Jurnal E-KOMTEK (Elektro-Komputer-Teknik)
Vol. 5, No. 2 (2021) pp. 178-189
https://jurnal.politeknik-kebumen.ac.id/index.php/E-KOMTEK
p-ISSN : 2580-3719 e-ISSN : 2622-3066

REST API Architecture Design on Multi-Platform Device Development

Yuda Syahidin, Randy Ramadhan
Department of Information System, Politeknik Piksi Ganesha Bandung, Indonesia, 40274

yuda.syahidin@piksi.ac.id

https://doi.org/10.37339/e-komtek.v5i2.762

Published by Politeknik Dharma Patria Kebumen

Artikel Info
Submitted:
30-10-2021
Revised:
03-11-2021
Accepted:
03-11-2021
Online first :
30-12-2021

Abstract

Multi-platform devices are becoming the standard technology in software development. However, the problem of multi-platform is that it must provide a single parent data source, so there is a good relationship between users of multi-platform devices and servers. REST APIs are the best choices that bridge the servers and multiple platforms. That way, the use of REST APIs will be very beneficial, both in terms of servers and platforms as consumption. This research planned to create a REST API architecture that can run in multi-platform with testing techniques using black boxes through postman software. As the basis for making REST APIs, it is better to understand the Architecture of REST APIs, especially REST APIs as a multi-platform device.

Keywords: Architecture, Multi-platform, REST API

Abstrak

Perangkat multi-platform menjadi teknologi standar dalam pengembangan perangkat lunak. Namun, masalah multi-platform harus menyediakan sumber data single parent, sehingga ada hubungan yang baik antara pengguna perangkat multi-platform dan server. REST API adalah pilihan terbaik yang menjembatani server dan berbagai platform. Dengan begitu, penggunaan REST API akan sangat menguntungkan, baik dari sisi server maupun platform sebagai konsumsinya. Penelitian ini direncanakan untuk membuat arsitektur REST API yang dapat berjalan secara multi platform dengan teknik pengujian menggunakan black box melalui software tukang pos. Sebagai dasar untuk membuat REST API, lebih baik memahami Arsitektur REST API, terutama REST API sebagai perangkat multi-platform.

Kata-kata kunci: *Arsitektur, Mlti-Platform, REST API*

1. Introduction

Information and communication technology is vital today. The procurement of data and software plays a very striking role to protect its users. In addition to this, the company's various innovations resulted in many platforms, such as Smartphones, Smart TVs, and common devices such as laptops and PCs. Undoubtedly, the applications made must be adjusted to the platform, because the architecture on the hardware might be different. In other words, there must be a synchronized data on the program, whether it is a smartphone or the web.

On the side of everyday users, certainly, the current situation is very helpful with the aforementioned conditions. However, the problem occurs on the development side of the software itself. Therefore, a bridge is primary, so that the data on multi-platform devices can communicate properly.

Speaking of the development of data-driven technology, it needs to be thought of how the data flow from the server is the same or synchronized with several platforms used at once. That way, the communication across platforms runs smoothly without data redundancies, which due to too much storage, might cause programs not effectively used.

In the previous research **[1]**, shortcomings in the architecture were not applied and explained in detail. The explanation was only oriented to the REST API application. In other study **[2]**, there is no authentication process, so the security on REST APIs was particularly vulnerable to sensitive requests and responses.

To cover the shortcomings of these studies, the authors conducted the research and created a multi-platform REST API architecture design that gives more meaning to the function and usability of the REST API.

2. Method

Architectural Design is the art and science of designing and constructing buildings, bridges, and so on **[3]**. In this case, architecture refers to designing and building a construction of a REST API network to interact with multi-platform devices.

REST is a web service that uses the concept of transferring from state to state over HTTP to make a specific connection **[4]**,**[5].** With a stateless REST nature, requests should always include complete data and parameters. An API (Application Programming Interface) is an interface that is implemented to software to interact with other lifters **[5]**.

Software development is the activity of writing and managing program code, but in a broad sense, the term encompasses all the things involved between the creation of desired software through software finalization, ideally in a planned and structured process **[6]**. Therefore, software development may include research, new development, prototypes, modifications, reuse, reengineering, management, or other activities that produce software products **[7]**.

Multi-platform is an application that can be run by any operating system. The function of the multi-platform problem is as a "consumption" rest API created by the backend. Technologies such as website platforms, mobile platforms, or desktop platforms. The advantage of multi-platform use is the data that each platform can be interacted with each other **[8]**.

Architectural design uses object-based analysis and design, namely OOAD (Object Oriented Analysis and Design). OOAD can be considered to be a relatively new project management rather than analysis and design based on diagram data. Object Oriented Analysis and Design is a method used to make objects or objects referred to as actors, namely representations of humans who will interact with the system. The following **Figure 1** describes the design of objects based on Iterative and Incremental Development.

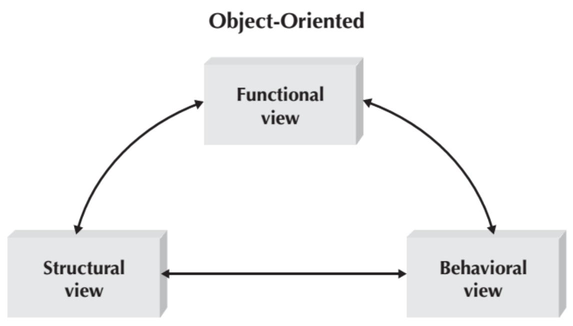


Figure 1. Iterative and Incremental Development [9]

By utilizing this basic concept of OOAD, it will be very easy to represent all forms of behavior from the components of the analysis component to produce a clear REST API architecture**[10]**.

Web services are purpose-built web servers that support the needs of a site or any other application. Client programs use application programming interfaces (APIs) to communicate with web services. Generally speaking, an API exposes a set of data and functions to facilitate interactions between computer programs and allow them to exchange information**[11]**. As depicted in Figure 2, a Web API is the face of a web service, directly listening and responding to client requests.

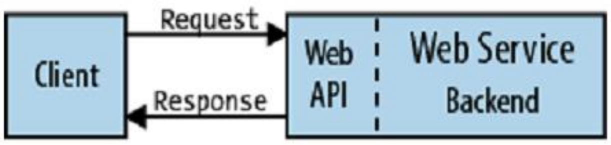


Figure 2. Web API [11]

The Architecture testing of the REST API used black-box testing. Blackbox testing is testing the smoothness of programs that have been created, especially in the functional parts of the architecture. Blackbox testing requires no design or program code, making it suitable for Architecture testing of the REST API [12]. The following Figure 3 describes the design of Black-Box Testing.

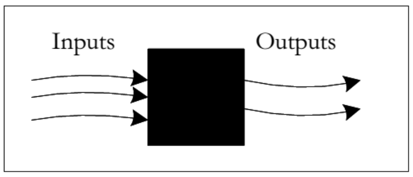
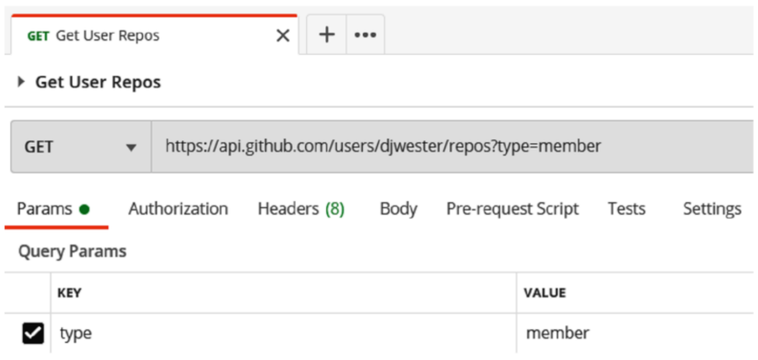


Figure 3. Black-Box Testing [12]

The software used to support black box testing is Postman. Postman is software that runs on a multi-system computer operating system, where its function is as a client to test the REST API that has been created [13]. To add parameters to an API endpoint in Postman, make sure you have the Params tab selected and then put the name of the query parameter into the Key field and the value into the Value field. In this case, we used the type parameter, so enter that word into the Key field. For this endpoint, the type parameter allows filtering based on whether the owner of a repository is or just a member. See Figure 4 that describes the Query parameter type in an API call below.



GET https://api.github.com/users/djwester/repos?type=member

	KEY	VALUE
✓	type	member

Figure 4. Query parameter type in an API call [13]

The Research Stage consists of 5 stages, namely: (1) Identification, (2) Literature Study (3) Analysis System, (4) Design Architecture and Implementation (5) Testing Blackbox REST API on Postman. In flow, it is explained in Figure 5 below.

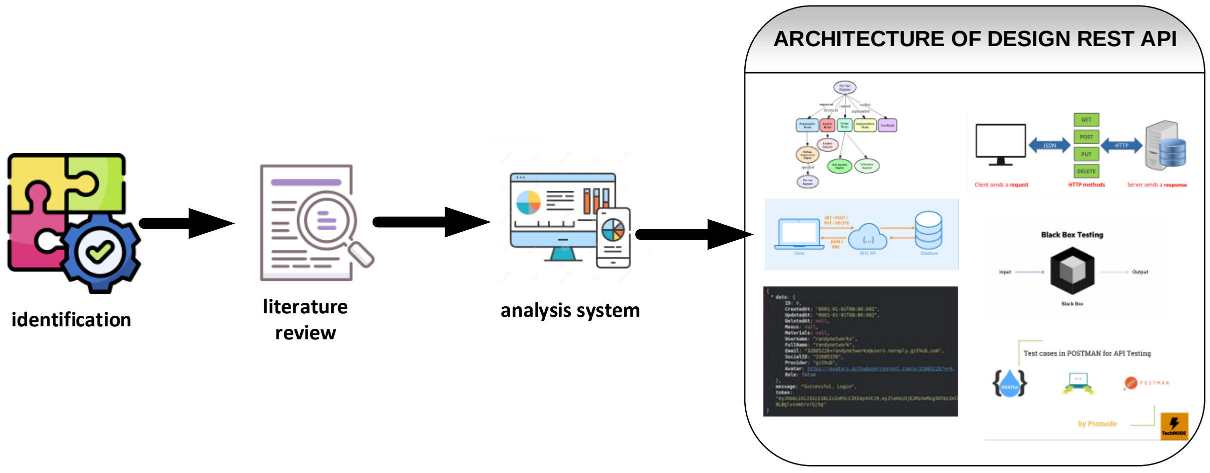

identification
literature review
analysis system
ARCHITECTURE OF DESIGN REST API
Black Box Testing
Test cases in POSTMAN for API Testing

Figure 5. Research Stage

The above research stages are elaborated as follows. The first stage is the identification of the problem. To find the subject, namely the concept of a multi-platform rest API architecture. The problem can be solved if the server creates an HTTP REST API endpoint so that each platform can consume the same data in one source. This resulted in synchronization between platforms [14].

The next stage is designing the REST API Architecture using OOAD to be technically clear, consisting of Use case Diagrams, Activity Diagrams, and Component Diagrams. Use case Diagrams are used to analyze the behavior of the software application to be created and describe an interaction between one or more actors and the application [9]. Use case Diagrams in the design of the REST API architecture are used to find out what and who is involved in the REST API architecture on multi-platform devices. Activity diagram describes the activities of a system or business process or menu in the software [15], which is used to find out the flow of how the REST API architecture works to be used by various multi-platform devices. The final component diagram shows the organizational environment, configuration, and destitution of the REST API and multi-platform devices.

The final stage is Blackbox testing, where testing is done on a created REST API endpoint, and testing whether the REST API architecture can run properly on multi-platform devices.

3. Results and Discussion

There are two components or actors involved in the REST API architecture on multi-platform devices, the first being servers, and multi-platform devices. In detail, the server can configure any data that you want to issue as REST API data, and also has authorization so that the data consumed by multi-platform devices are more secure and selected, especially sensitive HTTP methods. On multi-platform devices, you can request rest API under what has been provided by the server. To represent the above statement, the following **Figure 6** is an analysis of UML Use case diagram.

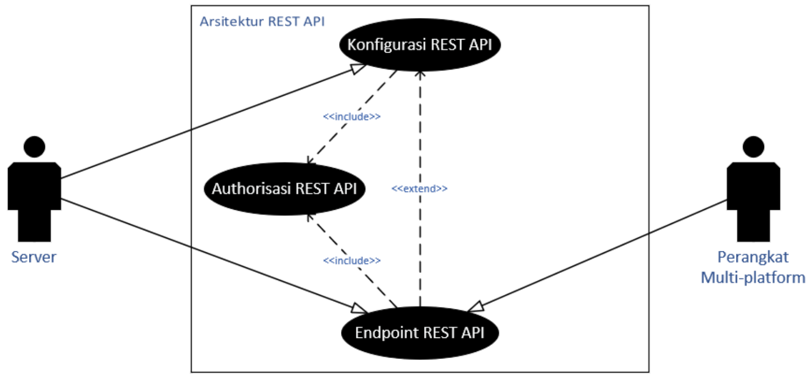


Figure 6. Use case Diagram

After knowing the design diagram UML use case, the next stage is to analyze the activities of each actor involved. The server side initially set the functionality of each HTTP endpoint to be used as a REST API, in the functionality of each endpoint, there are several customized configurations, one of which is the logic of the response or request. In conditions where data addition, data renewal, and data deletion functions, must use authorization to secure that data is not manipulated by users or platforms. Uses such as Oauth tokens on headers from requested APIs are highly recommended for authentication security. Certainly, some data are open to the public, such as displaying the entire data or specific data. Next is to set the HTTP endpoint route so that it can be accessed by multi-platform devices.

Endpoint writing has essentially been standardized internationally. Here are the details of how to write HTTP endpoint routes:

a) Use plural nouns such as: /products, /books, or /menus.
b) Use standard and relevant methods according to needs such as: GET, POST, PUT, or DELETE.
c) Use sub-resources if there are table relationships such as: /posts/1/comments.
d) Use parameters as needed such as: /posts?search='rest&api' or data filter /post?filter='rest'.

e) Use of HTTP code to return the response you want to issue. HTTP code is standardized and has its own role, such as: HTTP code 200 to restore that data is available and can be used, or HTTP code 401 to restore that data is not available.

f) Versioning REST API on HTTP is very must be so that the platform that uses it knows when using what version of rest API. Versioning rest API also makes it easy to create new versions of rest API and will not affect previous versions of development or production.

By utilizing the details of the standard way of writing, it is easy from the side of the platform that consumes the REST API.

The last stage of the server-side is to return the data according to the request of the multi-platform device. Several conditions must be handled by the server when multi-platform devices make requests, including:

1. Successful response

Successful responses are used when a request is successful. For example, the requested data is on the server. Returns made using JSON in the following formats:

```
{
 "status": "success",
 "Data":
 },
 "message": "Data successfully in"
}
```

2. Error response

The second treatment is when data or endpoints are not available on the server. Returns made of course use JSON with the following format:

```
{
 "status": "error",
 "data": null,
 "message": "message error",
 "errorCode":401
}
```

Requests exist to respond to requests from multi-platform devices, so they can be used or processed as needed. Here in **Figure 7** is an activity diagram on the server-side.

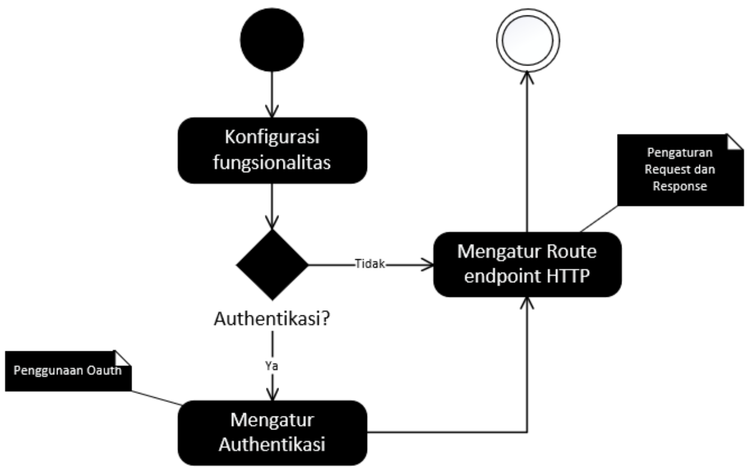

Konfigurasi fungsionalitas
Authentikasi?
Tidak
Ya
Mengatur Route endpoint HTTP
Pengaturan Request dan Response
Penggunaan Oauth
Mengatur Authentikasi

Figure 7. Activity Server Diagram

On the platform side, it initially set the route endpoint HTTP you want to access. In this section, adjusted to the methods used and authorization when needed. If the data is received, the platform will process the data, either displayed directly or reprocessed. In authorization, authentication is stored in an HTTP header, in the form of JSON, so it can be read by the server. If the server provides HTTP code 200, the data is successfully retrieved or manipulated. The activity diagram is displayed in **Figure 8** as follows.

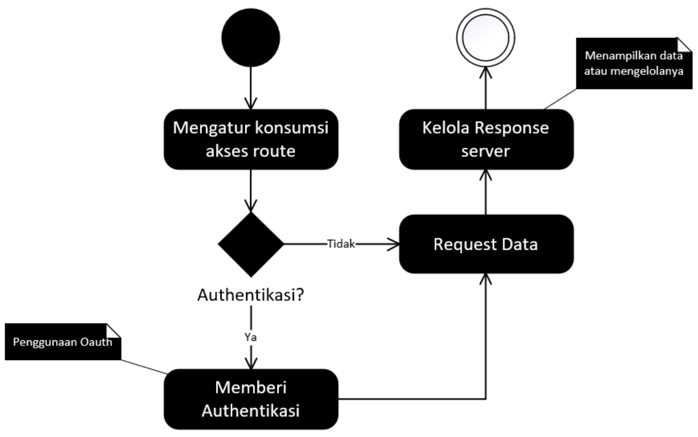


Figure 8. Activity Platform Diagram

In this part of the activity, it can be seen that the interaction between the server and the multi-platform device has been integrated and synchronized with each other.

a) The final analysis of the REST API Architecture on multi-platform devices is representing the organizations technically involved using component diagrams. In this section, it is depicted that both the server and the platform are outside the scope of the architecture. Internal architecture has several components as follows: Endpoint Component. This component is used as a portal for either a server that configures endpoints or a platform that accesses a specific endpoint.

b) Function Component

This component serves as a handler to address the functionality of the REST API request and response to the platform.

c) Authenticate Component

If the function component succeeds without constraints, the final stage is on the authentication component, which means that if there is a functional component that requires authentication, then this component will check, both on HTTP methods and uses such as Oauth for authentication security. The entire rest API architecture component is described in **Figure 9** as follows:

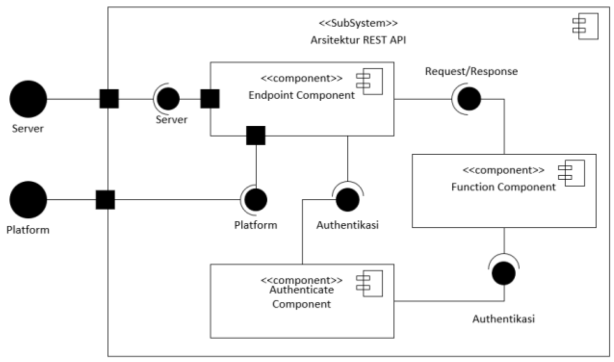
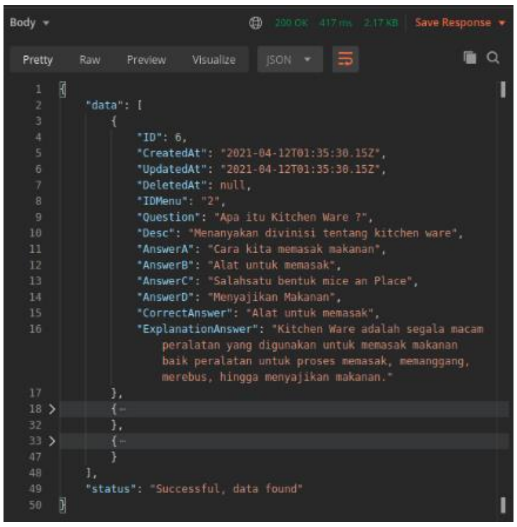


Figure 9. Component Diagram REST API

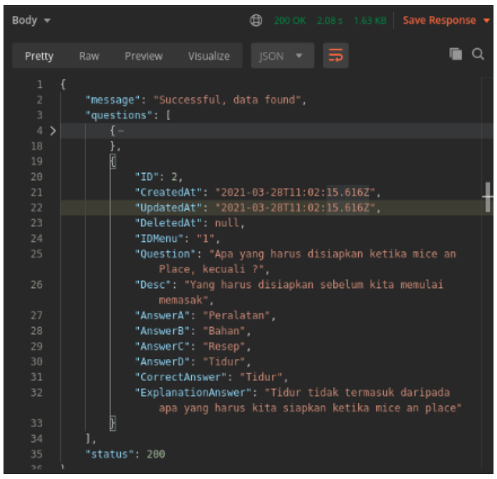
To find out the OOAD plan runs smoothly, then testing is needed. Therefore, black-box testing is used. Blackbox testing will be limited to endpoint access only because the purpose of black-box testing is to find out whether the server can respond to the data as desired or not. For examples of testing the REST API architecture, use APIs that have been adjusted both in configuration and functionality. Here are the results of requests and responses to the REST API using Postman (see **Figure 10**).



```
{
    "data": [
        {
            "ID": 6,
            "CreatedAt": "2021-04-12T01:35:30.15Z",
            "UpdatedAt": "2021-04-12T01:35:30.15Z",
            "DeletedAt": null,
            "IDMenu": "2",
            "Question": "Apa itu Kitchen Ware ?",
            "Desc": "Menanyakan divinisi tentang kitchen ware",
            "AnswerA": "Cara kita memasak makanan",
            "AnswerB": "Alat untuk memasak",
            "AnswerC": "Salahsatu bentuk mice an Place",
            "AnswerD": "Menyajikan Makanan",
            "CorrectAnswer": "Alat untuk memasak",
            "ExplanationAnswer": "Kitchen Ware adalah segala macam
                peralatan yang digunakan untuk memasak makanan
                baik peralatan untuk proses memasak, memanggang,
                merebus, hingga menyajikan makanan."
        },
        {…
        },
        {…
        }
    ],
    "status": "Successful, data found"
}
```

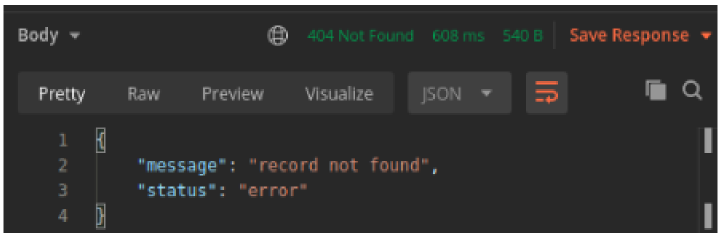
Figure 10. Request data with GET Method

It is explained that the request data with the get method in Figure 10 was successfully overcome as a response from the server. Displayed JSON response structure with HTTP code 200, and the duration needed to respond is 417 ms.



```
{
    "message": "Successful, data found",
    "questions": [
        {…
        },
        {
            "ID": 2,
            "CreatedAt": "2021-03-28T11:02:15.616Z",
            "UpdatedAt": "2021-03-28T11:02:15.616Z",
            "DeletedAt": null,
            "IDMenu": "1",
            "Question": "Apa yang harus disiapkan ketika mice an
              Place, kecuali ?",
            "Desc": "Yang harus disiapkan sebelum kita memulai
              memasak",
            "AnswerA": "Peralatan",
            "AnswerB": "Bahan",
            "AnswerC": "Resep",
            "AnswerD": "Tidur",
            "CorrectAnswer": "Tidur",
            "ExplanationAnswer": "Tidur tidak termasuk daripada
              apa yang harus kita siapkan ketika mice an place"
        }
    ],
    "status": 200
```

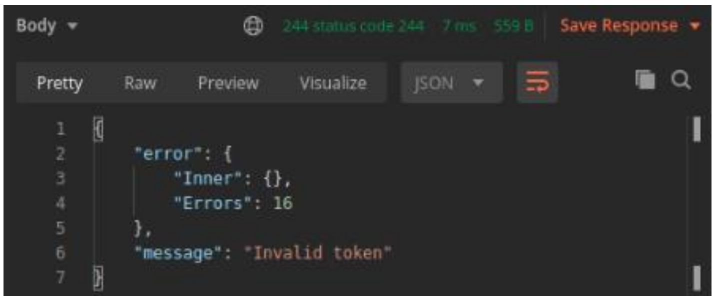
Figure 11. Request data with GET method accompanied by parameters



```
{
    "message": "record not found",
    "status": "error"
}
```

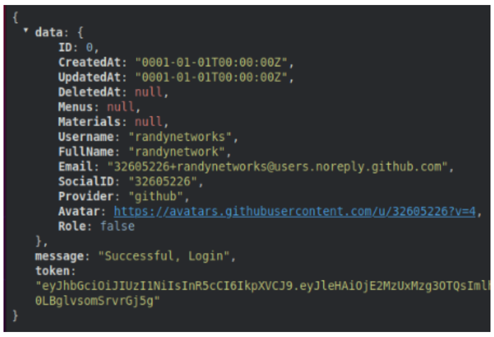
Figure 12. Request data with the GET method accompanied by parameters that do not exist

The use of requests on HTTP, whether it is data contained in the database or not, is handled properly. Both Figure 11 and Figure 12, in the form of JSON and each HTTP 200 code for available data, and HTTP 404 code for data are not available. The time it takes to request available data is 2.08 s and for unavailable data, it takes 608 ms.



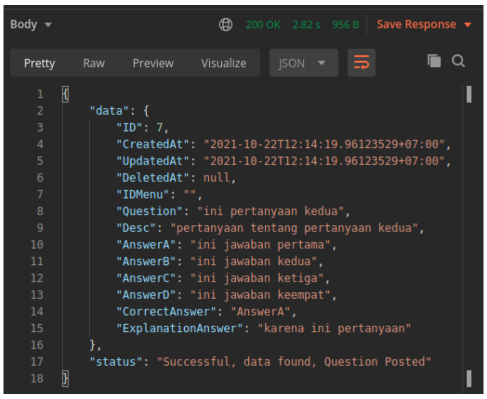
```
{
    "error": {
        "Inner": {},
        "Errors": 16
    },
    "message": "Invalid token"
}
```

Figure 13. Request data with POST method without OAuth



```
{
  ▾ data: {
      ID: 0,
      CreatedAt: "0001-01-01T00:00:00Z",
      UpdatedAt: "0001-01-01T00:00:00Z",
      DeletedAt: null,
      Menus: null,
      Materials: null,
      Username: "randynetworks",
      FullName: "randynetwork",
      Email: "32605226+randynetworks@users.noreply.github.com",
      SocialID: "32605226",
      Provider: "github",
      Avatar: https://avatars.githubusercontent.com/u/32605226?v=4,
      Role: false
  },
  message: "Successful, Login",
  token:
  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MzUxMzg3OTQsIml
  0LBglvsomSrvrGj5g"
}
```

Figure 14. Request OAuth



```
{
    "data": {
        "ID": 7,
        "CreatedAt": "2021-10-22T12:14:19.96123529+07:00",
        "UpdatedAt": "2021-10-22T12:14:19.96123529+07:00",
        "DeletedAt": null,
        "IDMenu": "",
        "Question": "ini pertanyaan kedua",
        "Desc": "pertanyaan tentang pertanyaan kedua",
        "AnswerA": "ini jawaban pertama",
        "AnswerB": "ini jawaban kedua",
        "AnswerC": "ini jawaban ketiga",
        "AnswerD": "ini jawaban keempat",
        "CorrectAnswer": "AnswerA",
        "ExplanationAnswer": "karena ini pertanyaan"
    },
    "status": "Successful, data found, Question Posted"
}
```

Figure 15. Request data with POST method with Oauth

The use of OAuth certainly affects the response server; the server will read the authentication of the request header. In Figure 14, in Request without OAuth, the server will respond to an invalid token with the form of JSON and HTTP code 244, and when we request the token in Figure 15 and enter the token in the header request, then in Figure 11 in the server explained responded correctly. The entire response server above is collected in detail in the Table 1.

Table 1. Black Box Rest API Test Results on Postman

No.	*Endpoint*	*Information*	Time Used	Ouput	Result
1	*/questions*	*Request data*	417ms	GET HTTP JSON	Succeeded with response 200 OK
2	*/quiz/mice-and-place*	*Request data*	2.08 ms	GET HTTP JSON with correct parameters	Succeeded with response 200 OK
3	*/quiz/mice-and-places*	*Request data*	608 ms	GET HTTP JSON with wrong parameters	Succeeded with response 404 Not Found
4	*/admin/question*	*Request data*	7 ms	POST HTTP JSON with Header without authentication	Succeeded with response 244
5	*/admin/question*	*Request data*	2.82 s	POST HTTP JSON Header authentication	Succeeded with response 200 OK

4. Conclusion

To solve the problem of multi-platform devices that are not synchronized with servers, an intermediate bridge is needed to connect servers and many devices or multi-platforms, namely by using the REST API architecture. The use of REST APIs has three important elements: the use of endpoints to access rest APIs by platforms, functionality to overcome response and request logic, and authentication to protect the manipulation of sensitive data. If all three elements are met, the REST API is following international standards. Rest API architecture testing on multi-platform devices can be tested well by the architectural analysis conducted, so it can be proven that rest API is good for multi-platform software development.

References

[1] M. A. K. Perdana, "PENGEMBANGAN REST API LAYANAN PENYIMPANAN MENGGUNAKAN METODE RAPID APPLICATION DEVELOPMENT (STUDI KASUS: PT. XYZ)."

[2] E. Yanti, Sari Noorlima; Rihyanti, "Penerapan Rest API untuk Sistem Informasi Film Secara Daring."

[3] Y. Syahidin, "Arsitektur Sistem Informasi Government To Government (G2G G) Perencanaan dan an Penganggaran Barang Milik Daerah dengan engan Metode Unified Software Development Process," J. Tek. Inform. Dan Sist. Informasi, 2(1). https//doi.org/10.28932/jutisi.v2i1.610, vol. 2, no. April, pp. 75–88, 2016.

[4] J. Herlian, "Perancangan Sistem Mobile POS (Point of Sale) Dengan Menggunakan Restful Web Services," 2015.

[5] F. Doglio, REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development. 2018.

[6] B. B. & A. H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and JavaTM Third Edition, vol. 821 LNCS. 1994.

[7] M. Rahmani, SOFTWARE MODELING AND DESIGN UML, Use Cases, Patterns, and Software Architectures, vol. 36, no. 4. 2011.

[8] T. Kristanto, R. K. Hapsari, V. S. Nita, and S. Maimunah, "Rancang Bangun Aplikasi E-Learning Berbasis Multiplatform untuk Mata Pelajaran Bahasa Indonesia dengan Menggunakan Pendekatan Technology Acceptance Model (TAM)," J. Tek. Inform. dan Sist. Inf., vol. 1, no. 3, 2015, doi: 10.28932/jutisi.v1i3.408.

[9] A. O. Approach, SYSTEMS ANALYSIS & DESIGN An Object-Oriented Approach with UML.

[10] R. S. Wazlawick, "Object-Oriented Analysis and Design for Information Systems," Object-Oriented Anal. Des. Inf. Syst., 2014, doi: 10.1016/c2012-0-06942-6.

[11] M. Mark, REST API Design Rulebook, vol. 53, no. 9. 2013.

[12] P. C. Jorgensen, Software testing: A craftsman's approach, third edition. 2013.

[13] D. Westerveld, API Testing and Development with Postman with Postman.

[14] J. Eckroth, Python Artificial Intelligence Projects for Beginners : Get up and Running with Artificial Intelligence Using 8 Smart and Exciting AI Applications. Packt Publishing, 2018.

[15] K. A. H. Booch, Grady; A.Maksismchuk, Robert; W.Engle, Michael., J.Young, Bobbi; Conallen, Jim, Object-Oriented Analysis And Design with Applications - Third Edition.