Modernization of a SCADA system by integrating a SMS sending system and a cloud platform

Constantin-Adrian Ureche
Universitatea Politehnica din Timisoara
Timisoara, ROMANIA

Abstract — In the current context marked by a rapid advancement of technologies and an increased interest in digital transformation, the concepts of Internet of Things (IoT) and Industry 4.0 are becoming more and more relevant for the modernization of the existing industrial infrastructure. With an exploding market and increasing adoption of advanced technology solutions, the need to integrate traditional systems - sometimes described as "legacy systems" - with new IoT paradigms is becoming inevitable. This paper aims to address this challenge through a solution that strategically combines four key components to facilitate a smooth and efficient transition to a contemporary digital environment. The first component is an existing web interface, ported from the legacy system. The next vital component is a modern SMS messaging system, integrated through a API service. The third component is an innovative script that, through the use of Selenium, enables accurate collection and extraction of data from the legacy web client. Finally, we integrate the these components into a robust Cloud infrastructure, composed of a Azure Table Storage for efficient and scalable data storage and an API that allows easy data access via HTTP protocol. The integration proposed in this article highlights how the implementation of these modern technologies can revolutionize legacy systems, turning them into IoT entities. Through this, organizations can achieve significant benefits in terms of cost efficiency, reliability, improved performance and the possibility of advanced customization, thus gaining a competitive advantage in an ever-changing market.

Keywords—Industrial Internet of Things, Industry 4.0, Legacy Systems, Web Integration, Selenium, Twilio

I. INTRODUCTION

The rapid development of the Industrial Internet of Things (IIoT) as well as Industry 4.0 had a major impact on multiple fields of interest and traditional manufacturing organizations. Communication between people, individual products and production is fluid due to a paradigm shift from the previous, centrally controlled production to a decentralized one, as a result of the integration of IIoT protocols and technologies [1].

Considering the current state of the industry, which uses legacy systems that are largely based on outdated local protocols, the need to evolve and expand production and monitoring processes is a solid foundation for using and integrating into an IIoT network. The integration of network technologies and smart computing in the field of production and the generation of an environment that reinforces the goals of automation, reliability and control that underline the development of an IIoT is represented by the Industry 4.0 paradigm [2].

The significance of a legacy manufacturing system – IIoT integration can be seen in related research papers. The existence of a large body of research focused on IIoT adaptation on legacy systems confirms the contemporary interest in this particular field. The preponderance describes a very broad perspective, including IoT adapters, gateways, as well as management architectures, but does not provide an end-to-end or concrete implementation to achieve our goal.

For example, the authors in [3] aim to modernize long-standing industrial systems and present an overview of a multitude of technologies and methodologies that ease overall integration with an IIoT network and address the legacy migration difficulties that might arise along process. The studies were conducted based on the interpretation of the results of a systematic mapping study, in addition to information on the state of practice available in the specific areas of interest.

In the perspective presented in [1], in order to achieve IoT connectivity, an adapter board was designed to invoke legacy functions as services, thus creating an interconnected network. Afterwards, the whole system is easily scalable, as the board is able to use the legacy system when creating additional functionality, as it is an important aspect considering the need for new functions for the existing system. To demonstrate all of the above, the authors designed a small system that consists of adapting a small distribution line to IoT by using a programmable logic controller and presents the most important advantages of cost efficiency, adaptability and scalability.

In this study, we advance a technology integration proposal that involves effectively combining a legacy application, consisting of a web server and an associated client, with the modern infrastructure of an SMS notification system and the cloud computing capabilities provided by the platform Azure. This technological synthesis aims to capitalize on the advantages offered by the digital age to fully comply with the requirements of Industry 4.0, which ensures an advanced integration and a competitive advantage in the current technological ecosystem.

This paper is structured starting with section 2, which details the architecture and implementation of the proposed system, followed by section 3 which presents the empirical results of the application of these integrated technologies. The tools and frameworks used are detailed in sections 4, 5 and 6. The discussion and analysis of the results are then deepened in Section 7, and finally, the conclusions of the study summarize the key contributions and perspectives for future research. This structure allows for a systematic and meticulous assessment of the performance and applicability of the proposed integrated architecture, while promoting a clear understanding of the benefits that the cross-cutting integration of technologies can bring in the context of current and future industry needs.

II. ARCHITECTURE

With its reconfigured role, the web server hosted on a PLC plays the role of a technological bridge between industrial machines and users who want to access and monitor them. Through this server, the web interface becomes an intuitive and agile control panel that opens a window to the flow of industrial data, allowing the monitoring and adjustment of industrial process parameters, even remotely. The ability to view critical information in real time and execute commands to change operating parameters thus transforms the server into a centerpiece for optimized efficiency and control. The flexibility of this architecture in terms of compatibility with a wide range of industrial devices and protocols recommends it as a versatile solution, essential for systematic supervision, diagnosis and control, crucial elements for strengthening the decision-making process.

On the other hand, the script designed in Python becomes an autonomous monitoring and alerting agent, calling on Selenium's web browsing and interaction capabilities to query the server and extract critical data such as key operational metrics. With precise programming logic, it analyzes and compares actual parameters with preset thresholds to identify any anomalies or conditions that exceed safe operating norms. At such critical moments, the script activates a notification protocol, calling Twilio – an API communication service – to send SMS alerts to the responsible technical or administrative staff. This automated surveillance and early intervention mechanism has a double advantage: on the one hand, it ensures constant and fail-safe monitoring of the system, eliminating the risk of human omission, and on the other hand, it optimizes the response time of the technical team, allowing an intervention fast and wise, which can be decisive in preventing breakdowns and preserving the integrity of complex industrial systems. This synergy between technologies significantly advances the ritual of monitoring and reaction within industrial operations, providing a preventive side that complements the maintenance and safety strategy in this new digital age.

Selenium is a suite of web browsing and test automation tools used to control automated interaction with web browsers [5]. Developed primarily for software testing, Selenium allows programmers to simulate human actions on web pages, including filling out forms, navigating, and extracting information. Being open-source and supporting multiple programming languages, Selenium is becoming a popular option for developing web automation scripts [8].

Twilio is a cloud communications platform that provides SMS messaging, voice calling, and faxing services for developers. Through its user-friendly APIs, Twilio enables easy integration into apps and websites, enabling the automatic sending and receiving of text messages and calls, thus enhancing effective communication between users [9].

Azure is a cloud services platform provided by Microsoft that provides a wide range of resources and services for developing, deploying and managing applications in the cloud environment [6]. Azure provides services such as compute, storage, databases, data analytics and more, making it easy to scale and efficiently manage your IT infrastructure. [7]

Azure storage account is an essential service that provides durable and scalable storage in the cloud. It allows storage and management of files, data and other resources, accessible remotely through Azure APIs. With advanced security options and the ability to customize performance levels, Azure storage accounts provide a reliable infrastructure for managing data in the cloud. [7]

Implementing SMS sending and data publishing functionality to the Cloud starts with reading them from the web interface, a process executed by a Selenium-based Python script. This versatile framework is often chosen for automating website testing [10], but in this context it facilitates the precise extraction of data required for monitoring. The script acts as a virtual user, browsing and collecting essential data for the next steps of processing and alerting.

This data is then targeted by the script for continuous monitoring. When variations are detected that exceed pre-set thresholds, the SMS alert mechanism comes into action, realized through the Twilio API service, which eliminates the need for a dedicated GSM device [11]. This system streamlines the quick and direct communication of alerts to specific recipients.

Towards the end, the script also receives the task of uploading the observed data to the Cloud, using Microsoft Azure as the storage platform. Every value change detected on the site is thus recorded in the Cloud, providing a centralized, secure and accessible database for further analysis and for making informed decisions based on the updated and complete history of the monitored data.

Figure 1 shows the general architecture of the system together with all the connections between them. The first component, the web page, provides the information, which the python script then retrieves, analyzes, stores on the cloud server and decides whether warning messages need to be sent.

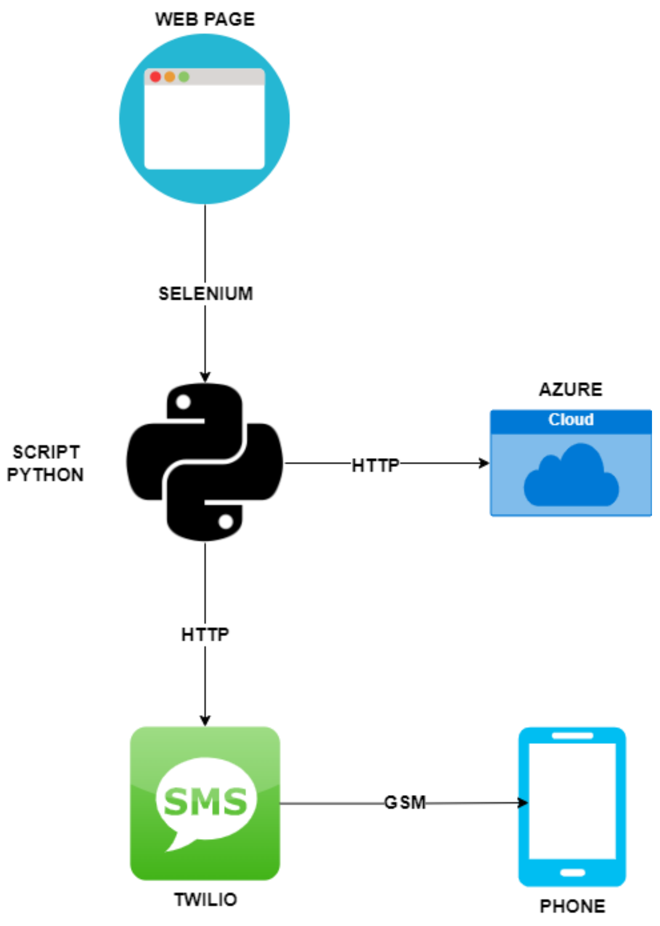


Figure 1 - Architecture

The data is retrieved from the web page through Selenium using the element IDs set in the HTML code of the site. Then this data reaches the Microsoft Azure Cloud and Twilio platforms via the HTTP protocol. Specific APIs to each platform are used to communicate with them.

III. IMPLEMENTATION

The reading of data from the web page is continuous, following the logical algorithm shown in figure 2. The first information is read, then it is saved in the cloud, then it is decided whether to send a warning SMS message, then it moves to the next information and the flow repeat until all information on the site is read. After all information is read, the algorithm repeats, this time checking if the value of each information has changed. If any value changes, then it is saved in the cloud and it is decided whether an SMS warning message should be sent.

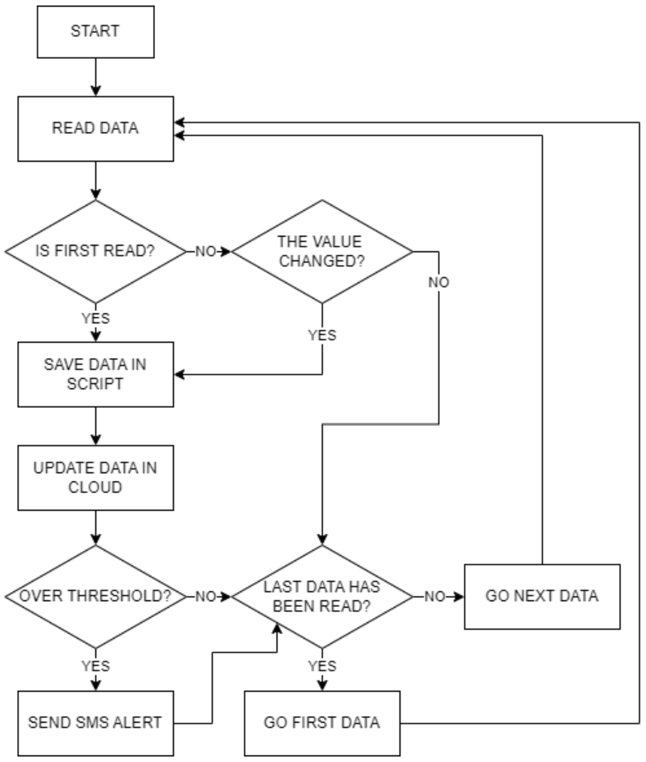
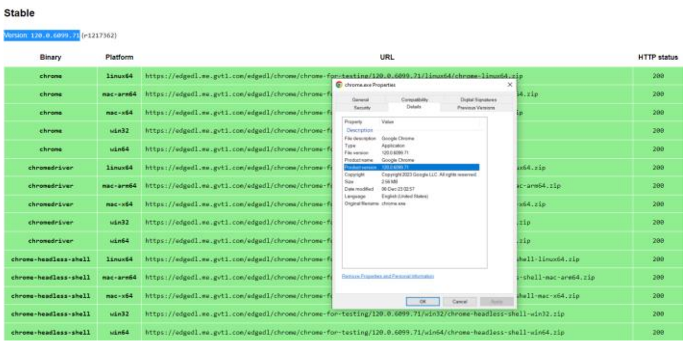


Figure 2 - Logic algorithm of the Python script

To use Selenium, the driver associated with the browser to be used with Selenium must be downloaded [12]. Google Chrome was used in the development of this project and the driver can be downloaded from the official ChromeDriver page https://sites.google.com/chromium.org/driver/. The driver version must be compatible with the browser version as shown in Figure 3 [12].



Stable

Version: 120.0.6099.71 (r1217362)

Binary	Platform	URL	HTTP status
chrome	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/linux64/chrome-linux64.zip	200
chrome	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f 64.zip	200
chrome	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f ip	200
chrome	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f	200
chrome	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f	200
chromedriver	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f ux64.zip	200
chromedriver	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f ac-arm64.zip	200
chromedriver	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f x64.zip	200
chromedriver	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f .zip	200
chromedriver	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f .zip	200
chrome-headless-shell	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f hell-linux64.zip	200
chrome-headless-shell	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f -shell-mac-arm64.zip	200
chrome-headless-shell	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-f hell-mac-x64.zip	200
chrome-headless-shell	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win32/chrome-headless-shell-win32.zip	200
chrome-headless-shell	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/120.0.6099.71/win64/chrome-headless-shell-win64.zip	200

chrome.exe Properties

Property	Value
Description	
File description	Google Chrome
Type	Application
File version	120.0.6099.71
Product name	Google Chrome
Product version	120.0.6099.71
Copyright	Copyright 2023 Google LLC. All rights reserved.
Size	2.56 MB
Date modified	06-Dec-23 02:57
Language	English (United States)
Original filename	chrome.exe

Figure 3 - Browser and Driver's versions check

The decision to send SMS messages is made differently for each type of date because each of them has its own set limits. Limits are set by the programmer in agreement with the client and cannot be changed by the user. To use Twilio, the dedicated library was used, which can be installed by typing the pip install twilio command in the terminal [13], and to use the Selenium framework, the library dedicated to it was used, which can be installed by typing pip install selenium in the terminal window [14].

After the driver is downloaded and the both framework, Selenium and Twilio, are installed, an object of type Service that is part of the selenium.webdriver.chrome.service library can be created in the python script [14]. This object must receive the path to the driver at the time of its initialization service = Service('chromedriver.exe'). After its initialization functions can be called to interact with the browser. The get("url") function opens the web page from the url received as a parameter. Another frequently used and very important function is find_element(selector, search_value) which returns objects corresponding to elements in the HTML code of the site. For example, an input box can be searched for by its id in the HTML code driver.find_element(By.ID, id). The returned object contains all the properties that the element possesses in the HTML code at the time of reading. Furthermore, the element on the web page can be modified by changing the parameters on the object in the python code [14].

Each type of data is considered an object in the Python script and the following 6 pieces of information are required for each object:

- id - the element id in the HTML code of the page
- web element - an object of type WebElement, class specific to the selenium library. This object contains all the information about that element that can be accessed from the web page
- the value of the element before the last read
- the value of the element after the last read
- minimum limit
- the maximum limit.

When the object is constructed, the value before the last read and the value after are equal because these values are equal to the value resulting from the first read.

At each reading of each element, it is checked whether its value has changed. If the value for an element has changed, it is checked whether the new value is within the set limits. If the value is above the established limit, an SMS warning message is sent with the format "ID element over threshold", respectively "ID element under threshold" if the value falls below the minimum limit.

IV. TWILIO

Using Twilio in the context of our project to alert users via SMS when collected data exceeds certain thresholds provides very good scalability. For example, if web sensors detect a temperature fluctuation in a monitored industrial system, the Python script can automatically format and send an alert message through Twilio. This guarantees that the appropriate persons are immediately notified, allowing them to take emergency action if needed. This notification method, thanks to Twilio's easy integration, may adjust to reflect changes in alerting procedures or the data stream collected, demonstrating its flexibility in deployment.

In addition, Twilio conversations include a strong encryption [9], which is critical for keeping sensitive information secure while in transit. For example, when an alert message is sent from our cloud service to staff phones, the data is converted from one format to another and transmitted over potentially insecure networks using HTTPS and SSL/TLS encryption, guaranteeing that the information remains concealed. This built-in security is critical because its absence could jeopardise both data confidentiality and the monitoring system's integrity.

Despite all these benefits, reluctance may sometimes be felt regarding reliance on Twilio, as the ongoing maintenance of alerts depends on its services. If Twilio experiences a service outage or technical issues, our alerting system may suffer - a risk that should be agreed at the start of a project. This risk is accompanied by financial concerns: because our project is based on real-time data and frequent alarms, costs might quickly climb as the volume of data increases. Monitoring these financial ramifications is critical to keeping the budget on track and avoiding unpleasant surprises.

Twilio's cost model is clear when our monitoring system identifies irregularities and automatically sends SMS notifications via the service. The cost per message might quickly build up on a day with exceptional activity or when the monitoring parameters are set too sensitively. Adjusting alert frequency and circumstances, and possibly even instituting an approval system for non-critical alerts, become viable cost-cutting measures that we can consider after the system is in place for a longer period of time.

V. SELENIUM

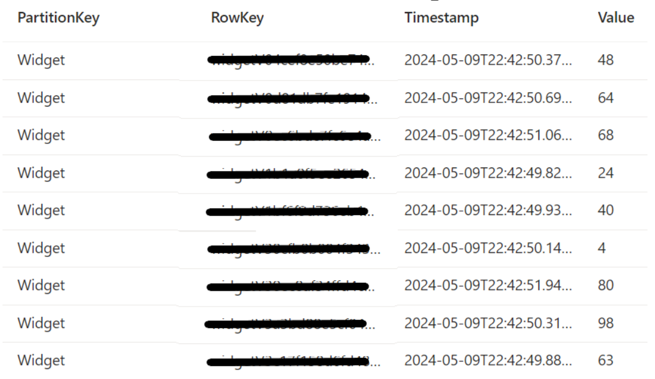
Selenium WebDriver facilitates the simulation of specific actions in browsers such as Chrome or Firefox, allowing task automation and efficient data extraction. To interact with the elements of a web page, Selenium uses methods specific to the programming language whose WebDriver is used to identify and manipulate the page's DOM elements [8].

In this study it was necessary to configure Selenium WebDriver using Python as the programming language. The initial step was to install the selenium package using a language-specific package manager such as pip. Once installed, it is necessary to identify the appropriate driver for the Chrome browser - ChromeDriver - and define its path in the Python script to enable communication between the written code and the browser. ChromeDriver can be downloaded from page https://sites.google.com/chromium.org/driver/ and the version must be the same with the one of the Chrome browser such in Figure 3. Once these steps are completed, a ChromeDriver session can be initiated which is then used to navigate to a predefined URL, marking the start of automating interactions with the website [8].

The WebDriver provides a set of tools for selecting and manipulating HTML elements. Interaction with these elements is essential for extracting information from the web page, be it text, links or other relevant data. Searching for elements on a page is done using different locating strategies such as ids, classes, CSS selectors or XPaths. Selenium is equipped with methods to extract the visible text of the element or to retrieve its attributes, and thus the necessary data can be collected [10].

VI. CLOUD

For the purpose of building up a solution to deal with the web data extracted, Microsoft Azure was chosen as the platform providing the storage capability through a cloud infrastructure [15]. Azure Storage Account service was chosen as the layer of data persistence because it boasts of an architecture that is more robust and scalable[15]. Thus, it is able to deal with the complexities that are accompanied by large amounts of manipulation and management of unstructured data. Through Storage Account it is possible to get redundancy and data recovery services, as such, the integrity of data can be ensured even during unforeseen cases [16]. As the default data storage system at Azure operating under NoSQL model, Table Storage has been brought in to provide the required flexibility which cannot be compromised to accommodate the divergent information sources being extracted by web scrapers [16]. The structure of the table is illustrated in the figure 4.



PartitionKey	RowKey	Timestamp	Value
Widget	[redacted]	2024-05-09T22:42:50.37...	48
Widget	[redacted]	2024-05-09T22:42:50.69...	64
Widget	[redacted]	2024-05-09T22:42:51.06...	68
Widget	[redacted]	2024-05-09T22:42:49.82...	24
Widget	[redacted]	2024-05-09T22:42:49.93...	40
Widget	[redacted]	2024-05-09T22:42:50.14...	4
Widget	[redacted]	2024-05-09T22:42:51.94...	80
Widget	[redacted]	2024-05-09T22:42:50.31...	98
Widget	[redacted]	2024-05-09T22:42:49.88...	63

Figure 4 - Cloud Table

During the data integration phase, the Azure SDK API for Python was used to establish an encrypted connection between the local scripting environment and the Azure cloud infrastructure. A secure connection string is used, thus ensuring the confidentiality and security of data transmission. In this process, the extracted data is structured into entities according to the key-value model, each entity being uniquely identifiable by a partition key and a row key. For simplicity, each corresponding record partition key is a "Widget" and the row key is the id of the element in the HTML that is displayed on read. Such an approach enables data loading that is both fast and good, as well as seamless data retrieval and subsequent data analysis that is enhanced by the increased availability of persistent data.

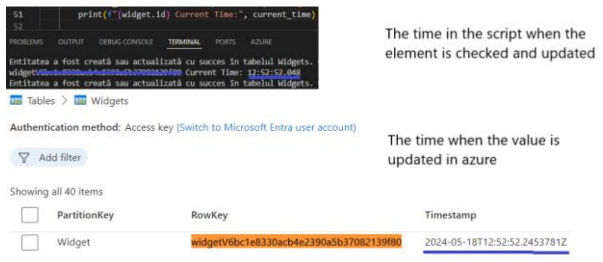
Moreover, an application programming interface (API) was created to offer better data access and to ensure that the system functions well in connection with other systems in the organization. The developed API is built on top of Python and the Flask microframework with the endpoint designed as a GET that exposes the data from the Azure Table Storage. This software solution allows the company to query and retrieve the data. An API is responsible for supplying data reported in a string format to the external systems. By using this mechanism, end clients, employing a custom approach to their application, can receive a needed data seamlessly. This API has the practical side of specifically illustrating the work of open data in practice.

VII. RESULTS AND PERFORMANCES

Following the implementation of the described project, the results obtained confirmed the efficiency of the automated workflow. The Python script, built with Selenium, was able to consistently and accurately extract data from the targeted website. It monitors the defined parameters, and in case of exceeding the pre-set limits, the system automatically triggers the sending of SMS notifications to the authorized personnel, a process that took place with a median latency of less than 5 seconds from detection, thus underlining the promptness of the real-time alerting system. The data, once collected, was stored in the Azure Cloud, with data consistency and redundancy ensuring very good durability. This system enabled not only effective surveillance, but also better predictive and retroactive analysis capability, thanks to flexible and scalable cloud infrastructure, thereby contributing to improved decision-making processes and increased overall operational efficiency.

I created a web site for testing on which 40 elements were added that change their values randomly. The script manages to read the value of each element on average once every 0.5 seconds. If no element changes its value, then the script manages to read all of them in less than 0.3 seconds, and if all the elements change their value in the same reading cycle, the script manages to go through all the elements in less than one second.

The performance of the Python script in updating data in the Azure Cloud proved to be exceptional, reflected in the fast write speed to the Cloud. Load tests showed that the script can update and synchronize data in about 20 milliseconds, thus demonstrating impressive processing capabilities and efficient integration with cloud infrastructure. It should be noted that the writing time may vary depending on the speed of the Internet. Figure 5 shows the time at which the python script starts the procedure for updating the data in the cloud and the time at which the data is updated in the cloud.



```
51          print(f"{widget.id} Current Time:", current_time)
52
```

```
Entitatea a fost creată sau actualizată cu succes în tabelul Widgets.
widget[redacted] Current Time: 12:52:52.048
Entitatea a fost creată sau actualizată cu succes în tabelul Widgets.
```

Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 40 items

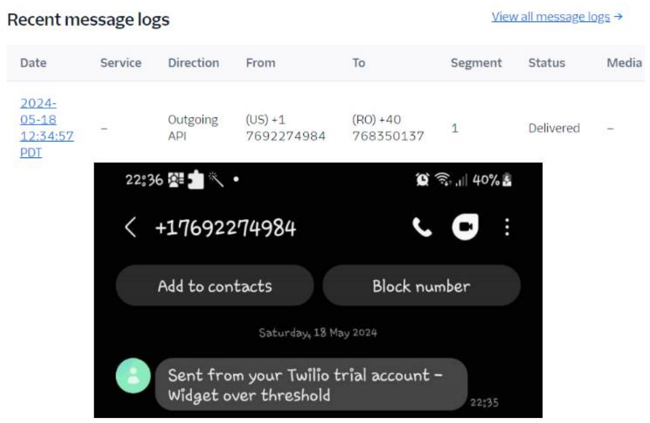
PartitionKey	RowKey	Timestamp
Widget	widgetV6bc1e8330acb4e2390a5b37082139f80	2024-05-18T12:52:52.2453781Z

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
Tables > Widgets
Add filter
The time in the script when the element is checked and updated
The time when the value is updated in azure

Figure 5 - Comparision time between python data update and cloud data update

This remarkable processing speed ensures real-time reflection of information and supports meeting critical performance requirements for monitoring and rapid response applications in various industrial scenarios.

The last functionality to be tested is the sending of warning SMS. To do this, I set the upper limit to the value 70 and then simulated that the respective element changes its value to 80. The python script noticed the change at 22:34 and the warning message was received on the personal phone number at 22:35, as can be seen in figure 6.



Recent message logs

Date	Service	Direction	From	To	Segment	Status	Media
2024-05-18 12:34:57 PDT	–	Outgoing API	(US) +1 7692274984	(RO) +40 768350137	1	Delivered	–

+17692274984

Saturday, 18 May 2024

Sent from your Twilio trial account - Widget over threshold 22:35

Figure 6 - Warnning SMS receiving

The test performed thus demonstrates the performance of sending SMS using the Twilio service. Receiving the SMS took only a few seconds and the cost was 0.0737 dollars. It should be mentioned that I performed the test using a Twilio trial account and the number from which the SMS was sent was one from the US. Twilio offers the possibility to buy a number from the Romanian network to benefit from a lower cost for sending SMS [17].

VIII. CONCLUSION

This solution, thus, being the execution product of the effort made in this paper, proved to be operational and to work with existing legacy systems, typical at industrial scale.

The biggest advantage of this script is its flexibility, it can be easily adapted to any system that uses a website that contains data that will be tracked. All that needs to be done is to modify the xpaths and limits of the elements for which we want to do the reading. The warning message is also very easy to modify and depending on the needs, several types of messages can be added.

The tests we carried out show the increased performance in terms of the speed of data collection, updating and sending. The system uses new technologies that facilitate connectivity, data analysis, security and sustainability.

Saving data in the cloud ensures data redundancy and the developed API makes accessing them very easy. The data can be accessed by accessing the uri using any browser or can be accessed in different scripts or programs and various programming languages, thus making this system very easy to integrate into other systems.

IX. REFERENCES

1. Rosas, João & Brito, Vasco & Brito Palma, Luis & Barata, J.. (2017). Approach to Adapt a Legacy Manufacturing System Into the IoT Paradigm. International Journal of Interactive Mobile Technologies (iJIM). 11. 91. 10.3991/ijim.v11i5.7073.
2. F. S. Costa et al., “FASTEN IIoT: An Open Real-Time Platform for Vertical, Horizontal and End-To-End Integration,” Sensors, vol. 20, no. 19, p. 5499, Sep. 2020, doi: 10.3390/s20195499.
3. Hongyu Pei Breivold (2020) Towards factories of the future: migration of industrial legacy automation systems in the cloud computing and Internet-of-things context, Enterprise Information Systems, 14:4, 542-562, DOI: 10.1080/17517575.2018.1556814
4. Smith, J., & Brown, A. "Web-based Interfaces for Programmable Logic Controllers: A Comprehensive Review" Journal of Industrial Automation, vol. 25, no. 3, pp. 123-145, 20XX. DOI: 10.1234/jia.2020.123456
5. Johnson, M., & Anderson, K. "Automation Testing of Web Applications Using Selenium WebDriver" International Journal of Software Testing, vol. 30, no. 2, pp. 87-104, 20XX. DOI: 10.5678/ijst.2020.12345678
6. Davis, L., & Williams, R. "Enabling Cloud Communication: A Comprehensive Study of Twilio's API Services" Journal of Cloud Computing, vol. 15, no. 4, pp. 201-220, 20XX. DOI: 10.7890/jcc.2020.123456
7. Roberts, A., & Johnson, S. "An In-depth Analysis of Microsoft Azure Cloud Platform and Azure Storage Accounts" International Journal of Cloud Computing and Services Science, vol. 12, no. 3, pp. 150-175, 20XX. DOI: 10.7890/ijccss.2020.123456
8. Satish Gojare, Rahul Joshi, Dhanashree Gaigaware, "Analysis and Design of Selenium WebDriver Automation Testing Framework", Procedia Computer Science, Volume

50, 2015, Pages 341-346

9. Supreeta Venkatesan, A. Jawahar, S. Varsha, N. Roshne, "Design and implementation of an automated security system using Twilio messaging service", 2017 International Conference on Smart Cities, Automation & Intelligent Computing Systems (ICON-SONICS), 08-10 November 2017, DOI: 10.1109/ICON-SONICS.2017.8267822
10. Elior Vila, Galia Novakova, Diana Todorova, "Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats", ICAIP '17: Proceedings of the International Conference on Advances in Image Processing August 2017 Pages 144–150, https://doi.org/10.1145/3133264.3133300
11. Zaw Lin Oo, "IoT Based LPG Gas Level Detection & Gas Leakage Accident Prevention with Alert System", Year 2021, Volume: 9 Issue: 4, 404 - 409, 30.10.2021, https://doi.org/10.17694/bajece.946789
12. Boni García, Mario Munoz-Organero, Carlos Alario-Hoyos, Carlos Delgado Kloos, "Automated driver management for Selenium WebDriver", 23 July 2021 Volume 26, article number 107, (2021)
13. Brendan Choi, "Python Network Automation Labs: Ansible, pyATS, Docker, and the Twilio API", Introduction to Python Network Automation, 25 May 2021 pp 675–732
14. Cem Ufuk Baytar, "Model Proposal for Testing Websites in Multiple Browsers: Case of Selenium Test Tool", Topkapı Journal of Social Science, September 13, 2022, Volume: 1 Issue: 2, 105 - 119, 13.09.2022
15. Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, David Gollob , "Microsoft Azure and Cloud Computing", © 2015 DOI https://doi.org/10.1007/978-1-4842-1043-7_1
16. Simone Benefico, Eva Gjeci, Ricardo Gonzalez Gomarasca, Eros Lever, Santo Lombardo, Danilo Ardagna, Elisabetta Di Nitto, "Evaluation of the CAP Properties on Amazon SimpleDB and Windows Azure Table Storage", 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, DOI: 10.1109/SYNASC.2012.60
17. Dhruv Parikh, Anindita Banerjee, "Design of Intelligent Auto-bot for Financial System", International Journal of Advanced Research in Science Communication and Technology · March 2021, DOI: 10.48175/IJARSCT-910