peddoju, and H. Upadhyay, "Evaluation of IoT data visualization tools and techniques," in *Data Visualization*, Singapore: Springer, 2020. pp. 115-139. DOI: 10.1007/978-981-15-2282-6_7

[30] "Using google charts | google developers," Google. [Online]. Available: https://developers.google.com/chart/interactive/docs. [Accessed: 02-Mar-2022].

[31] K. Tabassum, A. Ibrahim, and S. A. El Rahman, "Security Issues and Challenges in IoT," in ICCIS, Sakaka, Saudi Arabia, 2019, pp. 1-5, DOI: 10.1109/ICCISci.2019.8716460.

Zoltán Pödör received the M.Sc. degree in Mathematics and Computer Science from the University of Szeged in 1999. He wrote his PhD thesis on the extension opportunities of time series analysis based on special method at University of West Hungary in 2014. Between 2006 and 2020 he worked at the University of Sopron, Hungary in various positions including Head of Institute of Informatics and Economics. Since 2020 he has been associate professor at Eötvös Loránd University, faculty of Informatics. His current research interests cover time series analysis, data mining techniques in practice and handling and processing of sensor data. He has more than 70 publications.

Anna Szabó has been a BSc student at Eötvös Loránd University, Szombathely, Hungary in Computer Science since 2019. Her current interests are storage, handling, visualization and analysis of different kinds of data (mainly sensor data).