Rizki Arif Setiadi
Faculty of Information Technology
Master in Information Technology
Universitas Teknologi Digital Indonesia
Yogyakarta, Indonesia
email: student.rizkiarif@mti.utdi.ac.id

Bambang Purnomosidi DP[1]
Faculty of Information Technology
Master in Information Technology
Universitas Teknologi Digital Indonesia
Yogyakarta, Indonesia
email: bpdp@utdi.ac.id

Widyastuti Andriyani
Faculty of Information Technology
Master in Information Technology
Universitas Teknologi Digital Indonesia
Yogyakarta, Indonesia
emial: widya@utdi.ac.id

Sri Rezeki Candra Nursari
Faculty of Engineering
Informatics Engineering
Universitas Pancasila
email:
sri.rezeki.candra.n@univpancasila.ac.id

Microservices Architecture in Point of Sales Application Based on Restful API and Webhook

Microservices are a collection of small and inde- pendent processes that communicate with each other to create a complex application that is not dependent on a specific API language. In this study, the author seeks to implement Microservices Architecture, RESTful API, and Webhook in a Point of Sale application to achieve a system with superior performance, speed, and scalability, especially in the exchange and communication of data. This research combines two important testing methodolo- gies, namely unit testing and functional testing, to ensure the robustness and reliability of the Microservices-based Point of Sale application. Unit testing focuses on validating individual components and functions within the system, while functional testing assesses the overall functionality and behavior of the application. This testing approach aims to enhance the quality and reliability of the system. The findings of this reseach indicate that the implementation of Microservices Architecture, based on RESTful API and Webhook, has successfully improved the accuracy of data input in the Point of Sale application as evidenced by the calculations in the confusion matrix. Before the implementation of the webhook, the accuracy was only 80 percent, but after its implementation, the accuracy increased to 100 percent.

***KeyWords**: Microservices, Point of Sales, RESTful API, Webhook*

This Article was:
submited: 10-06-24
accepted: 24-06-24
publish on: 20-07-24

How to Cite:
F.A. Jhone, et al, "Template for JISS Journal Papers: JISS UTDI No More Then 12 Words", Journal of Intelligent Software Systems, Vol.3, No.1, 2024, pp.31–35, 10.26798/jiss.v3i1.1336

1 Introduction

In the ever-growing digital era, Point of Sale (POS) systems play an important role in retail and service business operations. This system is not only used to manage transactions and inventory, but also to optimize interactions with customers. However, traditional POS systems often face challenges, including limitations in terms of scale, complex integration, liability to change, and suboptimal performance. To overcome these challenges, more modern and efficient solutions are needed. To address these challenges, modern and efficient solutions are needed. This is where the concepts of Microservices architecture, RESTful API, and webhook come into play. Mi- croservices enable the development of modular POS systems, facilitating changes and maintenance of individual components without disrupting the entire system [1]. RESTful API allows smooth and standardized interactions between POS services and external services [2]. Webhooks serve as a method for notifying events through a push mechanism. In this setup, the client subscribes to the server for specific resources and awaits the server's response when updates become available. Unlike the traditional client- server arrangement, where the server cannot initiate commu- nication with the client, webhooks enable such initiation. As mentioned in [3], it is common among the push based methods to break this pattern and that is what the webhook server has to do.When the client subscribes it attaches a Uniform Resource Locator (URL) to the request, specifying where it wants to receive the update of the resource.The server then uses the URL when an update has occurred to notify the client. Putra, 2020 [4] researched slow and low-quality teaching apps, causing delays in developing the Home Pesantren app and inability to meet fast-changing needs. They used app performance data. The result: a new app with microservices for scalability and reliability. Meanwhile, Atmojo et al., 2022 [5] found manual data collection delaying reports and Pengalangan village system development. This led to wasted time and increased security risks. They also used app performance data. Their solution: a new village system with microservices for better security and service. Seviro Bima Sakti and Hermawan [6], 2020 studied traffic violations in Depok from 2017 to 2019. They found that as workload increases, the app's ability to manage and distribute data decreases, affecting police work. They used accident and distribution data. Their solution: an app with microservices for better data management and real-time distribution. Research conducted by Pamungkas, 2021 [7] found a lack of guidance for implementing enterprise architecture in microservices apps. This makes it hard for enterprises to adopt such architecture. Their study provides clearer guidance for this process. Therefore, this research aims to explore and implement the integration of Microservices, RESTful API, and webhooks in the context of POS systems. The goal is to enhance scalability, responsiveness, and performance in POS systems. Thus, this research will assist retail and service companies in overcoming challenges posed by the rapidly changing digital era. Additionally, webhooks serve as a push-based notification mechanism that allows POS systems to send information and events directly to clients. This improves system responsiveness, as clients do not need to poll periodically for updates. This research also adopts two main types of testing, namely unit testing and functional testing. Unit testing aims to validate each individual component in the system, while functional testing assesses the overall functionality and behavior of this microservices-based POS application. With this approach, this research aims to provide solutions that are more responsive, well integrated, and have better performance in facing changing business needs in the digital era.

[1]Corresponding Author.

2 Methodology

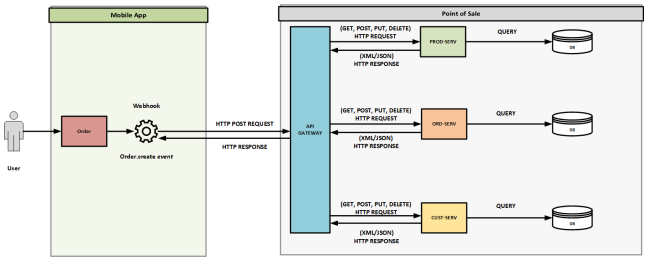

Fig. 1 System Diagram

2.1 System Diagram. The diagram above represents the overall system design scheme, consisting of:

(1) Mobile App (External System): This application serves as the primary point where users can create new orders. The mobile app has the capability to transform user operations (creating new orders) into an event, which is then sent via webhook to the Point of Sale Application using the HTTP POST method.
(2) Point of Sale: This application functions to process order data sent by the Mobile App (external system). The Point of Sale application implements a microservices architecture, as depicted in the diagram. There are four core components of this application:
   (a) API Gateway: Acts as the main access point accessible by external applications for data transmission.
   (b) PROD-SERV: Functions as a service managing product data.
   (c) ORD-SERV: Functions as a service managing order data.
   (d) CUST-SERV: Functions as a service managing customer data.

2.2 Microservices. In the past, applications were typically developed using monolithic methods, where a single codebase managed the entire application. Creating a monolithic application was considered simpler compared to more decentralized approaches. Running them on the other hand is not without challenges. Monolithic applications lack some cardinal features, namely scalability and flexibility [8] These issues in addition to the rise of the new technologies the software industry to search for alternatives. Microservices, which have recently become popular among Software Engineering practitioners, offer a slightly different approach. Applications are divided into small, specific-functioning (high cohesion) parts that do not depend on other program components (loose coupling), with an API (Application Programming Interface) interface [1] [8].Even though Internet and microservice security [9],performance [10,11],traceability [12], compatibility [13], complexity [14], effectiveness and scalability [15] have become a leading issue and there are noticeable privacy concerns. As a result, analysts require programming skills and explicit tools [16], framework [17], model [18] to test them.

2.2.1 RESTful API. REST was originally developed to explain the design of the Webs network protocols and client, server, and proxy software. Nor can it connect graphs of components without forcing complete mutual trust, since its only mechanism for composition is the linear proxy pipeline [2]. Currently, the development of a RESTful API isbased on frameworks and libraries. However, this approach does not prevent developers to wrongly use GET requests for updating resources,although it violates the safety requirement of HTTP [19].

2.2.2 Webhook. Webhooks serve as a method for notifying events through a push mechanism. This approach has become increasingly pop- ular and is now widely utilized in modern web applications, including major platforms like GitHub, Facebook, and Google. Unlike the conventional client-server model where updates are fetched through repeated requests, webhooks require a shift in this paradigm. As mentioned in [3], it is common among the push based methods to break this pattern and that is what the webhook server has to do.When the client subscribes it attaches a Uniform Resource Locator (URL) to the request, specifying where it wants to receive the update of the resource. This thesis adopts the understanding of webhooks as presented in [20–22], wherein the server functions as an API provider and the client assumes the role of a subscriber. The client initiates a subscription request for a service provided by the server, offering a webhook endpoint to receive notifications from the server upon event occurrence. As noted by Matthias Biehl [20], it's crucial to differentiate between a webhook and a webhook endpoint, where the latter represents the client-side implementation of the receiving endpoint. Webhook, on the other hand, refers to the concept of transmitting events to a webhook endpoint hosted by the client. Webhooks effectively address the delivery challenge from the server to the client and stand out as the predominant solution for managing push-based events between server and client, particularly in conjunction with RESTful APIs.

2.2.3 Testing and Evaluation. The types of testing to be conducted in this research are Unit Testing and Functional Testing. Unit testing has evolved into a widely embraced practice, frequently enforced by development methodologies such as test-driven development. Leveraging familiar testing tools, unit testing has morphed into a chameleon, capable of subtly blending in and assuming the guise of other testing types.As mentioned by Andrew Hunter, "Unit tests have quickly become the proverbial hammer that makes everything look like a nail" [23]. Meanwhile Black box testing is also called as functional testing, a functional testing technique that designs test cases based on the information from the specification [24].

2.3 Confusion matrix. Many different multi label classification algorithms have been developed for problems such as text categorization [19], multimedia content annotation [25], disease recognition [26], and web mining [27].A powerful method for analyzing a multi-class classifier is the 2-dimensional confusion matrix to show the distribution of false predictions in one view. The extracted metrics from the confusion matrix, such as precision,recall, and F score for each class and micro, macro, and weighted average of all classes, are used for measuring the overall performance of a classifier. However, in multi-label classification problems, the confusion matrix is an undefined method. In such problems the performance measurement is limited to calculating aggregate metrics [28] such as hamming loss, accuracy, subset accuracy, precision, recall, and F score (F-score).

3 Result and Discussions

3.1 Unit Testing. The testing is conducted to ensure that each controller available in every service, webhook and API Gateway can process the requests properly.

Table 1 Unit Test Results

No	Service	Test Case	Result
1	Prod-serv	Index	Pass
2	Prod-serv	show with valid product id	Pass
3	Prod-serv	show with invalid product id	Pass
4	Prod-serv	Store	Pass
5	Prod-serv	Update	Pass
6	Prod-serv	Destroy	Pass
7	Ord-serv	Index	Pass
8	Ord-serv	Create with valid data	Pass
9	Ord-serv	Create with invalid data	Pass
10	Cust-serv	Index	Pass
11	Cust-serv	Add new customer with valid data	Pass
12	Cust-serv	Add new customer invalid data	Pass
13	Cust-serv	Show	Pass
14	Webhook	Webhook request with valid data	Pass
15	Webhook	Webhook request with invalid data	Pass
16	API Gateway	Index	Pass
17	API Gateway	show with valid product id	Pass
18	API Gateway	show with invalid product id	Pass
19	API Gateway	Store	Pass
20	API Gateway	Update	Pass
21	API Gateway	Create order with valid data	Pass
22	API Gateway	Create order with invalid data	Pass
23	API Gateway	Add new customer with valid data	Pass
24	API Gateway	Add new customer with invalid data	Pass
25	API Gateway	Show list of customer	Pass

Based on the Table 1, all test cases for the various services and API Endpoint, including Prod-serv, Ord-serv, and Cust-serv, were executed successfully, with each test producing a 'Pass' result, indicating that the controllers and API Endpoint functioned as intended.

3.2 Functional Testing. Functional testing of RESTful API involves testing end- points for various scenarios, including creating, updating, retrieving, and deleting data.

Table 2 Functional Test Results

No	Service	Test Case	Test Case	Result
1	Ord-serv	POST	Create with valid data	Pass
2	Ord-serv	POST	Create with invalid data	Pass
3	Prod-serv	GET	Index	Pass
4	Prod-serv	POST	Create with valid data	Pass
5	Prod-serv	POST	Create with invalid data	Pass
6	Prod-serv	GET	Show with valid id	Pass
7	Prod-serv	PUT	Update	Pass
8	Prod-serv	DELETE	Delete	Pass

Referring to the Table 2, every test case for the diverse services and API endpoints, such as Prod-serv, Ord-serv has been executed successfully. Each test resulted in a 'Pass,' affirming that the API endpoints operated as intended.

3.3 Data. In this section, we present the outcomes of our study based on a dataset comprising 30 samples. The data, which simulates manual and automatic processes for a given task, is structured into four columns: 'Customer name,' 'Manual Input,' 'Automatic (Webhook),' and 'Hasil' (Result).

Table 3 Data Sample

Customer	Manual Input	Webhook	Result
Andre	Pass	Pass	Match
Dodi	Pass	Pass	Match
Candra	Fail	Pass	Not Match
Reni	Pass	Pass	Match
Riko	Pass	Pass	Match
Aldi	Pass	Pass	Match
Taher	Pass	Pass	Match
Luna	Pass	Pass	Match
Maya	Pass	Pass	Match
Eko	Pass	Pass	Match
Arfi	Pass	Pass	Match
Tiwi	Fail	Pass	Not Match
Nisha	Pass	Pass	Match
Tuti	Pass	Pass	Match
Robin	Pass	Pass	Match
Van	Fail	Pass	Not Match
Cipto	Pass	Pass	Match
Fian	Pass	Pass	Match
Rama	Pass	Pass	Match
Arfa	Pass	Pass	Match
Didu	Pass	Pass	Match
Pras	Fail	Pass	Not Match
Tigor	Pass	Pass	Match
Rehan	Pass	Pass	Match
Dave	Pass	Pass	Match
David	Pass	Pass	Match
Amy	Pass	Pass	Match
Niko	Pass	Pass	Match
Bayu	Pass	Pass	Match
Bagas	Pass	Pass	Match

Then, the comparison data will be further assessed for accuracy using a confusion matrix.

Table 4 Confusion Matrix for Manual Input

	Predicted Match	Predicted Not Match
Actual Match	21	3
Actual Not Match	3	3

Based on the data in Table 4, the accuracy can be calculated as follows:

$$\text{Accuracy} = \frac{TP+TN}{\text{Total}} = \frac{21+3}{30} = \frac{24}{30} = 0.8(80.0\%) \quad (1)$$

(1) True Positive (TP) means the number of data obtained based on cases where manual input = pass and result = match, which is a total of 21 data.
(2) False Positive (FP) means that the number of data obtained based on cases where manual input = fail and result = match is a total of 3 data.
(3) True Negative (TN) means that the number of data obtained based on cases where manual input = fail and result = not match is a total of 3 data.
(4) False Negative (FN) means that the number of data obtained based on cases where manual input = pass and result = not match is a total of 3 data.

Table 5 Confusion Matrix for Webhook

	Predicted Match	Predicted Not Match
Actual Match	24	0
Actual Not Match	0	6

Based on the data in Table 5, the accuracy can be calculated as follows:

(1) True Positive (TP) means the number of data obtained based on cases where manual input = pass and result = match, which is a total of 24 data.
(2) False Positive (FP) means that the number of data obtained based on cases where manual input = fail and result = match is a total of 0 data.
(3) True Negative (TN) means that the number of data obtained based on cases where manual input = fail and result = not match is a total of 6 data.
(4) False Negative (FN) means that the number of data obtained based on cases where manual input = pass and result = not match is a total of 0 data.

The calculation results between manual input and automatic input can also be used as an indicator of the system's success in accumulating data. Manual input with an accuracy of 80 percent potential to cause losses for business owners, especially in sales or stock reports, while the implementation of webhook with 100 percent accuracy greatly benefits business owners as it can provide accurate reports.

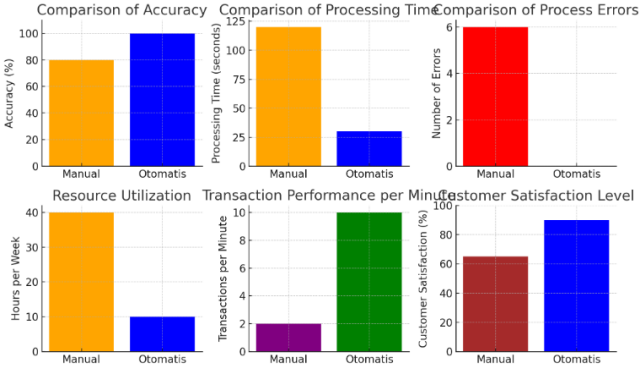

Fig. 2 Compare the performance of manual and automated processes across various metrics.

Figure 2 illustrates the superior performance of automated processes over manual processes in terms of accuracy, processing time, error rates, resource utilization, transaction efficiency, and customer satisfaction. Automated processes not only increase efficiency and accuracy but also improve customer experience and reduce labor hours, making it a highly profitable approach

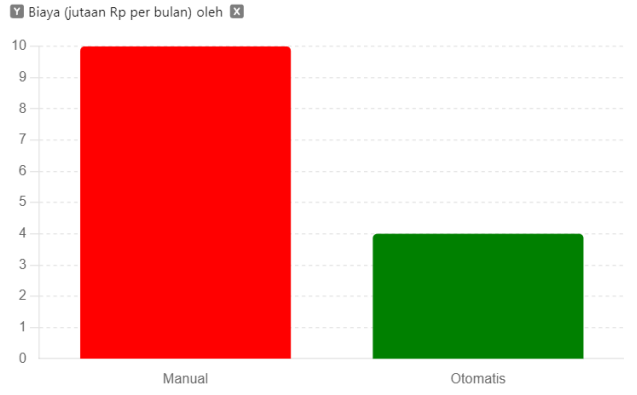

Fig. 3 Operational cost savings between manual and automated processes

The graph on Figure 3 shows the operational cost savings between manual and automated processes. The vertical axis (Y) displays costs in millions of Rupiah per month, while the horizontal axis (X) displays two methods, namely manual and automatic. From the graph, it can be seen that operational costs for manual processes reach around 10 million Rupiah per month, which is surrounded by a red line. Unfortunately the operational costs for this automatic process are much lower, namely around 4 million Rupiah per month surrounded by green bars. In other words, automated processes provide significant cost savings over manual processes, reducing monthly costs by almost half. This shows that automation not only increases efficiency and accuracy, but is also very effective in reducing overall operational costs

Fig. 4 Performance between manual processes and automatic processes

Compared the performance between manual processes (represented by orange bars) and automated processes (represented by yellow bars) on several key metrics, namely: accuracy, processing time, number of errors, resource usage, customer satisfaction, and monthly operational costs. It can be seen in terms of accuracy, the automatic process reaches 100%, while the manual process only reaches around 80%. This shows that automation provides more accurate results. Completion time is also more efficient in the automatic process, which only takes around 30 seconds compared to 120 seconds in the manual process. The number of errors that occurred in the manual process was 6, while the automatic process produced no errors, this shows the superiority of automation in reducing operational errors. Resource usage in terms of working hours per week is also lower in the automated process, requiring only 10 hours per week compared to 40 hours per week in the manual process. Customer satisfaction is higher with automated processes, with a satisfaction rate of around 90% compared to 65% for manual processes. Finally, in terms of operational costs, the automatic process requires lower costs, only around 4 million Rupiah per month, while the manual process costs around 10 million Rupiah per month. The graph shows that automation not only increases efficiency and accuracy, but also reduces errors, increases customer satisfaction, optimizes resource use, and saves operational costs significantly compared to manual processes.

4 Conclusions and Recommendations

4.1 Conclusions. This research successfully demonstrates that the imple- mentation of Microservices architecture, RESTful API, and webhook concepts has effectively overcome the challenges faced by traditional POS systems in the retail and service industries. The research results consistently show a significant improvement in scalability, integration, and performance of POS systems. This is evidenced by the microservices archi- tecture's ability to handle high workloads and provide a better customer experience, as each service only performs one task, resulting in faster system performance and enabling module expansion or integration with other external applications such as inventory management or

business analytics due to its RESTful API-based nature. The calculation results between manual input and automatic input can also be used as an indicator of the system's success in accumulating data. Manual input with an accuracy of 80 percent has the potential to cause losses for business owners, especially in sales or stock reports, while the implementation of webhook with 100 percent accuracy greatly benefits busi- ness owners as it can provide accurate reports. This research provides strong evidence that retail and service companies should seriously consider the implementation of Microservices architecture, RESTful API, and webhook, concepts in their POS systems as an effective solution to remain competitive in rapidly changing markets. The integration of this technology brings forth flexibility, modularity, and extraordinary adaptability, substantially assisting companies in overcoming the challenges of the current digital era.

4.2 Recommendations. Based on the limitations of the webhook event types described in the problem constraints, it is expected that the point of sale system can be further developed to support event types for order status updates and order cancellations. Supporting data/product synchronization with external applications to make data exchange more seamless.

References

[1] Namiot, D. and Sneps-Sneppe, M., 2014, "On micro-services architecture," International Journal of Open Information Technologies, **2**(9).

[2] Khare, R. and Taylor, R., 2004, "Extending the Representational State Transfer (REST) architectural style for decentralized systems," *Proceedings of the 26th International Conference on Software Engineering*, Edinburgh, UK, pp. 428–437.

[3] Bozdag, E., Mesbah, A., and van Deursen, A., 2007, "A comparison of push and pull techniques for ajax," *2007 9th IEEE International Workshop on Web Site Evolution*, pp. 15–22, doi: 10.1109/WSE.2007.4380239.

[4] Putra, Y. C. T., 2020, "Implementasi Arsitektur Microservice pada Aplikasi Web Pengajaran Agama Islam Home Pesantren," Jurnal Informatika Atma Jogja, **1**(1), pp. 88–97.

[5] Atmojo, S., Utami, R., Dewi, S., and Widhiyanta, N., 2022, "Implementasi Sistem informasi Desa Berbasis Arsitektur Microservices," Smatika Jurnal, **12**(01), pp. 55–66.

[6] Sakti, C. S. B. and Hermawan, I., 2020, "Implementasi Arsitektur Microservice pada Back End Sistem Informasi Atlantas berbasis Website," Jurnal Teknologi Terpadu, **6**(2), pp. 96–104.

[7] Pamungkas, N. Y., 2021, "Implementasi Arsitektur Enterprise Pola Finansial pada Aplikasi Berbasis Microservices," .

[8] Jaramillo, D., Nguyen, D. V., and Smart, R., 2016, "Leveraging microservices architecture by using Docker technology," *SoutheastCon*.

[9] Yu, D., Jin, Y., Zhang, Y., and Zheng, X., 2017, "A survey on security issues in services communication of Microservices-enabled fog applications," Concurrency and Computation, pp. 1–19.

[10] Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M. K., and Sekar, V., 2016, "Gremlin: Systematic Resilience Testing of Microservices," *Proceedings of the International Conference on Distributed Computing Systems*, Vol. 2016-August, pp. 57–66.

[11] de Camargo, A., Salvadori, I., dos S. Mello, R., and Siqueira, F., 2016, "An architecture to automate performance tests on microservices," *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, pp. 422–429.

[12] Carrasco, A., van Bladel, B., and Demeyer, S., 2018, "Migrating towards microservices: migration and architecture smells," *Proceedings of the 2nd International Workshop on Refactoring*, pp. 1–6.

[13] TOGAF, 2016, *Microservices Architecture*.

[14] Taibi, D., Lenarduzzi, V., Pahl, C., and Janes, A., 2017, "Microservices in agile software development: a workshop based study into issues, advantages, and disadvantages," *Proceedings of XP2017 Scientific Workshops*, Vol. XP '17, pp. 1–5.

[15] Cerny, T., Donahoo, M. J., and Trnka, M., 2018, "Contextual understanding of microservice architecture," ACM SIGAPP Applied Computing Review, **17**(4), pp. 29–45.

[16] Soldani, J., Tamburri, D. A., and Heuvel, W. J. V. D., 2018, "The pains and gains of microservices: A Systematic grey literature review," Journal of Systems and Software, **146**, pp. 215–232.

[17] Chaturvedi, A. and Gupta, A., 2013, "A tool supported approach to perform efficient regression testing of web services," *2013 IEEE 7th International Symposium on Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, pp. 50–55.

[18] Ma, S. P., Fan, C. Y., Chuang, Y., Lee, W. T., Lee, S. J., and Hsueh, N. L., 2018, "Using Service Dependency Graph to Analyze and Test Microservices," *Proceedings of the International Computer Software and Applications Conference*, Vol. 2, pp. 81–86.

[19] Altınel, B. and Ganiz, M. C., 2018, "Semantic text classification: A survey of past and recent advances," Information Processing Management, **54**(6), pp. 1129–1153.

[20] Biehl, M., 2017, *Webhooks – Events for RESTful APIs*, API-University Press.

[21] Zapier, "What are webhooks?" https://zapier.com/blog/what-are-webhooks/

[22] Shakhovska, N., Basystiuk, O., and Shakhovska, K., 2019, "Development of the speech-to-text chatbot interface based on google api," *MoMLeT*.

[23] Hunter, A., 2012, "Are unit test overused," https://www.simple-talk.com/dotnet/netframework/are-unit-tests-overused/

[24] Liu, H. and Tan, H. B. K., 2009, "Covering code behavior on input validation in functional testing," Information and Software Technology, **51**(2), pp. 546–553.

[25] Li, Z., Fan, Y., Jiang, B., Lei, T., and Liu, W., 2019, "A survey on sentiment analysis and opinion mining for social multimedia," Multimedia Tools and Applications, **78**(6), pp. 6939–6967.

[26] Fatima, M. and Pasha, M., 2017, "Survey of machine learning algorithms for disease diagnostic," Journal of Intelligent Learning Systems and Applications, **9**(1), pp. 1–16.

[27] Martinez-Rodriguez, J. L., Hogan, A., and Lopez-Arevalo, I., 2020, "Information extraction meets the semantic web: A survey," Semantic Web, **11**(2), pp. 255–335.

[28] Zhang, M.-L. and Zhou, Z.-H., 2014, "A review on multi-label learning algorithms," IEEE Transactions on Knowledge and Data Engineering, **26**(8), pp. 1819–1837.

[29] Fielding, R. T., 2008, "REST APIs must be hypertext-driven," http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[30] Camargo, A. D., Salvadori, I., Mello, R., Dos, S., and Siqueira, F., 2016, "An architecture to automate performance tests on microservices," *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*.