5th International Seminar of Research Month 2020
Volume 2021
http://dx.doi.org/10.11594/nstp.2021.0908

NST PROCEEDINGS

OPEN ACCESS

Conference Paper

Comparative Analysis of Rest and GraphQL Technology on Nodejs-Based Api Development

Gede Susrama Mas Diyasa[1]*, Gideon Setya Budiwitjaksono[2], Hafidz Amarul Ma'rufi[1], Ilham Ade Widya Sampurno[1]

[1]Department of Informatics, Faculty of Computer Science, Universitas Pembangunan Nasional "Veteran" Jawa Timur, Indonesia
[2]Departments of Accounting, Faculty of Economics and Business, Universitas Pembangunan Nasional "Veteran" Jawa Timur, Indonesia

Corresponding author:
E-mail:
igsusrama.if@upnjatim.ac.id

ABSTRACT

Web service is a method of connecting servers and client applications. There are several types of technology in developing a web service, such as REST and Graph-QL. Graph-QL is an alternative technology created by Facebook to correct REST technology's shortcomings, especially in the data presentation section. Graph-QL provides an alternative where the client application can determine for them what data is needed. This paper analyzes the performance of the two technologies to determine which technology is suitable for their needs. The analysis carried out is to compare the response speed and data efficiency to optimize the available bandwidth. The development model uses the waterfall model, which consists of research, design, implementation, and testing. As a test object, two Node-JS based applications were developed with the Express Framework, which applied REST and Graph-QL concepts on each test object. The results obtained are that REST has better performance than Graph-QL in its response speed. On the other hand, Graph-QL also excels in data presentation by client application requests to optimize the available bandwidth.

Keywords: REST, Graph-QL, API, Node-JS, Web Service

Introduction

An application has the term back-end, which is a data source and application logic source. The application's back-end data source is accessed by the client application using an API (Application Programming Interfaces) (Meng et al., 2018). The API is used to distribute the back-end and client applications to each other; in other words, one back-end can be accessed by several different client applications.

In implementing API, several technologies can be used, such as REST (Representational State Transfer) (Ghebremicael, 2017) or GraphQL (Brito et al., 2019). Both technologies present the same data in JSON (JavaScript Object Notation) (Lv et al., 2018). GraphQL is an alternative technology to REST introduced by Facebook. GraphQL (Brito and Valente, 2020) was created to improve the REST's shortcomings in terms of data presentation, where the data returned by the server can be adjusted according to client needs (Čechák, 2017).

Several studies related to the implementation of REST and GraphQL technology include references [7] [8] [9] [10], in relation (Masdiyasa et al., 2020) tested Graph-QL was on creating an integrated system of competency certification testing, which is web-based and compared with the efficiency of using the REST API. The test results found that Graph-QL has advantages in presenting data on the server, and the REST API has better performance in response speed but does not discuss NODEJS. Reference (Sayan & Shreyasi, 2020) compares Graph-QL with restful services in

How to cite:
Diyasa, I. S. M et al. (2021). Comparative analysis of rest and graphQL technology on nodejs-based api development. *5rd International Seminar of Research Month 2020*. NST Proceedings. pages 1-9. doi: 10.11594/ nstp.2021.0908

architectural API management and explains in detail both the management methods and architectural design of GraphQL and REST APIs and analyzing the potential benefits of GraphQL compared to REST in stateless API architectural design.

In reference (Brito & Valente, 2020), services on REST and GraphQL architectures provide similar performance to reference research (Seabra et al., 2019), where the value between REST and GraphQL services without discussing NODEJS. In contrast, reference (Hartig & Pérez, 2017) discuss Nondeterministic Logarithmic Space (NL) by analyzing Graph-QL Facebook to provide a new type of data access interface on the Web.

This paper will analyze the comparison between REST technology and GraphQL in handling a request for an integrated online competency test system application. Some of the analyzed things are the response speed and the efficiency of the data returned by the server (Khan & Mian, 2020). In its implementation, two applications will be created with different API technologies and accessing the same database. Applications built running on NodeJS as object appeal (Tran, 2019). It is hoped that this paper can help determine the appropriate technology to be used to make an application.

Literature Review

API (Application Programming Interface)

API stands for Application Programming Interface and allows developers to integrate two parts of an application or with different applications simultaneously (Hou et al., 2017). API consists of various elements such as functions, protocols, and other tools that allow developers to create applications. The purpose of using the API is to speed up the development process by providing separate functions so that developers don't have to build similar features (Mattia et al., 2019). will be significantly felt Implementation of the API if the desired quality is very complicated. Of course, it takes time to create something similar to it. For example, integration with a payment gateway. Various types of system APIs can be used, including operating systems, libraries, and the Web (Cutler & Dickenson, 2020).

REST

REST is an architectural model connecting two web-based client and server applications using HTTP (HyperText Transfer Protocol) is a standard protocol (Halili & Ramadani, 2018). Thus, the server can be accessed by several client applications even though different platforms such as mobile, desktop, and website. Website applications that follow the REST architecture are called RESTful web services. Some of the HTTP methods that are often used are GET to retrieve data, POST to add data, PUT to edit data, and DELETE to delete data (Melnichuk et al., 2018). To handle the response, a server with REST architecture usually sends HTTP codes to state the request's status. Commonly used codes include 200 (OK), 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), 429 (Too Many Request) and 500 (Internal Server Error) (Dell, 2018).

GraphQL

GraphQL is a query language for website-based API implementations. Unlike REST, GraphQL has the advantage of presenting data dynamically according to client applications (Nogatz & Seipel, 2017). GraphQL provides one endpoint for accessing all data sources on the server. To describe GraphQL compared to REST, we can take an example of user data with twelve attributes. In REST, will be returned all existing features to the client application. Meanwhile, in GraphQL, the client application can determine what data is needed so that the data produced is very accurate, depending on the client application (Ritsilä, 2017). Some basic terms in GraphQL are types that define data, queries are a transaction model from server to client used for data retrieval, and mutations are transaction models that allow changing data in the database (Mukhiya et al., 2019).

JSON

JSON is an independent data format in JavaScript with Standard ECMA-262 3rd Edition - December 1999 (Bray, 2014). The JSON data format is widely used in data exchange on a web service. JSON is used because it is a universal format for multiple platforms. In other words, this data format can be read by any medium, so it is very suitable for use in data exchange, especially between server and client applications.

NodeJS

NodeJS is software used to build website-based applications (Bangare et al., 2016). NodeJS runs on the server-side (Rimal, 2019). NodeJS was introduced by Rayn Dahl in 2009 to provide a lighter and more efficient input/output model. NodeJS is written in C, C ++, and JavaScript built on Google Chrome's V8 engine (Brown, 2016).

Methods

The flow of development uses the Waterfall method, one of the classic life cycles in software development. This method describes a reasonably systematic and sequential approach to software development, starting from: specification of user requirements, planning, modeling, construction, submission of systems to users, and system maintenance.

In the waterfall method. If stage 1 has not been completed (Bassil, 2012), then step 2 cannot run, and so on. All sets are interrelated, and each must be done in detail and documented. The waterfall method requires that every specification, requirements, and system objectives be defined at the initial stage (requirements & design) before entering the process (implementation). This is because the waterfall method does not accommodate changes in the middle of the development process (Dima & Maassen, 2018). So, what the analyst and client team agreed on in the early stages will be the final result. Everything must comply and be consistent with it until the application is completed and submitted to the client. The client itself cannot intervene with the programmers. This is different from some other development models that allow the two to communicate to determine or revise the work at the coding stage. The waterfall method is generally used in large system creation projects with high complexity and requires many human resources in its development (Yu, 2018). Can be seen the waterfall method stages in Figure 1.

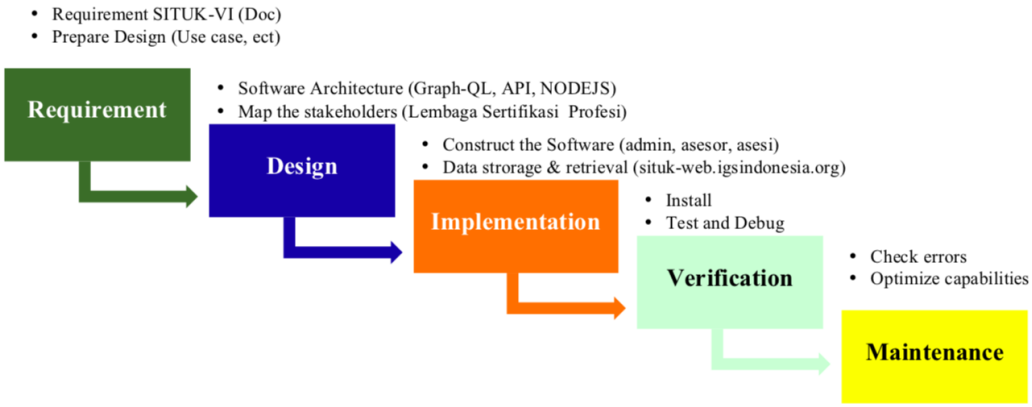


Figure 1. The flow of system development with the waterfall method

Analysis requirement

At this stage, several system requirements are analyzed (Casteren, 2017). Necessity is the process of investigating or collecting data relating to the system to be created. This data collection can be done by interview, literature study, observation, or direct research. In this phase, the analyst

team will dig up as much information as possible from the client or user about what software they want and other system requirements. The results of this stage will produce a document called a "User Requirement Document" or "User Requirements Specification." The requirements analysis stage has various other terms, including system requirements, customer requirement gathering, analysis, or user needs analysis (Buchori et al., 2017).

The analysis carried out at this stage analyzes the technology to be used by searching for some literary sources, especially about GraphQL and REST. At this stage, the analysis of tools, libraries, and frameworks used to support system development needs is also carried out. As test objects, it will be created ¸can be seen with different API technologies, namely GraphQL and REST. The application will be built based on NodeJS with the Express Framework and use MongoDB as a database that runs on the local server.

Design and implementation stages

This design process will focus on building data structures, software architecture, designing interfaces, designing internal and external functions, and the details of each procedural algorithm (Eason, 2016). The design stage will produce a document called "Software Requirements," which will become the basis for programmers in creating application codes. Meanwhile, the implementation process is the stage of making applications by programmers using specific programming language codes. The application coding process refers to the previously created documents. In this document, there is usually a solution to the system modules. Several programmers can do the application work at once without disturbing the other system as a whole. The implementation stage is also called the code and debug step, called the integration and system testing stage (Salim et al., 2014).

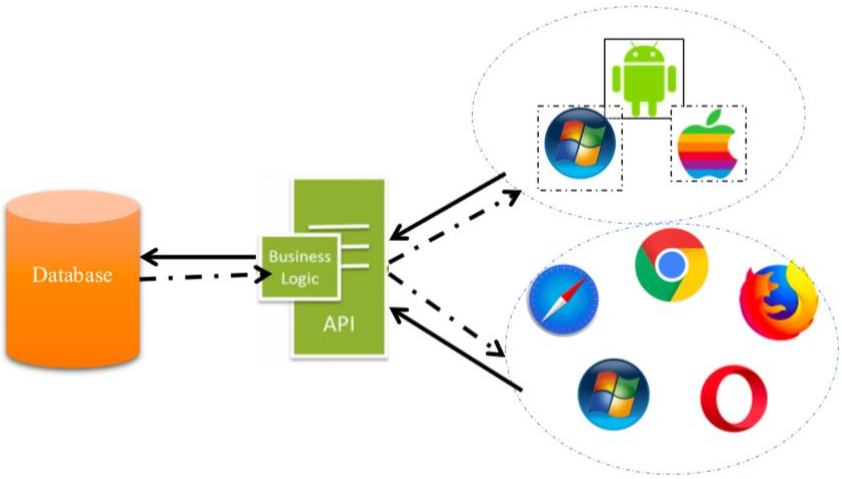


Figure 2. API process flow

APIs that work at the operating system level help applications communicate with the base layer and with each other following a series of protocols and specifications. An example that can illustrate this specification is POSIX (Portable Operating System Interface). Using the POSIX standard, applications compiled to run on a particular operating system can also run on other systems with the same criteria. The software library also plays an essential role in creating compatibility between different systems (Walli, 1995).

Applications that interact with the library must follow a set of rules defined by the API. This approach makes it easy for software developers to build applications that communicate with multiple libraries without rethinking the strategies used as long as all libraries follow the same API.

Another advantage of this method is how easy it is to use the same library in different programming languages. As the name suggests, the Web API is accessed via the HTTP protocol. This is a concept, not a technology. And Web APIs can be made using different technologies such as PHP, Java, .NET, Etc. For example, the Rest API from Twitter provides read and write access to data by integrating Twitter into our applications.

In this study, the flow starts from an incoming request to the API Gateway, which is at the "/api" endpoint, as shown in Figure 2. At this endpoint, we will use GraphQL and REST with some data retrieval logic to the database. The database used contains a table named "todos" with a structure, as shown in Figure 3.

Several APIs will be made in REST as candidates, as shown in Table 1. While the GraphQL type API will make several queries and mutations as candidates, as shown in Table 2.

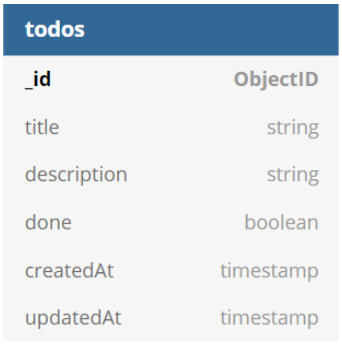


Figure 3. Table structure "todos."

Table 1. Candidate API on REST

Method	Endpoint	Deskripsi
GET	/api/todos	Fetch all todo data
GET	/api/todos/:_id	Retrieves todo data based on _id
POST	/api/todos	Create a new todo
PUT	/api/todos/:_id	Changing todo data based on _id
DELETE	/api/todos/:_id	Delete todo data based on _id

Table 2. Candidate queries in Graph-QL

Type	Query	Deskripsi
Query	todos	Fetch all todo data
Query	todo(_id: ID)	Fetch data todo based on id
Mutation	created(data: TodoData)	Create a new todo
Mutation	updateTodo(_id: ID,data: TodoData)	Change todo data based on id
Mutation	deleted(_id: ID)	Delete todo data based on id

Stages of Testing (Verification) and Maintenance

The testing (verification) stage includes system integration and also testing the applications that have been made. The system will be verified to be tested to what extent it is feasible. In this stage, all modules that are worked on by different programmers will be combined and then tried whether they are by the specified specifications or errors/errors in the system before being corrected again

The maintenance stage generally includes the steps of software installation and application testing. Maintenance is also a form of the development team's responsibility to ensure the application can run smoothly after being handed over to the client for a certain period. In a broader definition, maintenance is the process of fixing an application from any errors or security holes, improving application performance, ensuring the application can run in a new scope, and adding further application development modules

In this study, testing was carried out using the Postman application by requesting an API. In this test, the response time or speed of an API is calculated in handling a request. In this test, there were 100 iterations of an API.

Results and Discussion

Testing

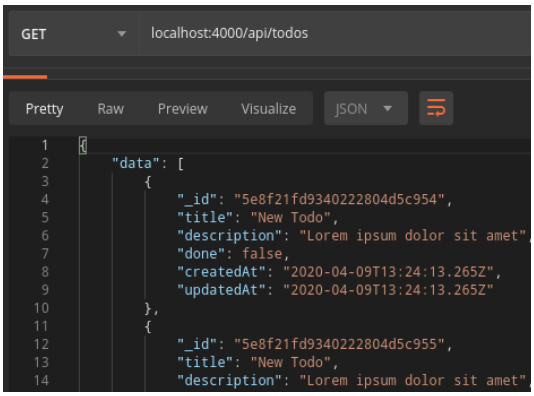
The response results from the REST-based API in Figure 4.a. The response data in Figure 4.a. is obtained by requesting the endpoint "/ api / todos" using the GET method. In this case, the API returns 1000 data with a response size of 183.82KB. Meanwhile, the response results from the GraphQL-based API are presented in Figure 4.b.

The response data in Figure 5 is obtained by requesting the GraphQL-based API by sending the request body in the query attribute as the requested data request to GraphQL. In this case, the API returns 1000 data with a response size of 183.83KB. The ordered request body structure is shown in Figure 4.c.

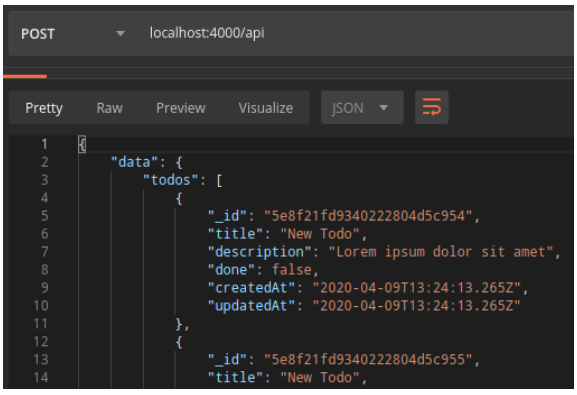
GraphQL request body is done by sending the required API type data and attributes. In this case, the requested API is "todos" with type "query" and the required attributes include "_id", "title", "description", "done", "createdAt" and "updatedAt".

On the other hand, an API built using GraphQL can handle requests with attributes that are custom defined by the client application with results, as shown in Figure 4.d below. Unlike the previous bid, in the business attributes, the client application can determine what features are needed. The response size returned by the API is 75.43KB. The request body structure is requested, as shown in Figure 4.e.

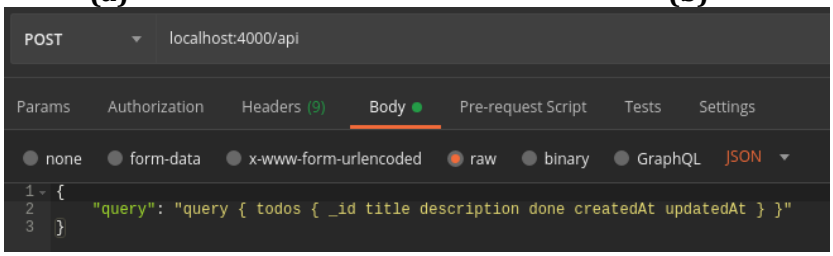
In this case, the client application only requests three custom attributes to the API, which includes the "title," "description," and "done" attributes.



(a)



(b)



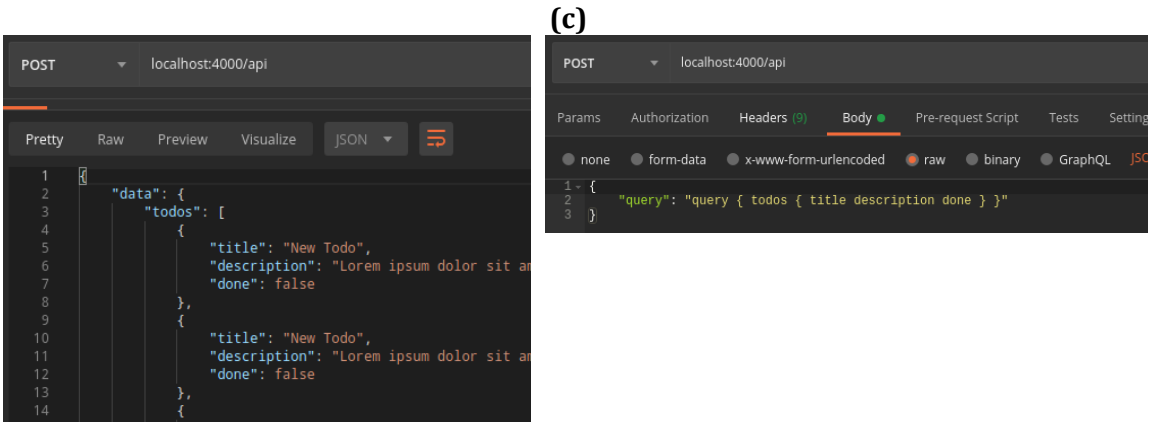


Figure 4. (a) Response data from REST-based API. (b) Response data from GraphQL-based API. (c) Request body on GraphQL-based API, (d) Response data from GraphQL-based API with custom attributes, (e) FRequest body in GraphQL-based API with custom attributes

Results

By making 100 repeated requests using the Postman application, the results of the response speed on the REST-based API are shown in Figure 5.a. In the REST-based API, the average speed for each type of API is obtained with 1000 data in the database as follows:

1. GET: 46.5 ms
2. POST: 3.67 ms
3. PUT: 3.51 ms
4. DELETE: 3.25 ms

The same test method is carried out on the GraphQL-based API with the test results in Figure 5.b. In the GraphQL-based API, the average response speed for each type of API is obtained in handling a request with 1000 data in the database as follows:

1. todos: 49 ms
2. createTodo: 3.88 ms
3. updateTodo: 3.81 ms
4. deleteTodo: 3.6 ms

So that the speed between the two can be compared, which is presented in Table 3., and Figure 6 is shown in graphic form.

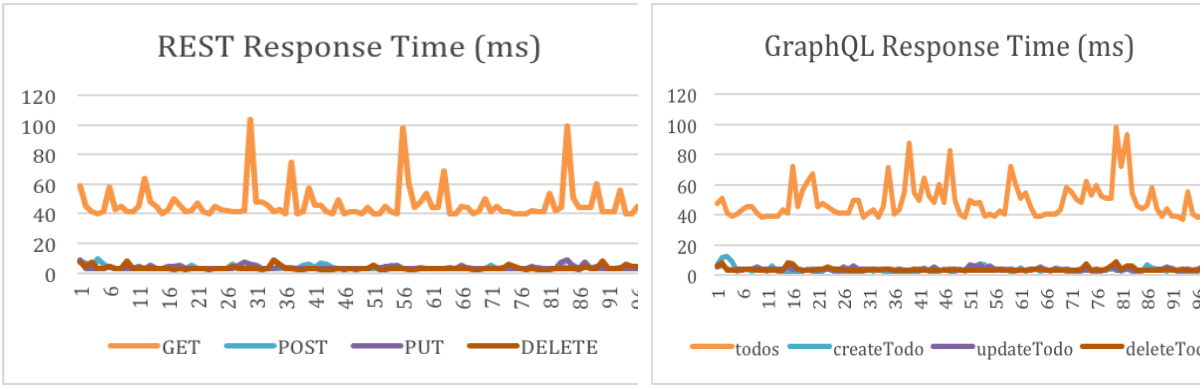


Figure 5. (a) Response speed on REST-based API. (b) Response speed on GraphQL-based API.

Table 3. Comparison of average response rates

Type	Average (ms)	
	REST	Graph-QL
Read	46.5	49
Create	3.67	3.88
Update	3.51	3.81
Delete	3.25	3.6

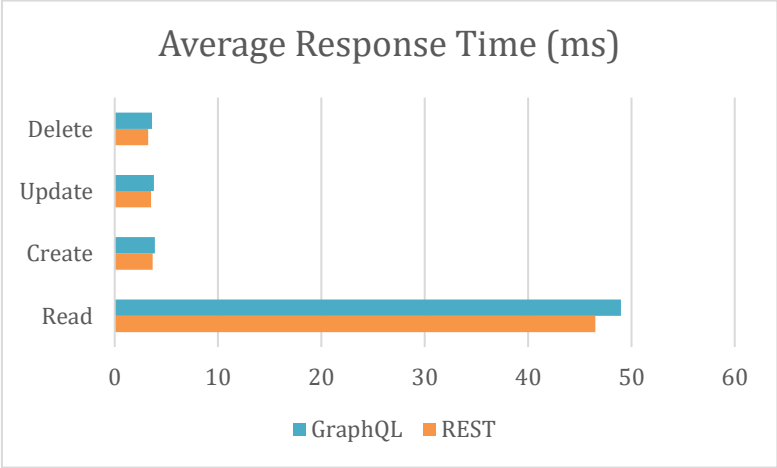


Figure 6. Graph of average response speed

The average data obtained between REST and GraphQL in terms of response speed to a request can say that REST performs better than GraphQL, although with a slight difference in performance.

Conclusion

The strain *S. prasinopilosus* Pn-TN2 which isolated from termite nest sample, showed a broad range of

1. The average response speed data on the REST-based API and NodeJS-based GraphQL with the Express Framework can conclude that REST has better performance than GraphQL in terms of response speed on the server in handling a request.
2. GraphQL is superior in presenting data to client applications, where client applications can define their custom attributes so that no data is useless. This makes bandwidth usage more optimal

Acknowledgment

Thank you to LPPM UPN Veteran East Java, for the research that has been given, so that we can publish papers in several journals

References

Bangare, S, Gupta, S., Dalal, M., & Inamdar, A. (2016). Using Node.Js to build high speed and scalable backend database server. *International Journal of Research in Advent Technology, 4*, 19.

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *International Journal of Engineering & Technology (iJET), 2*(5), 1–7.

Bray, T. (2014). The JavaScript Object Notation (JSON) data interchange format. *Internet Engineering Task Force (IETF), ECMA International, 1st Editio (October),* p. 8. Available at: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf%5Cnhttps://www.rfc-editor.org/info/rfc7158.

Brito, G., & Valente, M. T. (2020). *REST vs GraphQL: A controlled experiment', Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020, (February), 81–91*. doi: 10.1109/ICSA47634.2020.00016.

Brito, G., Mombach, T., & Valente, M. T. (2019). Migrating to GraphQL: A Practical Assessment. *SANER 2019 - Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering,* pp. 140–150. doi: 10.1109/SANER.2019.8667986.

Brown, E, (2016). Web development with Node & Express. *Journal of Chemical Information and Modeling*, *53* (9), 1689-1699

Buchori, A., Setyosari, P., Dasna, I. W., & Ulfa, S. (2017). Mobile augmented reality media design with waterfall model for learning geometry in college. *International Journal of Applied Engineering Research, 12*(13), 3773–3780.

Casteren, W. V. (2017). The waterfall model and the agile methodologies: A comparison by project characteristics-short the waterfall model and agile methodologies. *Academic Competences in the Bachelor, (February),* 10–13. doi: 10.13140/RG.2.2.36825.72805.

Čechák, D. (2017). *Using GraphQL for content delivery in Kentico Cloud', Is.Muni.Cz*. Available at: https://is.muni.cz/th/qm0cs/thesis.pdf.

Cutler, J., & Dickenson, M. (2020). *Application programming interfaces*, 87–97. doi: 10.1007/978-3-030-36826-5_7.

Dell, Emc. (2018). *Unisphere ® Management REST API Programmer ' s Guide', in Dell EMC Unity TM Family Version 4.3*, 1–122.

Dima, A. M., & Maassen, M. A. (2018). From waterfall to agile software: Development models in the IT sector, 2006 to 2018. impacts on company management. *Journal of International Studies, 11*(2), 315–326. doi: 10.14254/2071-8330.2018/11-2/21.

Eason, O. K. (2016). *Information systems development methodologies transitions: an analysis of waterfall to agile methodology.* University of New Hampshire, pp. 1–23. Available at: http://scholars.unh.edu/honors%0Ahttp://scholars.unh.edu/honors.

Ghebremicael, E. S. (2017). *Transformation of REST API to GraphQL for OpenTOSCA*. doi: 10.18419/opus-9352.

Halili, F. & Ramadani, E. (2018). Web services: a comparison of soap and rest services. *Modern applied science, 12*(3), 175. doi: 10.5539/mas.v12n3p175.

Hartig, O., & Pérez, J. (2017). An initial analysis of facebook's GraphQL language. *CEUR Workshop Proceedings, 1912(May).*

Hou, L., Zhao, S., Li, X., Chatzimisios, P., & Zheng, K. (2017). Design and implementation of application programming interface for Internet of things cloud. *International Journal of Network Management, 27*(3), 1-10. doi: 10.1002/nem.1936.

Khan, R., & Mian, A. N. (2020). Sustainable IoT sensing applications development through graphQL-based abstraction layer. *Electronics (Switzerland), 9*(4), 1–23. doi: 10.3390/electronics9040564.

Lv, T., Yan, P., & He, W. (2018). Survey on JSON data modelling. *Journal of Physics: Conference Series, 1069*(1), 1-10. doi: 10.1088/1742-6596/1069/1/012101.

Masdiyasa, I. G. S., Budiwitjaksono, G. S., Amarul, M. H., Sampurno, I. A. W., & Mandenni, N. M. I. M. (2020). 'Graph-QL Responsibility Analysis at Integrated Competency Certification Test System Base on Web. *Lontar Komputer, 11*(2), 114–123. doi: https://doi.org/10.24843/LKJITI.2020.v11.i02.p05

Mattia, S., Lorenzino, V., Dimitrios, M., Robin, S., Monica, P. S., & Dietmar, G. (2019*). Web application programming interfaces (apis): General purpose standards, terms and European commission initiatives.* doi: 10.2760/675.

Melnichuk, M., Kornienko, Y., & Boytsova, O. (2018). Web-Service. Restful Architecture. *Automation of technological and business processes, 10*(1), 17–22. doi: 10.15673/atbp.v10i1.876.

Meng, M., Steinhardt, S., & Schubert, A. (2018). Application programming interface documentation: What do software developers want?. *Journal of Technical Writing and Communication, 48*(3), 295–330. doi: 10.1177/0047281617721853.

Mukhiya, S. K., Rabbi, F., Pun, V. K. I., Rutle, A., & Lamo, Y. (2019). A graphQL approach to healthcare information exchange with HL7 FHIR. *Procedia Computer Science, 160,* 338–345. doi: 10.1016/j.procs.2019.11.082.

Nogatz, F., & Seipel, D. (2017). Implementing GraphQL as a query language for deductive databases in SWI-Prolog using DCGs, quasi quotations, and dicts', Electronic Proceedings in Theoretical Computer Science, *EPTCS, 234(January 2017),* 42–56. doi: 10.4204/EPTCS.234.4.

Rimal, A. (2019). *Developing a Web Application on NodeJS and MongoDB using ES6 and Beyond. Bachelor of Engineering, Information Technology, University of Applied Science*. Available at: https://www.theseus.fi/bitstream/handle/10024/159951/Rimal_Aashis.pdf?sequence=1&isAllowed=y.

Ritsilä, A. (2017). *GraphQL: The API Design Revolution.* Bachelor's Thesis Degree Programme in Bit, University of Applied Science, Haage-helia.

Salim, A. P., Chithra, P., & Sreeja, S, (2014). Survey on Different Process Models Used In Software Development. *International Journal of Computer Science and Information Technologies*, *5*(3), 2855–2860.

Sayan, G., & Shreyasi, M. (2020). A Comparative Study Between Graph-QL & Restful Services In API Management Of Stateless Architectures. *International Journal on Web Service Computing (IJWSC), 11*(02), 1–16. doi: 10.5121/ijwsc.2020.11201.

Seabra, M., Nazário, M. F., & Pinto, G. (2019). REST or GraphQL? A performance comparative study. *ACM International Conference Proceeding Series, (June),* 123–132. doi: 10.1145/3357141.3357149.

Tran, T. (2019). *Build a GraphQL application with Node. js and React', (April).*

Walli, S. R. (1995). The POSIX family of standards. *StandardView, 3*(1), 11–17. doi: 10.1145/210308.210315.

Yu, J. (2018). Research Process on Software Development Model. *IOP Conference Series: Materials Science and Engineering*, *394*(3). 1-10. doi: 10.1088/1757-899X/394/3/032045.