Nigerian Research Journal of Engineering and Environmental Sciences 7(1) 2022 pp. 27-39
p ISSN: 2635-3342; e ISSN: 2635-3350

Nigerian Research Journal of Engineering and Environmental Sciences
Journal homepage: www.rjees.com

Original Research Article

Development of an Interactive User Interface for the Diagnosis and Monitoring of a Hydrogen Generator

***[1]Idris, M.M., [1]Mohammed, A., [2]Zango, M.S., [2]Dandago, K.K., [1]Hassan, A. and [1]Yahaya, J.U.**

[1]Mechatronic Engineering Department, Faculty of Air Engineering, Air Force Institute of Technology, PMB 1154, Kaduna, Nigeria.
[2]Aerospace Engineering Department, Air Force Institute of Technology. PMB 1154, Kaduna, Nigeria.
*mi.muhammad@yahoo.com; mi.muhammad@afit.edu.ng

http://doi.org/10.5281/zenodo.6720607

ARTICLE INFORMATION

Article history:
Received 21 Jan, 2022
Revised 09 Apr, 2022
Accepted 14 Apr, 2022
Available online 30 Jun, 2022

Keywords:
Aviation industry
Hydrogen generator
Monitoring hydrogen generators
Microservices software architecture
Interactive user interface

ABSTRACT

As the aviation industry is making transition towards cleaner propulsion, hydrogen fuel cells have high prospects. On-board generation of hydrogen is regarded more feasible due to safety issues and other restrictions of hydrogen storage systems. The hydrogen generator is a vital part of the propulsion system; therefore, its conditions need to be monitored constantly to avoid failure. This paper focuses on developing software that provides an interactive user interface (IUI) to monitor the condition of the hydrogen generator and control the output pressure of the gas from it. The IUI software was designed and developed based on a microservices architecture which split its functionality into three independent modules (microservices, mediator, and IUI) to make it more reliable, robust, and scalable. The microservices module collects diagnostic data from all subsystems of the generator while the mediator allows the exchange of data based on the principle of global variables between the microservices components. The IUI software was developed using a widget library.js which is written in JavaScript and allows remote user communication. The IUI software was found to effectively track the working condition of the generator, set specific (required) gas pressure at the output of the generator, and turn on/off the generator.

1. INTRODUCTION

Aviation appears to be one of the fastest-growing industries and a study indicated that 12% of vehicular greenhouse gas emissions are contributed by aircraft (Bicer and Dincer, 2017). Fossil-based aircraft fuels are the major pollutants that lead to these detrimental emissions of greenhouse gases and it is required to

limit dependency on them (Khandelwal et al., 2013; Seyam et al., 2021)). Furthermore, the energy efficiency and depletion of these resources are other factors that necessitate the search for viable alternative in the aviation industry (Akdeniz and Balli, 2021). The world is currently transitioning from fossil-based propulsion towards cleaner electric propulsion resulting in the conversion of the existing pneumatic and hydraulic systems into electric (Elitzur et al., 2017).

The power plants (reciprocating engines and gas turbines) for small-scale unmanned aerial vehicles (UAV) usually have low efficiency and therefore use batteries as their main source of power. However, the energy density of batteries is too low to power such small-scale vehicles because of the increasing need for endurance (Kim and Kwon, 2012). Some researchers regard fuel cells as a new power source and an alternative to existing batteries (Kim and Kim, 2014). Fuel cells are eco-friendly, reliable, and usually employ hydrogen as the fuel. Hydrogen (free from carbon and other impurities) is a suitable alternative fuel since it is in abundance, lightweight, and can easily be synthesized through gasification and other methods (Elitzur et al., 2017; Akdeniz and Balli, 2021) The fact that hydrogen is lightweight makes it suitable for long-range operations and helps maintain the maximum take-off weight at the desired level, apart from being a clean fuel that guarantees a reduction in greenhouse gas emissions (Baharozu et al., 2017). Additionally, studies showed that approximately 11% of the energy consumed when traditional fuels are used during journeys is reduced when hydrogen is utilized (Verstraete, 2013).

Even though hydrogen possesses suitable attributes to serve as aircraft fuel, its main drawback is the meticulous design and implementation that are required for its adoption (Godula-Jopek and Westenberger, 2016). The shape, size, and thermal insulation of the tank required to store the hydrogen are the main concerns in its utilization (Winnefeld et al., 2018). Hydrogen has a very low density in both gaseous and liquid phases; therefore, it requires a special storage system. Typically, hydrogen must be stored under excessive pressure (350-700 bar) in a gaseous state and at cryogenic temperatures in a liquid state (Elitzur et al., 2017). A cylindrical and comparatively heavier tank than the one required for conventional fuels is required for hydrogen storage (Westenberger, 2003; Bicer and Dincer, 2017). Consequently, operating empty mass of aircraft is increased. Although thermal insulation of a hydrogen tank is required to prevent boil-off losses and overheating, an isolated hydrogen tank leads to an increase in drag which subsequently results in to rise in energy consumption by an average of 11.5% (Westenberger, 2003).

To prevent the problems associated with hydrogen tanks, hydrogen can be generated onboard using generators that operate based on different methods such as photocatalytic water splitting and water electrolysis (Petrescu et al., 2020). In this case, water would be used as the compound for Liquid Hydrogen (LH_2) production. Aircrafts that make use of generators to generate hydrogen as fuel can lead to a 30% reduction in maximum take-off weight—due to the absence of fuel tanks in wings—and a 3% decrease in operating cost (Verstraete, 2013). However, for the effective operation of onboard hydrogen generators, there is a need for an interactive user interface (IUI) to allow a ground station operator or pilot to set and monitor its working conditions.

In terms of the development of the hydrogen generator's IUI, there are three most commonly used architectures: monolithic software architecture (MA), service-oriented software architecture (SOA), and micro-services software architecture (MSA) (Waseem et al., 2021). MA-based software is generally easy to develop, test and scale. However, when MA-based software starts evolving and being updated, they become complex, its structure grows larger and the number of functionality increases. This growth makes MA based software difficult to manage, scale and further increase their functionalities. Many companies still have their software as monolithic and due to the limitations mentioned they are forced to either buy new software instead of developing new functionalities (De Lauretis, 2019). Many works describe migration from MA to MSA from different perspectives. In Mazlami et al. (2017) and Gouigoux and Tamzalit, (2017), they described the migration from an architectural perspective without specifying how to execute the migrations. On the other hand, De Lauretis, (2019) and Bucchiarone et al. (2017) described migration strategy from low-level with a description of executing the migration.

SOA is another style of developing and integrating software mainly designed to address the limitations associated with MA. This approach breaks down an application into repeatable services that can be used by other internal and external applications in a system independent of the computing platform and applications relied on by the business and its partners (Dehne and DiMare, 2007). SOA has a service-based modular architecture and encapsulates many applications and data sources; hence, they offer transparency, service reusability, and flexible integration.MSA is the new trend in software architecture that takes advantage of the benefits of cloud computing. It also uses concepts and principles from SOA giving more emphasis on the design and development of scalable and maintainable software (De Lauretis, 2019). MSA is emerging as a new paradigm for programming applications through the composition of small services each running its individual processes. De Lauretis, (2019) highlighted some key benefits of MSA which include scalability, replaceability, improved maintainability, higher reliability, fault-tolerant, and better code understandability.

Various programming languages could be used to implement the software architecture of the IUI. However, scripting language (JavaScript) was adopted for development due to its ability to coexist with other programming languages (Frankston, 2020). A JavaScript runtime environment, called 'Node.js', was implemented in (Nodejs.dev, 2018). The software is open source and has a cross-platform tool that can be used for any kind of project. In Keshet and Ketterle, (2013), a graphical user interface-based control system for output sequences atomic physics experiment was described. The developed software makes use of a client-server separation between a user interface for sequence design and a set of output hardware servers. Monitoring, maintenance, and repair procedures were developed on a supervisory control and data acquisition (SCADA) system for a smart microgrid project (Palma-Behnke et al., 2011).

To address these issues, this work focused on the development of an IUI that could be used to teleoperate hydrogen generator. The IUI was developed using MSA and JavaScript due to their aforementioned advantages. The IUI can be used to switch on/off the generator, set desired output pressure, and monitor its operating conditions.

2. MATERIALS AND METHODS

2.1. Software Architecture Development

Service-oriented architecture was used to divide the software functionality between independent modules, to ensure the necessary reliability of the system, as well as perform continuous integration of new functions, as well as to adapt to changes. The software was divided into three main groups, each of which is well isolated from each other and are independent of one another (at the same time, it is only necessary to coordinate network interaction interfaces). The three groups are:

- Microservices
- A means of ensuring communication between microservices
- A user interface

A special software (Mediator) was used to meet the requirements of the hydrogen generator system. A mediator is a cross-platform software that provides components of one or more systems (microservices) with the ability to exchange data on the principle of global variables. It is the central link of the system, the interaction of the components of which is based on the star topology, which unites all microservices into an entity, providing data exchange between them. It also has a map of variables, termed the key-value pairs, with which microservices can exchange data among themselves. Microservices interact with each other on the principle of publisher-subscriber. Clients of the mediator (which are the microservices) subscribe to update certain variables with which they want to work with. When one of the clients updates the value of a variable, the others who previously subscribed to it receive this update. The mediator stores the values of variables and the timestamps of their last update, which allows you to read them at any time.

Another interesting feature of Mediator is the creation of a distributed network. Using such a network allows you to deploy an application based on microservices on several physically different computers that can be located in different geographical locations, thereby increasing the fault tolerance of the system. Traffic between nodes is filtered, which reduces the load on the network between them.

2.2. Design and Development of Microservices

The microservices software scheme is depicted in Figure 1. Based on the requirement of this work, 10 microservices were designed using MSA architecture software, each having a set of variables it subscribes to (reads), and a set of variables that it updates (acts as a manufacturer, publisher). Furthermore, the data of the microservices are linked to the mediator.

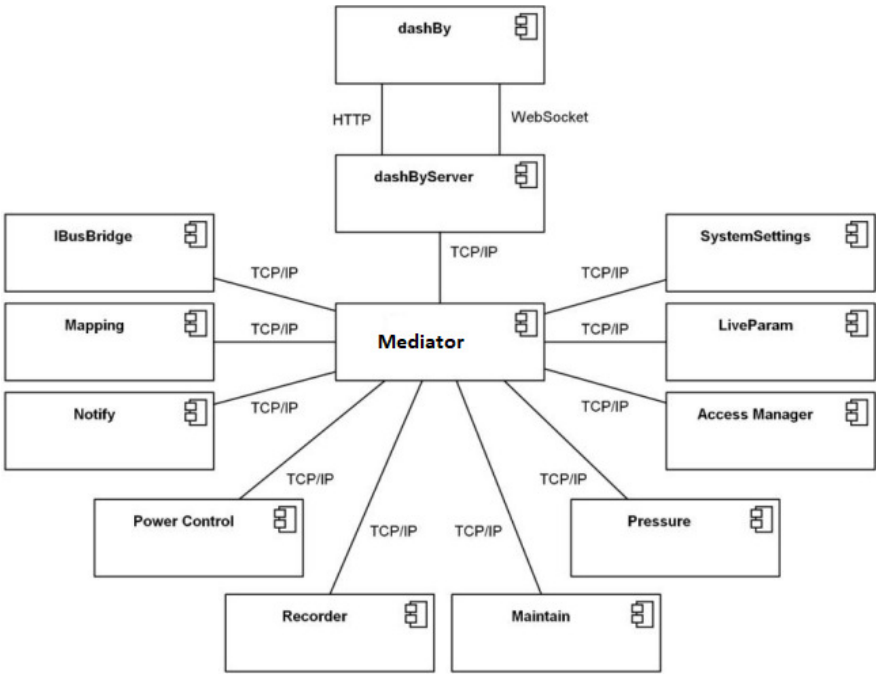

Figure 1: Microservices

Microservices read configuration files at startup and are configured according to them. This is necessary to adapt to the hardware configuration of generators of different models. Also, microservices can be used in another project (code reuse) without significant changes in the code itself. Monitoring of microservices is delegated to the operating system itself. Each microservice in the system works as a system service with special rights. To deploy, manage their launch, and restart (in case of an unexpected failure), the "systemctl" utility is used. Microservices maintain log files and terminate when a critical error occurs if it does not allow further work to continue. In case of a failure, the operating system registers this event in the system log and restarts the fallen process. The following subsections are the list of all the microservices used in the development of the hydrogen generator:

2.2.1. IBusBridge microservice

The main task of the IBusBridge microservice is to collect diagnostic data from all subsystems of the generator. The microservice opens the serial port to which the generator subsystems are connected, and then cyclically polls each hardware module. This cycle never ends for an objective reason. The read data is sent

to the mediator so that other services can then read it. The microservice allows commands to be send to generator devices. For this, IBusBridge subscribes to variables, each of which is tied to a specific device on the bus. Writing to such a variable means the need to interpret its value as a control command with its subsequent sending to the corresponding subsystem of the generator.

2.2.2. Mapping microservice

Mapping microservice converts hexadecimal data (as microservice read data is in this format) to numeric values to allow the exchange of commands with other microservices and also maps one variable to another. The main task of mapping microservice is to extract the hardware-dependent variables that IBusBridge works with, reducing them to variables that are "understandable" to other microservices to ensure compatibility with various types of generators. Also, mapping microservice perform linear transformations on raw data, read from the generator subsystems, into a “human-readable” form to be used in calculations, display on the IUI, standards (such as a unit of measurement).

2.2.3. Recorder microservice

IBusBridge and mapping microservices allow for partial control of the generator if commands are written to certain mediator variables. This can also allow for the setting up of the generator and starting it, but there is a nuance: the preset settings will be reset as soon as the generator is disconnected from the power grid. The recorder microservice subscribes to the mediator variables set in its settings and monitors their update. When one of the microservices updates such a variable, the recorder gets its new value and puts it in its storage. The next time the microservice is started, the variables are read from the storage and written to the mediator. Thus, it becomes possible to save the state of the software and then restore it at the next startup.

2.2.4. Power control microservice

There is need to turn on or off the generator, and start or stop gas production. The use of modular hardware in the new line of generators allows for a reduction in the cost of production and production of ready-made devices. However, with this approach, there is a need for centralized management of all subsystems of the generator. The task of centralized control of generator subsystems, their simultaneous start, and stop, as well as the control of the operating modes of the entire system as a whole, is implemented in the power control microservice.

2.2.5. Pressure microservice

The pressure microservice is used to monitor the correspondence of the measured pressure at the generator outlet to the one set by the user. The pressure microservice also monitors a sharp drop in pressure in the system, which may be caused by some local "accident", for example, a break in the air pipe through which gas is supplied to the consumer. In this case, the microservice sends a command to stop the generator to the power control microservice, and the notification center registers a corresponding message about a potential "accident".

2.2.6. Notification microservice

The notifications microservice constantly collects information about the states of microservices, analyzes it, and decides the health of the system, which is then displayed as one of the messages on the main screen: GOOD ("Good"), WARNING ("Attention") or ALARM ("Accident/Failure"). Also, this microservice provides the functionality of the notification center screen, which provides the user with the opportunity to view details about the operation of the device, as well as current notifications about important events or errors.

2.2.7. Maintenance microservice

Planning of gas generator maintenance tasks and control of their execution by personnel are implemented in the maintenance microservice. The microservice keeps a schedule of deadlines and a history of maintenance (maintenance) and notifies the user about the upcoming deadlines of maintenance tasks. The microservice is subscribed to mediator variables for the current time in Unix format and the current day of the month.

2.2.8. System settings microservice

The microservices considered previously in section 2.2.1 to section 2.2.7 can be classified as basic. The microservice in this section and section 2.2.9 and 2.2.10 provide additional functionality and configuration of the software and the entire system. The system settings microservice takes full responsibility for the settings menu. This is the only microservice of all that modifies the operating system settings and runs with super-user rights. These rights are required to allow the process to write to the system directories of the file system.

2.2.9. Live parameter microservice

The live parameter microservice is related to the functionality of viewing diagnostic data on the map (section 2.2.7) and how they are displayed. The variables that the microservice outputs in the diagnostic table are read from the mapping microservice configuration file.

2.2.10. Access manager microservice

Access manager microservice monitors user activity in the system and performs automatic blocking of the user interface in case of long inactivity of the first one (section 2.2.1). Because microservices do not have access to the interactive user interface and its components but can communicate with them only through mediator variables, it was decided to slightly expand the functionality of the JavaScript widget library, which is described in detail in the next section and is used to develop the Front-end part of the software. A "hidden" widget has been added to the library, with which the Access Manager microservice monitors transitions between screens and, if necessary, can also manage them.

2.3. Design and Development of Interactive User Interface

The design of modern devices, especially from the “Internet of Things” series, often requires a web panel that provides the user with the opportunity to interact with the equipment in real-time. The dashBy widget library.js was used for this work and is shown in Figure 2. This library is completely written in JavaScript and is aimed at minimizing efforts when creating and managing human-machine interfaces over the Internet.

Figure 2: Widgets from dashBy.js library

The framework consists of two parts: front-end library *dashBy.js* and back-end tool *dashByServer.js*. The front-end is a script that parses a web page, once it is loaded into a browser and converts special tags into the controls (gauges, charts, buttons...).

Example: {::range(throttle){0,100}::}
{::chart(throttle, speed){400,200,1m30s}::}

Back-end tool provides interface to user applications through ASCII strings over TCP with the following format:

<variableName>=<value> Example:
speed=121.5
throttle=35
oil.level=77
oil.temperature=91.2
oil.status=ok

The server part of the library is an application written in JavaScript and requires a software platform (Node.js) for its launch. The application is an undemanding web server that allows clients (web browsers) to receive static web pages and resources (for example, images and JS scripts). The server application also supports the WebSocket protocol, which is used to organize two-way data exchange between the client and the server, thereby ensuring the interactivity of widgets on web pages. The network architecture of this library is shown in Figure 3.

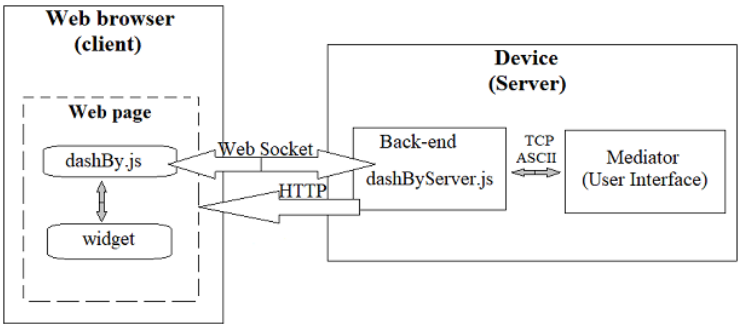

Figure 3: Network communication between client (front-end) and server (back-end)

By default, dashByServer.js opens an HTTP server with WebSocket support at port 8080 TCP server for local client connections opened at port 8000. WebSocket connection timeout is 2000 msec. These settings can be modified by loading them from an "*ini*" file using the command line option:

node dashByServer.js -c dashBy.ini

Here is an example of dashByServer.ini file

[http]
port=8080
[websock]
timeout=2000
[TCP]
port=8000

The development of the user interface software begins with the main screen and takes place simultaneously with the development of microservices: A list of web pages that were developed during the preliminary design of the software has been compiled as follows: main page; notification center page; page settings for automatic shutdown of the generator; mode selection page; diagnostic data page; quick start guide page; maintenance task scheduler page; maintenance history page; maintenance task view page; settings menu page; date and time settings page; wireless network connection settings page; LAN connection settings page; unit settings page; screen saver settings page; remote access settings page; and lock code settings page.

2.4. Hardware Prototype Development

Developing a prototype hydrogen generator requires a complete definition of all the components to be used, as well as the assembly of all the components into a single unit, running the generator itself, and testing the model. The following are all the components used for the development of the prototype:

- Pump engine generator (KNF model: NF 1.100_EX)
- Static dryer
- Power supply with open housing
- Raspberry Pi 3 Model B
- LCD display

The pump holds up to 1.2 bar of pressure. The internal pump forces distilled water to flow from the external water reservoir to the proton exchange membrane (PEM) electrolysis cell; mixed with oxygen, a byproduct of electrolysis, the water returns to the tank. On the way into the chamber, the water is filtered and then deionized through a special cartridge and its measured conductivity. The wet hydrogen then passes through the membrane and is dried using a gas-liquid separator and then a static dryer. The dry waterfall is then passed through a high-performance purification module based on the pressure swing adsorption (PSA) principle, where the final pressure is adjusted by a proportional valve.

The power supply with open housing is designed for integration or installation within a system enclosure. The Raspberry Pi used for this work has a 64-bit Quad-core ARM cortex-A53 processor with a clock frequency of 1.2 GHz per core as part of the Broadcom BCM2837 single-chip platform. A 5-inch LCD with a resistive touch screen was integrated along with all other components to obtain a final prototype as shown in Figure 4.

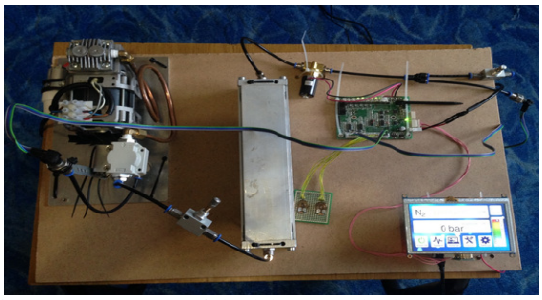

Figure 4: Prototype of the laboratory hydrogen generator

2.5. Software Testing

Based on the configuration shown in Figure 4, different scenarios of the software testing are presented in the following subsections.

2.5.1. Positive testing scenario

Operating mode: The test was to start the generator with a precondition that the subsystems of the generator are in a disconnected state; the power of the generator itself is turned off. Activating the ON button in Figure 5 turns the generator on and starts the hydrogen production. There is an automatic transition back to the main screen (Figure 6). The switch "ON" indicator at the bottom left corner shows the generator is "ON" i.e., it is live.

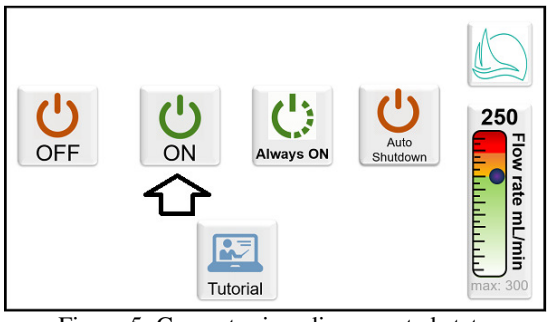

Figure 5: Generator in a disconnected state

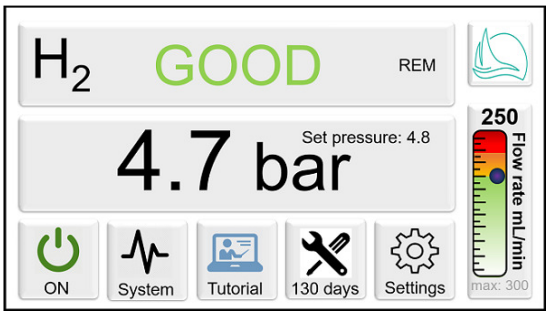

Figure 6: The expected result after conducting the test

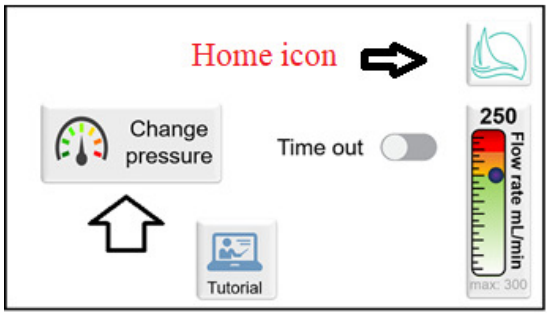

Figure 7: Pressure panel of the hydrogen generator

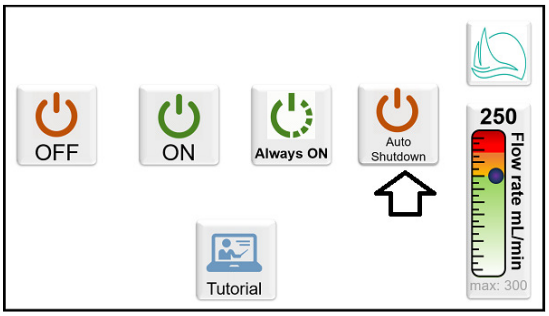

Figure 8: Selecting Auto Shutdown

The pressure module setting the supported pressure with a precondition that the generator subsystems are in the switched-on state is shown in Figure 6. The set pressure measurement unit is the "bar". The following set of steps were taken to conduct the test:

- The pressure value was activated in Figure 6
- The pressure value was changed to the permissible pressure value in bars (value ranges from 0.1 to 10 bar) as shown in Figure 7
- The Home icon is activated

Once the pressure value is changed, the pressure in the system begins to slowly increase or decrease (depending on the current and entered values), as indicated by the reading on the main screen (Figure 6).

A test was then conducted on the System module that turns off the generator when there is insufficient pressure, with a precondition that the gas consumption is not connected. The following set of steps was taken to conduct the test:

- Activating the space where the pressure can be entered gives the option to activate the "ÄUTO SHUTDOWN" button

After a specified period, the generator is automatically switched off. A corresponding notification is displayed on the screen.

2.5.2. Negative testing scenario

To show the output of the user interface when the internal subsystem of the system module breaks down and when the generator is enabled, the following steps were taken:

- To ensure that the communication of the software with the generator subsystems is fully functional. The status "INFO" or "WARNING" is displayed on the main screen of Figure 9.

- Disconnecting one or more generator subsystems from the internal RS-485 bus changes the system status to an "ALARM" state as depicted in Figure 10.

Figure 9: The main screen of the generator is in a "WARNING" state

Figure 10: The expected result after disconnecting one or more subsystems

3. RESULTS AND DISCUSSION

Based on the configuration shown in Figure 4, the results of the experimental setup are presented in the following subsection. This section discusses an example of a scenario for using the developed software to control a hydrogen generator. Only the main functions of the application that are often used by the user (laboratory personnel interacting directly with the device) are demonstrated. For the generator to be ready for operation, it takes time to load all its components. The application starts automatically after the start of the operating system, which itself starts immediately when the generator is connected to the supply voltage network. Figure 11 is the main screen (default) once the generator is turned on. At the very top left of the main screen is a gas molecule, or rather its textual representation. In this case, it is hydrogen "H_2". This logo may differ depending on which type of generator this software is deployed on: hydrogen, nitrogen, or "clean air". At the lower right is the system status indicator; more detail on its function will be provided. Below the gas indicator button is a panel that displays the current gas pressure in the system (at the generator outlet), as well as the one set by the user. In the upper right corner, there is a button for a quick return to the main screen ("Home"), which is present on all other screens, and in this case, is used as a logo. Under the "Home" button, there is an indicator of the current consumption of gas (hydrogen) in liters per minute. At the bottom of the screen there are five buttons: "Operating mode", "System", "Help", "Maintenance" and "Settings".

The inscription "WARNING" indicates an important system events notification. The notification center screen is shown in Figure 12, which is the name of the system event notification screen, displays a detailed

description of the notifications and their level of importance: “INFO”, “WARNING”, and “ERROR”. The latter indicates a problem in the system and the need to seek qualified help from the manufacturer.

The event on the notification center screen indicates that one of the generator maintenance tasks was not performed on time, which is also indicated by the sign “Expired” on the corresponding button on the main screen (Figure 11). Once a task is completed, the user is required to add a note to the system, before the system can recalculate the next schedule of maintenance, as depicted in Figure 13. Figure 13 shows the maintenance tasks that must be performed periodically by personnel to maintain the generator in working condition and extend its service life. This screen displays the expiration date of a task, as well as the number of remaining days, in place of which “Expired” is now displayed. Below this, is the text description of the task.

Figure 11: The main screen of the software after turning on the generator

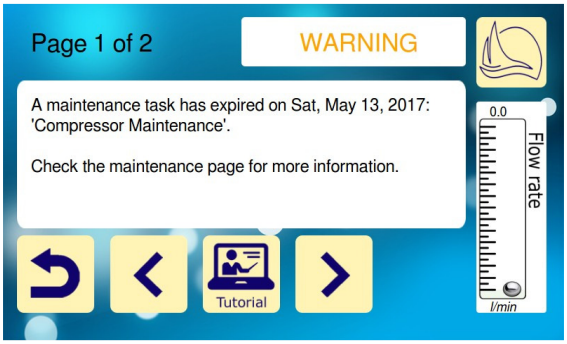

Figure 12: Technical maintenance has expired

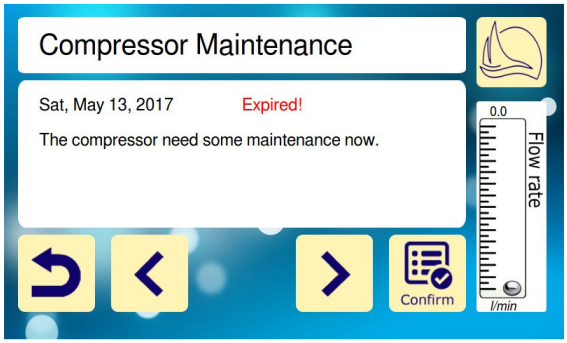

Figure 13: Entry action of technical maintenance implementation

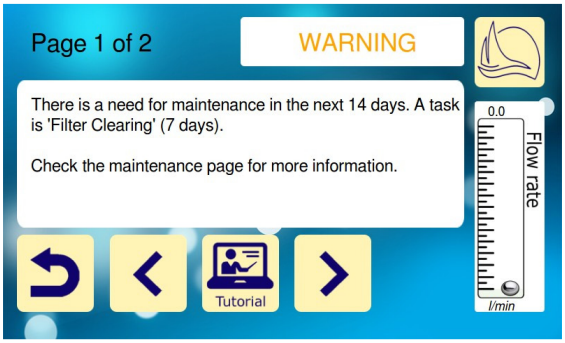

Figure 14: Notice page of next technical maintenance

Once maintenance is completed successfully the notification on Figure 12, warns about its occurrence. By default, the system notifies the user about these 14 days in advance as shown in Figure 14. Figure 15 shows the maintenance schedule tasks that were conducted (date) and the next time it will be required.

After the necessary maintenance, which is performed to maintain the generator in working condition, as well as to prevent a decrease in the quality of the gas produced, the generator is ready for operation. This should be indicated by the status of the system “GOOD” at the very top of the main screen (Figure 11). Activating the “OFF” button in the main screen (Figure 11) enable a smooth navigation to the mode of operation page shown in Figure 16. The three buttons here as indicated turns “OFF”, “ON”, and “Always ON” (automatically keep the generator on after a restart).

Activating the "ON" button, the generator starts directly and start producing pure hydrogen (subject to the presence of distilled water in the tank), suitable for use in chromatographic installations or other purposes as shown in Figure 17. To stabilize the pressure in the system, the generator needs some time, depending largely on the set value. The higher the pressure, the more time it takes to stabilize it. This scenario of using a hydrogen generator is typical in the laboratory and does not consider such functions of the developed software as configuring access from the network and other parameters of the application and operating system, troubleshooting, remote access, and several other functions that are secondary in laboratory practice.

Figure 15: Maintenance schedule task page

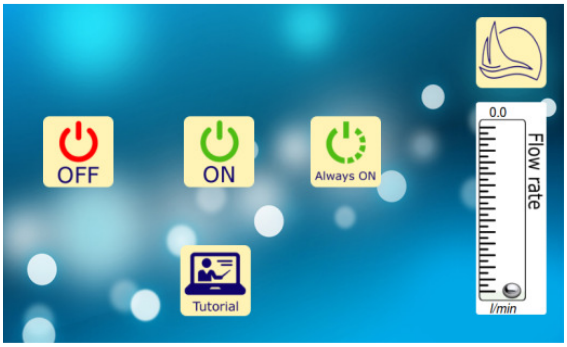

Figure 16: Selecting generator mode of operation

Figure 17: Hydrogen production started page

4. CONCLUSION

The development of an interactive user interface that mainly focuses on monitoring a hydrogen generator was presented. The IUI is based on an architecture that is built on microservices. Thus, a service-oriented architecture was used to properly split the software functionality into independent modules. Testing of the software solution was carried out on real equipment, which made it possible to identify and correct design errors directly during development. The result of this work was a universal software for monitoring and controlling not only hydrogen generators but also other pure gas generators.

5. CONFLICT OF INTEREST

There is no conflict of interest associated with this work.

REFERENCES

Akdeniz, H. Y. and Balli, O. (2021). Energetic and exergetic assessment of operating biofuel, hydrogen and conventional JP-8 in a J69 type of aircraft turbojet engine. *Journal of Thermal Analysis and Calorimetry*, pp. 1–13.

Baharozu, E., Soykan, G. and Ozerdem, M. B. (2017). Future aircraft concept in terms of energy efficiency and environmental factors. *Energy*, *140*, pp. 1368–1377.

Bicer, Y. and Dincer, I. (2017). Life cycle evaluation of hydrogen and other potential fuels for aircrafts. *International Journal of Hydrogen Energy*, *42*(16), pp. 10722–10738.

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T. and Mazzara, M. (2017). *From Monolithic to Microservices: An experience report*. Technical Report. Technical University of Denmark, Örebro University, TU.

De Lauretis, L. (2019). From monolithic architecture to microservices architecture. *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 93–96.

Dehne, D. and DiMare, J. (2007). Service-oriented architecture-Unlocking hidden value in insurance systems. *IBM Global Business Service*, *8*(7),

Elitzur, S., Rosenband, V. and Gany, A. (2017). On-board hydrogen production for auxiliary power in passenger aircraft. *International Journal of Hydrogen Energy*, *42*(19), pp. 14003–14009.

Frankston, B. (2020). The javascript ecosystem. *IEEE Consumer Electronics Magazine*, *9*(6), pp. 84–89.

Godula-Jopek, A. and Westenberger, A. (2016). Hydrogen-fueled aeroplanes. In: *Compendium of hydrogen energy* (pp. 67–85). Elsevier.

Gouigoux, J.-P. and Tamzalit, D. (2017). From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 62–65.

Keshet, A. and Ketterle, W. (2013). A distributed, graphical user interface based, computer control system for atomic physics experiments. *Review of Scientific Instruments*, *84*(1), p. 015105

Khandelwal, B., Karakurt, A., Sekaran, P. R., Sethi, V. and Singh, R. (2013). Hydrogen powered aircraft: The future of air transport. *Progress in Aerospace Sciences*, *60*, pp. 45–59.

Kim, J. and Kim, T. (2014). Compact PEM fuel cell system using chemical hydride hydrogen source for portable power generators. *Energy Procedia*, *61*, pp. 1992–1995.

Kim, T. and Kwon, S. (2012). Design and development of a fuel cell-powered small unmanned aircraft. *International Journal of Hydrogen Energy*, *37*(1), pp. 615–622.

Mazlami, G., Cito, J. and Leitner, P. (2017). Extraction of microservices from monolithic software architectures. *2017 IEEE International Conference on Web Services (ICWS)*, pp. 524–531.

Nodejs.dev. (2018). *Introduction to Node.js*. https://nodejs.dev/learn/introduction-to-nodejs

Palma-Behnke, R., Ortiz, D., Reyes, L. and Garrido, N. (2011). *A Social SCADA Approach for a Renewable based Microgrid – The Huatacondo Project*.

Petrescu, R. V. V., Machín, A., Fontánez, K., Arango, J. C., Márquez, F. M. and Petrescu, F. I. T. (2020). Hydrogen for aircraft power and propulsion. *International Journal of Hydrogen Energy*, *45*(41), pp. 20740–20764.

Verstraete, D. (2013). Long range transport aircraft using hydrogen fuel. *International Journal of Hydrogen Energy*, *38*(34), pp. 14824–14831.

Waseem, M., Liang, P., Shahin, M., Di Salle, A. and Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, *182*, p. 111061.

Westenberger, A. (2003). Hydrogen fueled aircraft. *AIAA International Air and Space Symposium and Exposition: The Next 100 Years*, 2880.

Winnefeld, C., Kadyk, T., Bensmann, B., Krewer, U. and Hanke-Rauschenbach, R. (2018). Modelling and designing cryogenic hydrogen tanks for future aircraft applications. *Energies*, *11*(1), p. 105.