Journal of Engineering Science and Technology
Vol. 19, No. 2 (2024) 447 - 455
© School of Engineering, Taylor's University

IMPLEMENTATION OF GRAPHQL IN THE DODO KIDS BROWSER APPLICATION

ADAM MUKHARIL BACHTIAR*,
DIAN DHARMAYANTI,DIMAS MIFTAHUL HUDA

Faculty of Engineering and Computer Science, Universitas Komputer Indonesia, Indonesia
*Corresponding Author: adam@email.unikom.ac.id

Abstract

This study aims to implement GraphQL to address the issues of under-fetching and over-fetching in the Dodo application. Dodo Kids Browser (Dodo) is a cross-platform application that serves as a parental control tool. This application assists parents in managing and monitoring their children's online activities. The initial testing revealed problems with the backend of the Dodo application, which utilizes the REST API. Specifically, under-fetching and over-fetching were identified as issues that negatively impact performance and the application development process. After a comprehensive review of the existing literature, implementing GraphQL can serve as a viable solution. This solution is primarily due to the functions of GraphQL as a query language, enabling clients to precisely determine the specific data requested from the server. A series of tests were conducted to assess this implementation's effectiveness. These tests involved comparing the required data attributes specified by the client with the actual data attributes transmitted from the server. Additionally, tests were performed to evaluate the performance of the previous and current systems. Upon analysing the results of these tests, it was discovered that GraphQL effectively resolves the issues of under-fetching and over-fetching that commonly arise in the Dodo Kids Browser application.

Keywords: Indonesian sign language system, Learning media, Learning mode, Question mode, Sign language.

1. Introduction

Dodo Kids Browser (Dodo) is an application that operates across multiple platforms and serves as a parental control mechanism. This particular application assists parents in effectively managing and monitoring their children's online activities. Dodo encompasses three primary functionalities, namely Surfior, Notifior, and Reportior. Currently, Dodo is compatible with the Windows Phone platform and Browser Extensions [1]. In addition, Dodo is also being expanded to include Desktop, Android, and iOS platforms to cater to a broader range of users. An Application Programming Interface (API) has been developed to address these requirements using REST technology [2].

REST is widely regarded as the most popular API technology due to its numerous advantages. These advantages include the availability of multiple response types of options, the relative ease of implementation, and the clear separation between client and server [3]. However, it is essential to acknowledge that REST also has significant drawbacks, namely under-fetching and over-fetching [4]. Under-fetching occurs when the server receives a request from the client but cannot provide all the required data, necessitating an additional request as a supplement. This occurrence can increase latency, causing users to experience longer loading times for successful page rendering. Additionally, the program's complexity is heightened as the client must make multiple requests for a single data requirement.

The issue of under-fetching in REST can be resolved by incorporating an endpoint that caters to the client's specific requirements. However, this course of action gives rise to additional complications. As the application expands, the complexity of the code escalates when multiple instances of code with similar functionalities are written. Furthermore, the progress of the client-side development process is impeded as it is postponed until the API completes its updating procedure [4].

In 2015, Facebook released an API technology named GraphQL [5]. GraphQL is a query language that enables the client to specify the data that the server needs to request. This adaptability is anticipated to address under-fetching and over-fetching commonly encountered in REST. In this study, a GraphQL implementation was conducted to mitigate the problems of under-fetching and over-fetching in the Dodo application after the preliminary testing of the running application. By incorporating GraphQL, it is expected that the performance of the Dodo application can be enhanced.

2. Related Works

GraphQL is a query language for implementing web service architectures. The language was internally developed at Facebook to solve several problems they faced when using standard architectural styles, such as REST. In 2015, Facebook open-sourced the definition and implementation of GraphQL. As a result, the language started gaining momentum and is now supported by significant Web APIs, including those provided by GitHub, Airbnb, Netflix, and Twitter. In December 2018, Facebook transferred GraphQL to a non-profit organization called GraphQL Foundation [6].

GraphQL is an alternative to REST-based applications. To understand GraphQL's differences from REST, it should be noted that endpoints are the key abstraction provided by REST. In REST, an endpoint is defined by a URL and a list of parameters. For example, in GitHub's REST API [6].

GraphQL can diminish the dimensions of JSON documents produced by REST-based APIs by 94%, as measured by the number of fields contained. Additionally, the reduction in size amounts to an impressive 99%, as determined by the number of bytes. These measurements, representative of the median values, were obtained through a comprehensive study. This study involved executing 24 queries, which seven open-source clients performed. These clients were employed to interact with two widely-used REST APIs, namely GitHub and arXiv. Furthermore, an additional 14 queries were conducted by seven recent empirical research papers presented at two distinct software engineering conferences [7].

3. Method

3.1. Analysis of case domain

At this stage, an analysis of the application backend is carried out thoroughly by referring to the functionality of the application and the running web service API. Then, under-fetching and over-fetching analysis is carried out to see how big the problem will be solved by implementing GraphQL on the new system.

The application used as the case study is the Dodo Kids Browser (Dodo) application, an original application developed by the developers from UNIKOM CodeLabs. This application aims to help parents monitor and control their children's activities so that they can browse the internet safely. The backend of the running application is built using REST API technology and the Python programming language with the Flask framework. The main features of the Dodo application are as follows:

(i) Surfior: The Surfior feature is used to monitor children's browsing activities. After a parent registers their child's device on the Dodo application, any browsing and internet searches the child performs will be intercepted and identified as to whether each activity is safe.

(ii) Notifior: If a child's browsing activity monitored by the Surfior feature is indicated to be unsafe, such as accessing adult sites or searching with keywords that are not suitable for their age, Dodo will send a notification to the parent's device. The notification contains the URL of the site and a warning message that the child is suspected of accessing a harmful site. Parents can choose actions to allow or deny site access and send advice through Notifior's feature.

(iii) Reportior: All browsing and search activities from the registered child's device will be recorded in the Dodo application and presented on the dashboard. The data can be used as evaluation material and one of the reference materials for taking action for parents and children to enjoy a safer internet.

3.2. Web services API analysis

At this stage, a Web Services API analysis is carried out on the old system to find all endpoints from the Dodo application backend and identify weaknesses in the API. The analysis results are then compared with the application's needs according to the functional endpoint. Adjustments will be made based on the results of these comparisons.

Based on the results of the analysis, three functionalities still need to be implemented, namely View Dashboard, View Search Reports, and Browse Safely. These shortcomings certainly make the Dodo application unable to achieve its goals perfectly. For this reason, a new endpoint design will be carried out to meet the needs of functions that still need to be implemented and complement existing endpoints, such as Update and Delete for the Manage function Child User Data. This design is to be done in conjunction with the GraphQL implementation.

3.3. Under-fetching and over-fetching analysis

Under-fetching analysis assesses whether the server can meet the client's data needs in one request. The problem of under-fetching is often found in endpoints with GET methods in the REST API, so clients who consume the API need to make more than one request to fulfil one data need.

Over-fetching analysis is carried out to assess whether the data sent from the server is in accordance with the client's needs and nothing more. If there is an excess of data, it is undoubtedly inefficient because the client downloads data that is not needed. The large payload size will be a problem, especially on mobile platforms that expect a responsive system with low latency. Like under-fetching, this over-fetching problem is often found on endpoints with GET methods in the REST API.

Under-fetching and over-fetching testing compare the data requirements outlined in the application interface design with the data transmitted from the server through the endpoint responsible for fulfilling those requirements [8]. Instances of under-fetching occur when the attributes of the data received from the server are insufficient to fulfil the overall data attributes required by the client [9]. Conversely, over-fetching occurs when the attributes of the data received from the server exceed the necessary data attributes on the client [9]. This testing was performed on all API endpoints of the GET type, with the data requirements visible in the application interface design. A total of 12 endpoints were included in this test. An illustrative example of a comparison performed is provided in Table 1.

The examinations determined that five endpoints, or approximately 42% of the 12, were identified by implementing the GET method. Conversely, over-fetching scenarios were observed in all 12 endpoints, constituting 100% of the total endpoints utilizing the GET method.

Table 1. Under-fetching and over-fetching analysis.

View Name	Endpoint Name	Required data	Data sent	Under fetching	Over fetching
Log Activity - Accessed Web	Reportion – Get by Id Child	URL, date, type	child, date, id, status, web description, web title, web_url	Not	Yes
Activity Log – Search	Reportion – Get by Id Child	keyword, date, type	child, date, id, status, web description, web title, web URL	Yes	Yes
List of Child	User Child – Get All	name	id, name, parent parent id, parent_parent_name	Not	Yes

3.4. GraphQL deployment analysis

The analysis of GraphQL applications involves the examination of current application backends in order to ascertain the requirements of the application and identify areas that require modifications for enhancing application performance. Activities conducted during this process entail data analysis and the utilization of web service APIs from the previous system, which will subsequently serve as a point of reference for implementing GraphQL.

3.4.1. Data analysis

At this stage, the performance data analysis is conducted to ascertain the data composition of the backend of the operational Dodo application and detect any deficiencies in the data composition. The analysis process employs two forms of data modelling. The Logical model investigates the interconnections between tables, and the Physical model discerns the intricacies of each table, including the type employed [10].

3.4.2. Designing a new web service API

A novel Application Programming Interface (API) Web Service is formulated at this phase, incorporating GraphQL. The execution entails the creation of a Schema, Query, and Mutation by referencing recent data derived from analysis in the preceding phase [11, 12]. Schema is a data structure yielded via queries and mutations [13]. Below is an illustration of an ongoing schema design process in Table 2.

Table 2. Designing the parent device scheme.

No.	Field	Type	Nullable	Description
1	id	Int	Not	Id for parent user's device data
2	name	String	Not	The name of the parent user's device

A query corresponds to the GET method in the REST API, which retrieves data from the server. The ensuing example delineates the process of formulating a query and executing it.

Table 3. Query designing.

Name	Param	Param Type	Response Type	Description
GetParents	id	Int	[Parent]	Query to get parent user data
GetParentDevices	id, parentId	Int, Int	[Parent Device]	Query to get parental device data
Get Childs	id, parentId	Int, Int	[Child]	Query to get child user data

Mutation is a technique utilized for executing data manipulation on the server, with its equivalent operations in the REST API being POST, PUT, PATCH, and DELETE [14]. Presented below is an illustration of a mutation design that is implemented.

Table 4. Mutation design.

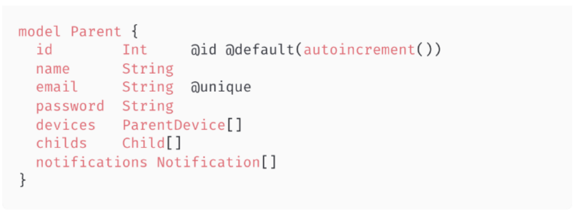
Name	Param	Param Type	Response Type	Description
Register Parent	name, email, password	String! String! String!	AUTH Payload	Mutatation to add parent user data
Login Parent	email, password	String! String!	AUTH Payload	Mutation to enter the system as a parent
Update Parent	email, oldPassword, newPassword	String! String! String!	String	Mutatation to change parent user settings

4. Results

This section examines the execution of the system based on the analysis that has been conducted. The execution procedure is categorized into three components, namely database execution, GraphQL execution, and documentation.

4.1. Database implementation

The Database implementation uses PostgreSQL RBMS and Prisma as the (Object Relational Model) ORM, using a Prisma that uses a Code-First approach. The database can be modelled using the Code-First approach in the form of code and becomes easily accessible in system programs [15]. In addition, implementing Prisma also supports database migration, accelerating changes or creations that occur in the database [16]. Here is the Prisma ORM implementation code for each table from the previous analysis. The Prism ORM implementation for the Parent table is shown in Fig. 1.



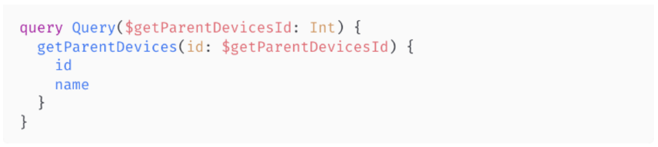
```
model Parent {
  id         Int     @id @default(autoincrement())
  name       String
  email      String  @unique
  password   String
  devices    ParentDevice[]
  childs     Child[]
  notifications Notification[]
}
```

Fig. 1. Parent table ORM implementation.

4.2. GraphQL implementation

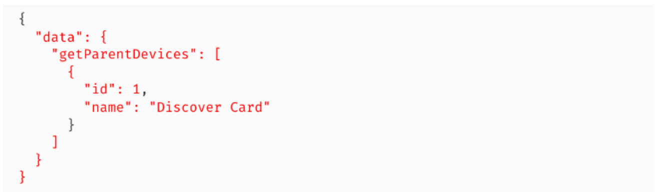
At this phase, an explanation is provided regarding the execution and specifics of each schema, query, and mutation present in the novel framework [17]. The novel framework is executed employing the TypeScript programming language and the Apollo Server Framework, which aids in implementing a web server based on GraphQL.

The subsequent passage presents an illustration of a query implementation, specifically the getParentDevices Query, derived from the preexisting design outcomes (see Figs. 2 and 3).



```
query Query($getParentDevicesId: Int) {
  getParentDevices(id: $getParentDevicesId) {
    id
    name
  }
}
```

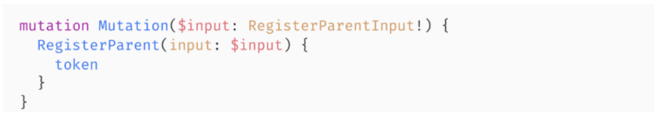
Fig. 2. Query invocation getparentdevices.



```
{
  "data": {
    "getParentDevices": [
      {
        "id": 1,
        "name": "Discover Card"
      }
    ]
  }
}
```

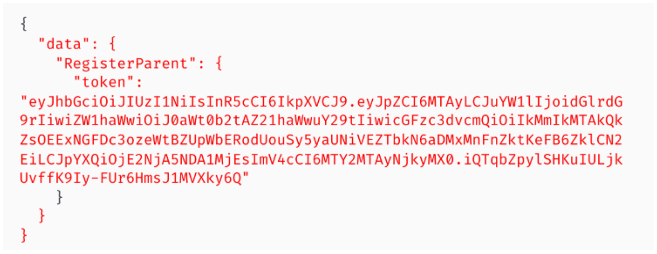
Fig. 3. GetParentDevices query response.

The subsequent is an illustration of a query implementation, specifically the getParentDevices Query predicated on the outcomes of the design that has been previously conducted (see Figs. 4 and 5).



```
mutation Mutation($input: RegisterParentInput!) {
  RegisterParent(input: $input) {
    token
  }
}
```

Fig. 4. RegisterParent mutation invocation.



```
{
  "data": {
    "RegisterParent": {
      "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTAyLCJuYW1lIjoidGlrdG
9rIiwiZW1haWwiOiJ0aWt0b2tAZ21haWwuY29tIiwicGFzc3dvcmQiOiIkMmIkMTAkQk
ZsOEExNGFDc3ozeWtBZUpWbERodUouSy5yaUNiVEZTbkN6aDMxMnFnZktKeFB6ZklCN2
EiLCJpYXQiOjE2NjA5NDA1MjEsImV4cCI6MTY2MTAyNjkyMX0.iQTqbZpylSHKuIULjk
UvffK9Iy-FUr6HmsJ1MVXky6Q"
    }
  }
}
```

Fig. 5. RegisterParent mutation response.

4.3. System documentation

Systems were constructed, deployed, and recorded using the GraphQL Playground tool. Using the GraphQL Playground tools, all schemas, queries, and mutations that have been constructed will be automatically recorded and made visible to the general public. Additionally, developers can query the Query Page and observe the results directly on the Response Page [18].

In order to prevent unauthorized access to data, the majority of queries and mutations necessitate user authentication prior to making requests. It is possible to request all queries and mutations at the same endpoint, namely GraphQL.

5. Discussion

Upon the culmination of the implementation process of the novel system, additional examination is conducted to assess the triumph of this investigation. The examinations conducted encompass under-fetching and over-fetching tests aimed at scrutinizing the persistence of the predicament within the Dodo Kids Browser application. The evaluation of the novel system is performed in a manner identical to the analysis of under-fetching and over-fetching conducted on the previous system. This process involves examining the display applications' data requirements and the server's data transmitted to fulfil those requirements. Some outcomes derived from these experiments can be found in Table 5.

Table 5. Under-fetching and over-fetching testing.

View Name	Query Name	Required data	Data sent	Issues found	Less/unnecessary data
Log Activity - Accessed Web	getLogActivities	URL, date, type	webUrl, date, category	None	None
Activity Log – Search	getLogActivities	keyword, date, type	keyword, date, category	None	None
Notification	getLogActivities	web_url, status	webUrl, category	None	None

The test results show that the new system did not encounter any issues related to under-fetching and over-fetching. This result demonstrates the successful resolution of the problem through the implementation of GraphQL. GraphQL plays a crucial role in website development by providing a flexible interface for data retrieval, enabling developers to design requests tailored to the specific needs of web pages. With GraphQL, optimizing data retrieval at the server level can enhance the performance and responsiveness of the website, delivering a more efficient and user-centric experience tailored to the requirements of each request [19].

6. Conclusion

From the outcomes of conducted tests on the novel system, the incorporation of GraphQL can potentially address the dilemmas of under-fetching and over-fetching in the Dodo Kids Browser application. In order to advance this study, the efficiency of both REST-based and GraphQL-based systems can be assessed, thereby allowing us to gauge the extent to which the resolution of under-fetching and over-fetching complications in the Dodo application impacts its overall performance.

References

1. Bachtiar, A.M.; and Sukirman, I.I. (2015). Pembangunan perangkat lunak extension browser pada aplikasi pengawasan penggunaan internet anak “Dodo kids’ browser.”. *Jurnal Ilmiah Komputer dan Informatika (KOMPUTA)*, 6(1), 45-50.
2. Roziqin, M.C.; Noor, M.S.; Iskandar, A.; and Yuliantika, A. (2023). Implementation of REST API in web service system for medical resume provision. *International Journal of Healthcare and Information Technology*, 1(1), 34-48.
3. Ogboada, J.G.; Anireh, V.I.E.; and Matthias, D. (2021). A model for optimizing the runtime of GraphQL queries. *International Journal of Innovative Information System*, 9(3), 11-39.
4. Chaves-Fraga, D.; Priyatna, F.; Alobaid, A.; and Corcho, O. (2020). Exploiting declarative mapping rules for generating GraphQL servers with morph-GraphQL. *International Journal of Software Engineering and Knowledge Engineering*, 30(06), 785-803.

5. Mikuła, M.; and Dzieńkowski, M. (2020). Comparison of REST and GraphQL web technology performance. *Journal of Computer Sciences Institute*, 16(1), 309-316.
6. Margański, P.; and Pańczyk, B. (2021). REST and GraphQL comparative analysis. *Journal of Computer Sciences Institute*, 19(1), 89-94.
7. Quiña-Mera, A.; Fernandez, P.; García, J.M.; and Ruiz-Cortés, A. (2023). GraphQL: A systematic mapping study. *ACM Computing Surveys*, 55(10), 1-35.
8. McGuinness, D.L.; and Da Silva, P.P. (2004). Explaining answers from the semantic web: The inference web approach. *Journal of Web Semantics*, 1(4), 397-413.
9. Guha, S. (2020). A comparative study between graph-ql & restful services in api management of stateless architectures. *International Journal on Web Service Computing (IJWSC)*, 11(2), 1-15.
10. Tinambunan, D.H.; Baehaqi, A.; Avrianto, R.P.; and Indrajit, R.E. (2023). Microgen implementation for building online learning management system with microservices and GraphQL generator approach. *Jurnal Teknik Informatika (JUTIF)*, 4(4), 967-976.
11. Qi, L.; Song, H.; Zhang, X.; Srivastava, G.; Xu, X.; and Yu, S. (2021). Compatibility-aware web API recommendation for mashup creation via textual description mining. *ACM Transactions on Multimidia Computing Communications and Applications*, 17(1s), 1-19.
12. Chaves-Fraga, D.; Priyatna, F.; Alobaid, A.; and Corcho, O. (2020). Exploiting declarative mapping rules for generating GraphQL servers with morph-GraphQL. *International Journal of Software Engineering and Knowledge Engineering*, 30(6), 785-803.
13. Silva, D.C.; Abreu, P.H.; Reis, L.P.; and Oliveira, E. (2017). Development of flexible languages for scenario and team description in multirobot missions. *AI EDAM*, 31(1), 69-86.
14. Tedyyana, A.; Ghazali, O.; and Purbo, O.W. (2023). A real-time hypertext transfer protocol intrusion detection system on web server. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 21(3), 566-573.
15. Panico, F.; Fleury, L.; Trojano, L.; and Rossetti, Y. (2021). Prism adaptation in M1. *Journal of Cognitive Neuroscience*, 33(4), 563-573.
16. Xu, J.; Pan, L.; Zeng, Q.; Sun, W.; and Wan, W. (2023). Based on TPUGRAPHS predicting model runtimes using graph neural networks. *Frontiers in Computing and Intelligent Systems*, 6(1), 66-69.
17. Gaebel, J.; Keller, J.; Schneider, D.; Lindenmeyer, A.; Neumuth, T.; and Franke, S. (2021). The digital twin: Modular model-based approach to personalized medicine. *Current Directions in Biomedical Engineering*, 7(2), 223-226.
18. Purwanto, D.D.; Honggara, E.S.; Tjandra, S.; Ardhi, S.; and Tjoa, N. (2023). Pengembangan aplikasi human resource management pada pt. HJMB menggunakan JS, react native, dan graphQL. *Journal of Information System, Graphics, Hospitality and Technology*, 5(2), 95-101.
19. Veach, A.M.; and Abualkibash, M. (2022). Phishing website detection using several machine learning algorithms: A review paper. *International Journal of Informatics, Information System and Computer Engineering (INJIISCOM)*, 3(2), 219-230.