JINAV: Journal of Information and Visualization *ISSN: 2746-1440 (online)*
Vol. 5 No. 2 (2024)
https://doi.org/10.35877/454RI.jinav3041

Performance Analysis of API Protocol Models as Recommendations for Developers in Application Development

Rizky Putra Fhonna[a,*], Yesy Afrillia[b], Veri Ilhadi[a], Abdul Halim Arif[a], & Riko Ardiansyah Selian[a]

[a]*Information System Department, Universitas Malikussaleh, Indonesia*
[b]*Informatics Department, Universitas Malikussaleh, Indonesia*

Abstract

The evaluation of various API types reveals distinct strengths and weaknesses. REST APIs exhibit inefficient performance with high average response times and an error rate of approximately 14%, indicating potential delays and instability under load. SOAP APIs, with an average response time of 167 ms, perform better than REST in terms of speed but still lag behind GraphQL and have a slightly higher error rate of 14.80%. GraphQL demonstrates the fastest average response time at around 171 ms, offering high efficiency in data delivery, although its error rate is notably high at 15%, signaling a need for improved stability. RPC APIs, with an average response time of 238 ms, are less speedy compared to GraphQL and SOAP but excel in stability with a very low or zero error rate, making them highly reliable under high loads. Overall, GraphQL is optimal for applications requiring rapid data interaction, RPC is best suited for scenarios demanding high consistency and reliability, SOAP offers a middle ground, and REST may be appropriate for simpler, less demanding applications.

Keywords: API; load test; REST; GraphQL; RPC; SOAP

Received: 17 October 2024 Revised: 10 December 2024 Accepted: 19 December 2024

1. Introduction

With the rapid advancement of information technology, the demand for fast, reliable, and efficient application services is increasing (Fhonna & Fadli, 2022). APIs (Application Programming Interfaces) play a crucial role in meeting this demand by providing protocols, tools, and definitions that enable seamless communication between various applications (Suwarno & Yulandi, 2023). The choice of API protocol and programming language significantly impacts the performance and reliability of services (Dinegoro & Winengko, 2020). Previous studies have shown that REST API applications developed with Go exhibit high stability, while C# and Java also offer good performance (Dinegoro & Winengko, 2020). Go (Golang) is known for its high performance, concurrency capabilities, and stability.

This study aims to analyze the performance of three main API protocol models REST, SOAP, and RPC through simulations of key operations such as Create, Read, Update, Delete, Sorting, and Searching. The analysis seeks to understand the strengths and weaknesses of each protocol and their impact on overall application performance (Rak, 2023). In the context of microservices architecture, which allows for dividing a system into independent modules that communicate through APIs, selecting the right API protocol is crucial for optimizing the entire system (Sari, Kurniawati, & Muriyanto, 2021). The results of this study are expected to provide comprehensive guidance for application developers in choosing the most suitable API protocol for their specific feature requirements, in line with modern application development standards (Pontarolli, Bigheti, de Sá, & Godoy, 2023).

2. Research Methods

In the initial stage, the researcher conducted a literature review to explore theories related to APIs and API protocols, as well as tools such as load testing and stress testing (Abdullah & Daud, 2023). The purpose of this review was to build a conceptual foundation and gather relevant scientific references. Next, the researcher developed a backend

* Corresponding author.
E-mail address: rizkyputrafhonna@unimal.ac.id

application using the Echo framework with the Go programming language, which produced API endpoints for various operations such as creating, reading, updating, deleting, sorting, and searching data (Effendy & Adhilaksono, 2021). This stage aimed to create an application capable of simulating the processes to be tested (Abdullah & Daud, 2023). The following are some of the related theories:

a. API (Application Programming Interface)

An API is a set of rules, protocols, and tools that allow various software applications to interact with each other (Woody, Burdick, Lapp, & Huang, 2020). APIs enable different systems or applications to communicate and exchange information, utilize services, or use functionalities provided by other applications or platforms (Wu, Jing, Zhang, Kong, Xie, & Huang, 2020). They are often used in software development to integrate or connect different components within a larger system(González-Mora, Garrigós, Zubcoff, & Mazón, 2020).

b. API Protocol

An API protocol is a set of rules, formats, and specifications used by applications to communicate with each other through an API (Application Programming Interface). These protocols define how data is sent, received, and processed between different applications or systems (Belouin, Chen, & Wang, 2021). API protocols ensure that interactions between applications proceed correctly, allowing them to understand each other's data structures, the messages sent, and how to access and use the services or functions provided (Fhonna, Afrillia, Ilhadi, Aqmal, & Afwan, 2022). Common examples of API protocols include HTTP (Hypertext Transfer Protocol) for web communication and REST (Representational State Transfer) for designing stateless web APIs (Saif, Lung, & Matrawy, 2021).

c. REST API

REST API (Representational State Transfer Application Programming Interface) is an architectural approach used in developing web services to communicate with other software via the HTTP protocol. This approach focuses on simplicity and scalability in designing distributed systems (Kemer & Samli, 2023).

d. SOAP API

SOAP (Simple Object Access Protocol) is a widely used protocol for message exchange in distributed computing environments. It provides a standardized way for applications to communicate over a network, regardless of the underlying platform or programming language. SOAP is designed to support interoperability between different systems by using a common XML-based messaging format, which allows different applications to exchange information in a structured and platform-independent manner. This makes SOAP particularly suitable for complex, enterprise-level applications where reliability, security, and transaction integrity are critical (Cinci, Cerasi, & Gultekin, 2022).

As an XML-based protocol, SOAP encapsulates messages in a specific format that includes a header and a body, ensuring that both sender and receiver understand the message content and structure. This format allows for extensive flexibility and customization, enabling support for various standards such as WS-Security for secure message exchanges (Tiwary, Stroulia, & Srivastava, 2021). Despite its robustness and flexibility, SOAP is often considered more heavyweight compared to other protocols like REST due to its reliance on XML and the additional overhead of processing SOAP envelopes. However, its built-in support for security, transactions, and complex operations makes it an ideal choice for scenarios that require a high level of reliability and security in distributed environments (Syed, 2021).

e. GraphQL API

GraphQL is a query language introduced by Facebook in 2015. Unlike REST (Representational State Transfer), which requires the client to request data in a predefined format from the server, GraphQL provides clients the flexibility to specify precisely what data they need. With GraphQL, clients can create queries that define the desired data structure, and the server will respond with data that matches the query (Lee, Kwon, & Yun, 2020). This allows developers to retrieve only the required data without loading unnecessary information, reducing the number of network requests, and improving communication efficiency between the client and server.

With GraphQL, clients can send a query that precisely specifies the shape and structure of the data they want to retrieve, enabling a high degree of flexibility and control over data consumption. The server processes these queries and returns the data in the exact format requested by the client, which simplifies the development process by decoupling the client-side data needs from server-side implementations. This adaptability is beneficial for applications that require dynamic interactions or have complex data needs, such as mobile apps or single-page applications (SPAs), where loading speed and data optimization are crucial. Moreover, GraphQL can combine multiple requests into a single query, further

reducing the number of network calls and improving overall application performance (Vadlamani, Emdon, Arts, & Baysal, 2021).

f. JMetere

JMeter is an open-source software developed by Apache, used for performance testing, load testing, and functional testing of applications. It is a powerful tool used to measure the performance of various types of applications, protocols, and servers (Tiwari, Upadhyay, Goswami, & Agrawal, 2023). The API will be tested by simulating API execution with scenarios that align with the predetermined testing methods and subsequently obtaining the performance data of the API to be evaluated (Hasan & Muhammad, 2020).

In the next stage, testing is conducted using JMeter to evaluate the performance of the three API protocols: REST, SOAP, and RPC. This testing involves stress testing, load testing, and performance testing to measure aspects such as response time, throughput, concurrency, latency, and error handling (Indrianto, 2023). The test data is analyzed using statistical methods and graphical representations to compare the performance of each protocol (Mahajan, Attar, & Kalamkar, 2022). Finally, the researcher concludes the findings and results of the study, summarizing the problem statement, the significance of the findings, limitations, suggestions, and practical implications to provide recommendations for application development. The research flowchart illustrates the process from the literature review to system testing and result analysis, ensuring the system functions according to specifications. Figure 1 is the flowchart of the study.

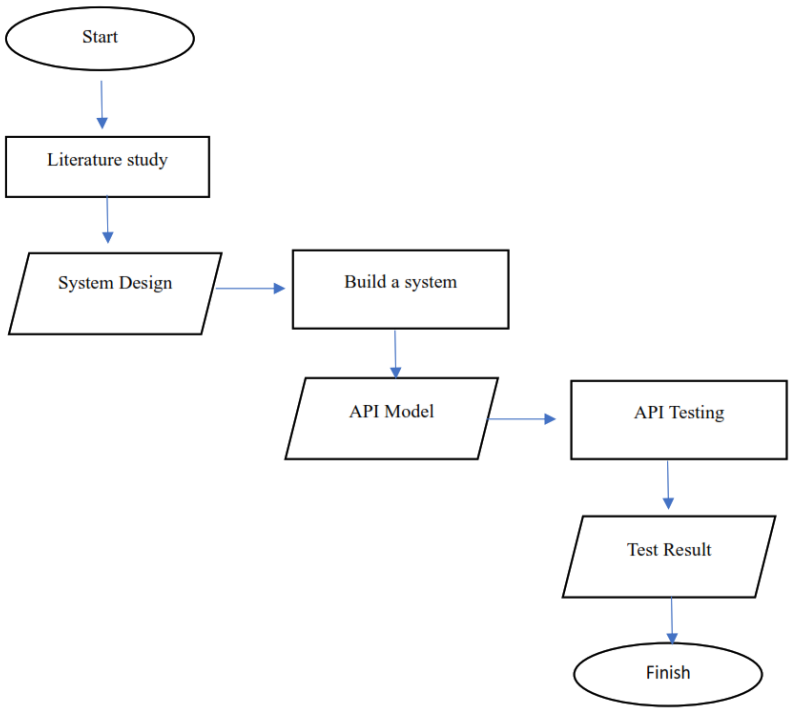

Figure 1. Research flow

3. Results and Discussions

3.1. Application Development Design

After conducting an in-depth literature review on various protocol models for server API development, including REST, SOAP, GraphQL, and RPC, the next step is to implement a server as the test object. This server will be built following the methodology and design planned in the research framework. The development process involves not only selecting the appropriate technology but also ensuring that the server can meet all the predefined criteria and testing scenarios.

Thus, the server built will serve as a valid platform for conducting comparative testing and performance evaluation according to the research objectives.

3.2. Testing Environment

Once the four API protocol models (REST, SOAP, GraphQL, and RPC) are developed, the next step is to set up an appropriate testing environment. This environment is crucial to ensure that each protocol model can be tested consistently and comprehensively. The research will be conducted in an online server environment using a provided hosting service.

3.3. Analysis and Comparative Results

The analysis results will address the research problem, which is the main objective of this study. The table below summarizes several points from the server API speed test, including both load and stress tests conducted on the four API architecture models: REST, SOAP, GraphQL, and RPC. The AVG value (in milliseconds) is obtained from the load test results, and the Error percentage (%) is derived from the stress test, serving as a benchmark for performance comparison among the models.

a. Read Simulation

Table 1. Read simulation result

	Read Simulation			
Model	REST API	SOAP API	GraphQL	RPC
Avg (ms)	337	160	155	236
Error (%)	16.40	10.20	9.80	0.40

Based on the results of the API read simulation test, the performance of several types of APIs can be compared. The REST API showed an average response time of 337 ms with an error rate of 16.40%, indicating suboptimal performance under high load conditions. In contrast, the SOAP API had an average response time of 160 ms and an error rate of 10.20%, demonstrating better performance compared to the REST API. GraphQL exhibited the best performance, with an average response time of 155 ms and an error rate of 9.80%, making it an efficient and reliable choice. Finally, RPC had an average response time of 236 ms and a very low error rate of 0.40%, showing good stability under high load conditions, even though its average response time was higher than SOAP and GraphQL.

b. Create simulation

Table 2. Create simulation result

	Create Simulation			
Model	REST API	SOAP API	GraphQL	RPC
Avg (ms)	315	236	155	247
Error (%)	8.40	9.80	14.80	0.00

Based on the results of the API create simulation test, the REST API recorded an average response time of 315 ms with an error rate of 8.40%, indicating decent performance but with room for improvement. The SOAP API was more efficient, with an average response time of 236 ms, though it had a slightly higher error rate of 9.80%. GraphQL demonstrated the fastest average response time of 155 ms but had a relatively high error rate of 14.80%, suggesting that while it is fast, its stability needs enhancement.

RPC showed the most stable performance, with an average response time of 247 ms and no errors (0.00%). Although RPC's response time was higher than that of GraphQL, the absence of errors indicates excellent reliability. These test results suggest that SOAP API and GraphQL offer better speed compared to REST API, albeit with some error-related challenges, while RPC provides the highest stability, despite not being as fast as GraphQL. The choice of API depends on the priority between speed and stability according to the application's needs.

c. Update simulation

Based on the results of the API update simulation test, the REST API demonstrated an average response time of 368 ms with an error rate of 14.40%. This indicates that the REST API has a relatively high average response time and a

significant error rate, reflecting suboptimal performance under high load. In contrast, the SOAP API showed better performance, with an average response time of 156 ms and an error rate of 9.60%, indicating a lower response time and fewer errors compared to the REST API.

Table 3. Update simulation result

	Update Simulation			
Model	REST API	SOAP API	GraphQL	RPC
Avg (ms)	368	156	156	245
Error (%)	14.40	9.60	14	0.20

GraphQL also exhibited a fast average response time of 156 ms but had a relatively high error rate of 14%. This suggests that while GraphQL is efficient in terms of speed, its stability still needs improvement. Meanwhile, RPC showed stable performance with an average response time of 245 ms and a very low error rate of only 0.20%. Although its average response time is higher than that of SOAP and GraphQL, the very low error rate indicates that RPC is highly reliable and stable under high load conditions.

d. Delete simulation

Table 4. Delete simulation result

	Delete Simulation			
Model	REST API	SOAP API	GraphQL	RPC
Avg (ms)	214	170	154	237
Error (%)	10	13.40	13.40	0.20

Based on the results of the API delete simulation test, the REST API demonstrated an average response time of 214 ms with an error rate of 10%. This indicates reasonably good performance with an acceptable response time and moderate error rate, suggesting adequate stability. On the other hand, the SOAP API showed a faster average response time of 170 ms but had a higher error rate of 13.40%, indicating that while it is faster, its stability still requires improvement.

GraphQL had the fastest response time at an average of 154 ms; however, since the error rate is not mentioned, it is difficult to assess its overall stability. Meanwhile, RPC showed very stable performance with an average response time of 237 ms and a very low error rate of just 0.20%. Although its response time is higher than that of SOAP and GraphQL, the very low error rate suggests that RPC is highly reliable and stable in data deletion scenarios.

e. Sorting simulation

Table 5. Sorting simulation result

	Sorting Simulation			
Model	REST API	SOAP API	GraphQL	RPC
Avg (ms)	216	172	153	267
Error (%)	8.40	13.80	13.80	0.00

Based on the results of the API sorting simulation test, the REST API demonstrated an average response time of 216 ms with an error rate of 8.40%. This indicates reasonably good performance with an acceptable response time and a moderate error rate, suggesting adequate stability. The SOAP API showed a faster average response time of 172 ms but had a higher error rate of 13.80%.

GraphQL exhibited the fastest response time with an average of 153 ms, but it also had a relatively high error rate of 13.80%. This indicates that while GraphQL is efficient in terms of speed, its stability still requires improvement. RPC displayed very stable performance with an average response time of 267 ms and no errors (0.00%). Although its response time is higher compared to SOAP and GraphQL, the absence of errors suggests that RPC is highly reliable and stable in sorting scenarios.

f. Searching simulation

Based on the results of the API searching simulation test, the REST API demonstrated an average response time of 315 ms with an error rate of 14%. This indicates that the REST API has a relatively high response time and a significant error rate, showing suboptimal performance under high load. The SOAP API performed better with an average response time of 167 ms, though it had a slightly higher error rate of 14.80%, indicating that while SOAP API is faster, it still has stability issues.

Table 6. Searching simulation result

Searching Simulation				
Model	REST API	SOAP API	GraphQL	RPC
Avg (ms)	315	167	171	238
Error (%)	14	14.80	15	0.00

GraphQL achieved a fast average response time of 171 ms, but it also had a relatively high error rate of 15%. RPC showed very stable performance with an average response time of 238 ms and no errors (0.00%). Despite its higher response time compared to SOAP and GraphQL, the absence of errors indicates that RPC is highly reliable and stable in searching scenarios.

In this section, we discuss the results obtained from the research experiments, focusing on the comparison and insights derived from the data. The study aimed to evaluate the performance of various API protocols—REST, SOAP, GraphQL, and RPC—across different operations such as create, update, delete, sorting, and searching. The results indicate that each protocol has its strengths and weaknesses.

GraphQL proved to be highly efficient with the fastest response times in most simulations, making it a strong choice for applications requiring rapid data interaction. However, despite its high speed, GraphQL also showed higher error rates, suggesting that stability and error handling need improvement for more consistent performance. On the other hand, SOAP demonstrated good performance with a balance between speed and stability. Although its response times were slightly slower compared to GraphQL, SOAP had lower error rates, indicating better reliability in the operations performed.

REST API, meanwhile, exhibited higher response times and significant error rates, especially under heavy load. This suggests that REST might not be the best choice for applications requiring high performance and stability. However, the simplicity and flexibility of REST make it suitable for applications with lighter loads. Conversely, RPC showed the most stable performance with very low error rates across simulations, though its response times were not as fast as other protocols. The reliability of RPC makes it ideal for applications requiring high data integrity and stability, even if it sacrifices some speed.

The comparison of measured data with theoretical models confirms the strengths and weaknesses of each protocol. The high speed of GraphQL supports its efficiency in dynamic data interactions, while the low error rates of RPC confirm its reliability. The balanced performance of SOAP reflects its design focus on resilience and security. These findings highlight the trade-offs between speed and stability and the importance of effective error handling. Future research could explore ways to enhance GraphQL's stability or improve RPC's speed. Integrating recent metrics that consider security and scalability could provide a more comprehensive assessment of API performance. These insights offer practical guidance for developers in selecting the API protocol best suited to their application's needs and priorities.

4. Conclusion

The performance analysis of various API protocols, including REST, SOAP, GraphQL, and RPC, reveals significant differences in terms of response speed, stability, and error rates. GraphQL stands out with the fastest response times in most simulations, making it highly efficient for applications that require rapid and dynamic data interactions, such as e-commerce and social media platforms. Despite its speed advantage, GraphQL exhibits a relatively high error rate, indicating a need for improvements in stability and reliability.

Conversely, RPC provides excellent stability and reliability with a very low error rate, although its response times are slower compared to GraphQL and SOAP. This makes RPC an ideal choice for applications requiring high consistency in data communication, such as backend systems for banking applications or Enterprise Resource Planning (ERP) systems. While RPC's response time is slightly higher, its stability and reliability are highly valuable for applications with critical transactional needs.

SOAP APIs demonstrate better performance than REST in terms of speed and security. With faster response times and support for robust security and transactional features, SOAP is a suitable choice for applications requiring compliance with high regulations and standards, such as healthcare management systems or integration with formal web services.

In contrast, REST APIs exhibit the slowest response times and a relatively high error rate under heavy load, making them less optimal for applications requiring fast responses or high stability. REST APIs are more suitable for

applications with simple architectures and light to moderate workloads, such as blogging platforms or news websites, where simplicity and flexibility are the primary priorities.

References

Abdullah, D., & Daud, M. (2023). Pengembangan sistem e-learning Politeknik Negeri Lhokseumawe dengan model VARK. *Jurnal Informasi dan Teknologi*, 222–228.

Belouin, P., Chen, S.-P., & Wang, S. (2021). Designing an API-based protocol for the interoperability of textual resources. *Digital Studies/Le champ numérique, 11*(1), 1–17.

Brito, G., & Valente, M. T. (2020). REST vs GraphQL: A controlled experiment. In *2020 IEEE International Conference on Software Architecture (ICSA)* (pp. 81–91). IEEE.

Cinci, M., Cerasi, C. C., & Gultekin, M. (2022). Token based novel approach to web service security. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)* (pp. 1–6). IEEE.

Dinegoro, A., & Winengko, M. (2020). *To close or not to close: Assessing the impact of open API to the bank performance in Indonesia*. Buletin Riset Kebijakan Perbankan, 2(1), 91–113.

Effendy, F., & Adhilaksono, B. (2021). Performance comparison of web backend and database: A case study of Node.js, Golang and MySQL, Mongo DB. *Recent Advances in Computer Science and Communications, 14*(6), 1955–1961.

Fhonna, R. P., & Fadli, M. (2022). Sistem informasi surat masuk dan surat keluar di Dinas Pendidikan Kota Binjai menggunakan Java NetBeans. *JTIK (Jurnal Teknik Informatika Kaputama, 6*(1), 32–40.

Fhonna, R. P., Afrillia, Y., Ilhadi, V., Aqmal, J., & Afwan, T. M. A. (2022). Pendeteksian masker secara real-time menggunakan TensorFlow untuk pencegahan Covid-19 di Prodi Sistem Informasi Universitas Malikussaleh. *G-Tech: Jurnal Teknologi Terapan, 6*(2), 183–190.

González-Mora, C., Garrigós, I., Zubcoff, J., & Mazón, J.-N. (2020). Model-based generation of web application programming interfaces to access open data. *Journal of Web Engineering, 19*(7–8), 1147–1172.

Hasan, S., & Muhammad, N. (2020). Sistem informasi pembayaran biaya studi berbasis web pada Politeknik Sains dan Teknologi Wiratama Maluku Utara. *IJIS-Indonesian Journal on Information System, 5*(1), 44–55.

Indrianto, I. (2023). Performance testing on web information system using Apache JMeter and Blazemeter. *Jurnal Ilmiah Ilmu Terapan Universitas Jambi, 7*(2), 138–149.

Kemer, E., & Samli, R. (2019). Performance comparison of scalable REST application programming interfaces in different platforms. *Computers in Industry, 111*, 41–50. https://doi.org/10.1016/j.csi.2019.05.001

Lee, E., Kwon, K., & Yun, J. (2020). Performance measurement of GraphQL API in home ESS data server. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 1929–1931). IEEE.

Mahajan, G., Attar, V., & Kalamkar, S. (2022). Generation of JMeter scripts for performance testing of Moodle server. In *2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)* (pp. 2277–2281). IEEE.

Pontarolli, R. P., Bigheti, J. A., de Sá, L. B. R., & Godoy, E. P. (2023). Microservice-oriented architecture for Industry 4.0. *Engineering, 4*(2), 1179–1197. https://doi.org/10.3390/eng4020069

Rak, T. (2023). Performance evaluation of an API stock exchange web system on cloud Docker containers. *Applied Sciences, 13*(17), 9896.

Saif, D., Lung, C.-H., & Matrawy, A. (2021). An early benchmark of quality of experience between HTTP/2 and HTTP/3 using lighthouse. In *ICC 2021-IEEE International Conference on Communications* (pp. 1–6). IEEE.

Sari, D. F., Kurniawati, D., & Muriyanto, F. (2021). Optimasi server menggunakan load balancing microservice Docker pada bot telegram. *Journal of Innovation Research and Knowledge, 1*(7), 335–342.

Suwarno & Yulandi, A.P. (2023). *Analisis performa backend framework: Studi komparasi framework Golang dan Node.js*. JURASIK, 8(3), 155–168. Available at https://tunasbangsa.ac.id/ejurnal/index.php/jurasik

Syed, A. R. (2021). The solution for XML external entity vulnerability in web application security. In *Smart intelligent computing and communication technology* (pp. 305–310). IOS Press.

Tiwari, V., Upadhyay, S., Goswami, J. K., & Agrawal, S. (2023). Analytical evaluation of web performance testing tools: Apache JMeter and SoapUI. In *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)* (pp. 519–523). IEEE.

Tiwary, G. P., Stroulia, E., & Srivastava, A. (2021). Compression of XML and JSON API responses. *IEEE Access, 9*, 57426–57439.

Vadlamani, S. L., Emdon, B., Arts, J., & Baysal, O. (2021). Can GraphQL replace REST? A study of their efficiency and viability. In *2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)* (pp. 10–17). IEEE.

Woody, S. K., Burdick, D., Lapp, H., & Huang, E. S. (2020). Application programming interfaces for knowledge transfer and generation in the life sciences and healthcare. *NPJ Digital Medicine, 3*(1), 24.

Wu, D., Jing, X., Zhang, H., Kong, X., Xie, Y., & Huang, Z. (2020). Data-driven approach to application programming interface documentation mining: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10*(5), e1369.