IMPLEMENTING GRAPHQL AND REST APPLICATION PROGRAMMING INTERFACES IN MOBILE APPLICATIONS

Avni RUSTEMI[1]

[1*]*Department of Informatics, Faculty of Natural Sciences and Mathematics, University of Tetovo, North Macedonia*
[*]*Corresponding author e-mail:avni.rustemi@unite.edu.mk*

Abstract

Mobile applications are increasingly being used in many areas of life, where most systems, in addition to being created for desktops, are also adapted for mobile and other smartphones, because people spend most of their time on mobile devices, and it is easier for them to perform their daily tasks. Of particular importance during the development of modern mobile applications is the choice of API architecture, which directly affects performance, efficiency and easier surfing for clients. The two main approaches most used are REST API and GraphQL, which offer different ways of accessing data and interacting between clients and servers. Initially, through the paper, we are introduced to the way these two architectures work, describing the advantages and disadvantages of each of them, and then we continue with experimental analysis and description of some code sequences that we used in our experiment, both during the implementation of the REST API and GraphQL, on the client side and server side. The results of our testing and practical implementation also reinforce the positions of other research, that GraphQL is much better when it comes to executing more complex queries and time efficiency, while REST API is better when we have simpler and more stable requests. We concluded the paper by describing future trends in mobile APIs, with particular emphasis on hybrid architectures and integration with edge computing and serverless technologies.

Keywords: Application Programming Interfaces, REST API, GraphQL, architecture, future trends, mobile AP

Introduction

Mobile application development is also closely linked to Application Programming Interfaces (APIs), which, among other things, provide easy and fast communication between the mobile application and the server, but also in the construction of microservices and distributed systems, facilitating real-time functionalities and reuse by different mobile applications. Today, most users of the network and modern technologies are connected through mobile applications, as they have become an inseparable part of people in every sphere of life, and on the other hand, it is a current challenge for developers because they must adapt the architecture of mobile systems to the requirements of users but also to the comfort of the development of contemporary technology. Despite the complicated architectures and numerous functionalities they offer, the performance of mobile applications must come first, because response time is one of the phenomena that directly affects the user experience and usability of the mobile application in various fields (Liu & Li; 2022). Despite the fact that the performance of mobile applications depends on the network and the programming of functionalities in mobile applications, recently most mobile applications provide services obtained from APIs and this dependence often makes communication a critical factor for the efficiency, scalability and real-time performance of mobile applications (Akamai Technologies, 2022). APIs define how client-server requests are processed and how responses are structured, enabling the integration of multiple microservices into a single application, integrating with databases, cloud services, and other external systems, providing scalability and efficiency (Alozie et al 2025). In addition to APIs, the network delay is also of great importance for the performance of mobile applications, since it affects the time of data processing from the server, sometimes it also

includes the time of data transmission through communication channels. Bandwidth, payload size, throughput also affect the performance of mobile applications (Haider et al, 2026).

Representational State Transfer (REST) has been one of the dominant standards for a long time, thanks to its simplicity, HTTP methods, and stateless architectures, where the server does not store client data after the session ends, and any request can be processed by available servers, making it ideal for both microservices and cloud applications. GraphQL, a new architecture, initially presented by Facebook, is being used more recently for building APIs, because unlike REST, it uses the client-driven model, where the client determines in advance the data it needs, and processing in this way becomes more unburdened and reduces problems known as over-fetching and under-fetching, which in fact represent the lack or excess of data that the client receives, directly affecting the bandwidth of the API (Kanthed; 2023). GraphQL has the ability to reduce unnecessary data traffic, significantly improving efficiency and bandwidth, which directly affects the performance of mobile applications, especially when services are integrated as APIs (Quiña-Mera et al. 2022). The choice between GraphQL and REST varies and depends on many factors, however REST continues to be very widespread due to caching, simplicity, and especially for CRUD (Create, Read, Update, and Delete) operations (Iyer; 2026). However, GraphQL is more suitable when it comes to processing data from multiple sources, but to process it with a single request, which reduces the response time and improves the performance of executing client requests (Hakia; 2025). Figure 1 graphically presents the main difference between REST and GraphQL API, where, among other things, it is clearly seen that in the REST API, the client makes many requests and each of them is processed as a separate endpoint and each of them returns data from certain databases, which often result in missing or overloaded data. Whereas in GraphQL the client processes a request to the GraphQL endpoint, and the server returns the correct response to the request made, requesting data from all data structures, thereby reducing problems with over-fetching and communication channels in networks.

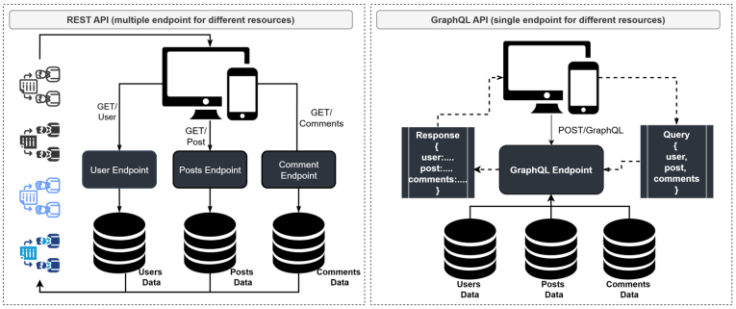


Figure 1. Visual comparison between REST API and GraphQL API

Background and related work

Recent research has shown an interest in the use and comparison between REST and GraphQL in the context of mobile applications and decentralized systems. However, each one has its importance depending on the context and the requests of the clients. In the analysis of Stępień & Skublewska-Paszkowska (2025), it is found that GraphQL reduces the size of the payload and the number of requests in the network, because the same are processed in a more organized form, and that there is no need for a client to make more than

one request, even if he needs services in different data structures, although again REST results faster in the execution of requests that are simpler and structured direct. Niswar et al (2024), studies the use of system resources in the execution of requests, and among other results and analyzes conclude that GraphQL is used and requires more CPU in the processing of complex queries, which depends on the client, if there are more requests for processing, while REST does not require a powerful CPU, because requests are processed one by one, and suggests that REST is more suitable when it comes to operational management in a more efficient way, while GraphQL if flexibility and better organization are required. Jin et al. (2024) investigates the performance in serverless architectures, where they come to the conclusion that GraphQL reduces the number of network requests and once provides the response in time faster than REST (that is, the final response with several client requests), and they support the position that GraphQL is more suitable for mobile applications with dynamic interfaces, where the number of requests is different and of different natures.
Recently, the migration from REST to GraphQL has been of great importance, where in the research of Amareen et al. (2026), a framework called GraphQLify is presented, which in automatic form by providing static analysis of the code offers migration, a phenomenon that significantly allows performance and response time reduction. The reason why performance is permitted is that only the necessary data is fetched, there is no network overhead, it avoids bugs during migration, it guarantees type mismatch, and out of 834 tested APIs, the study has shown high success. Through Table 1, we have compared these two architectural styles for creating mobile applications, based on the most important characteristics such as endpoints, flexibility, scalability, performance, and others.

Table 1. Comparison of key features between REST API and GraphQL API

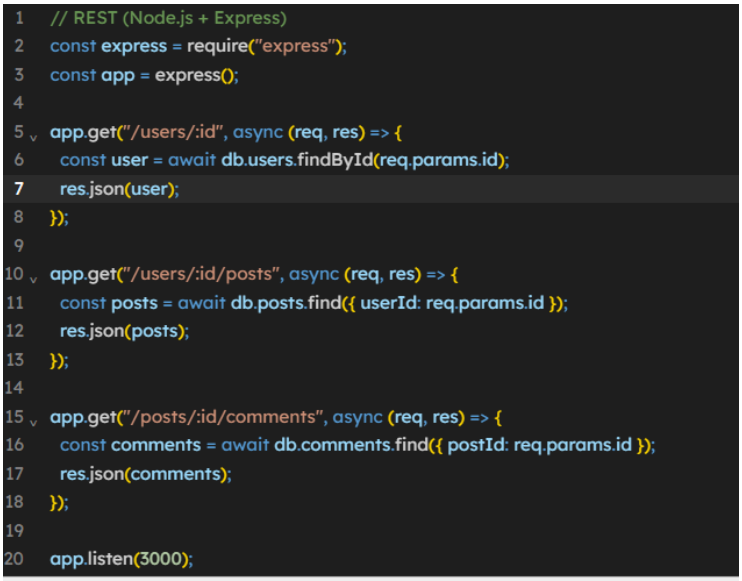
Characteristics	**REST API**	**GraphQL API**
Endpoints	Uses multiple endpoints, for each client request from one endpoint	One endpoint for further client requests, collects all data and returns responses
Data processing	The response is predetermined by the server, there may be a lack of data or an excess	The data is specified depending on the request and the answer is concrete
Number of requests	It processes multiple requests to retrieve more data from multiple data structures	It processes a query (multiple requests) and the response is unified from all data structures
Caching	It uses standard HTTP mechanisms, with less traffic, lower latency and longer battery life	Use POST /graphql, the queries are dynamic, use object IDs
Server	Simple maintenance, without any complex architecture and design	It requires more complex architectures, schemes for efficient query management
Performance in mobile applications	Good for direct and not many requests, fast and very simple in execution	Reduction of bandwidth, optimization of requests, in complex scenarios, more effective and faster
Scalability	Implemented in a longer time, stable and practically	It requires careful optimization, clearly

	proven for scalability, despite the disadvantages in processing	defined schemes for query processing, good practical results
Suitability for mobile applications	Simple mobile applications	Complex mobile applications, and with dynamic requirements

Despite the advantages and disadvantages of both paradigms, it is worth emphasizing that features such as security, privacy, secure query management, transparency, decentralization, and even immutability and integrity preservation remain as issues of particular importance for study. These features are only achieved by implementing these APIs with blockchain systems (Rustemi et al. 2023). In these blockchain systems, REST is mainly used for simple processing and transaction mediation, while GraphQL is used for more complex queries of data stored in blockchain storage.

Experimental analysis of REST and GraphQL API in mobile applications

To do an analysis between REST API and GraphQL, we have created a prototype with two separate services on the same dataset. The dataset contains 1000 users, 10 000 posts and 50 000 comments stored in JSON (MongoDB). The goal is simple, to just get data from users, posts and comments, and at the same time measure performance metrics. The measurements were performed using a test script, which generates approximately 100 requests for each scenario, with 10 parallel users, in a local environment with a latency of approximately 80ms to adapt to mobile conditions. For each API, the average response time, percentiles, payload size, number of requests per operation, throughput, among others, were recorded. From the results we have summarized, it follows that REST requires an average of 3 requests for an operation, with an average response time of 471ms (median around 452ms, percentiles 538ms), and a total payload of around 129KB. On the other hand, GraphQL performs the same operation, making only a single request, with an average time of 219ms (median 205ms, percentiles 262ms) and a payload of around 72KB. This means that GraphQL has the best performance and time with almost the same result. To test both the complexity and simplicity of REST versus GraphQL, a simpler scenario was also tested, just the user profile, where REST performed slightly better than GraphQL, 120ms versus 140ms. The over-fetching property of REST was also proven, where REST, in addition to sending fixed JSON structures, resulted in sending about 44% of the data capacity per operation, which directly affects mobile networks and bandwidth. Also using Prometheus, an open source system, CPU usage was monitored, and it turned out that GraphQL uses 20-30% more CPU than REST, which once again proves the accuracy of the research that GraphQL optimizes network and latency, while REST optimizes simplicity and cost. Similar practical results have been concluded by Stępień & Skublewska-Paszkowska (2025), where they conclude, among other things, reduced payload and better results for more complex scenarios, and better REST performance for simpler operations. In Figure 2 we have presented the minimal server in Node.js, respectively the code for implementing the REST API for specific endpoints.



```
// REST (Node.js + Express)
const express = require("express");
const app = express();

app.get("/users/:id", async (req, res) => {
  const user = await db.users.findById(req.params.id);
  res.json(user);
});

app.get("/users/:id/posts", async (req, res) => {
  const posts = await db.posts.find({ userId: req.params.id });
  res.json(posts);
});

app.get("/posts/:id/comments", async (req, res) => {
  const comments = await db.comments.find({ postId: req.params.id });
  res.json(comments);
});

app.listen(3000);
```

Figure 2. Execution of specific queries for 3 different types of requests in the REST API

Figure 3 shows a code sequence showing the client-side, which shows how a mobile application communicates with the REST API to retrieve data. Initially, the request for user retrieval is branched, via id, respectively the number at the end indicates the user's ID. Then the request for retrieving the user's mailing list is sent, and requests for retrieving comments for each post. Through Promise.all, the requests are made in parallel up to the last comments.

```
// Client (REST)
const user = await fetch("/users/1").then(r => r.json());
const posts = await fetch("/users/1/posts").then(r => r.json());
const comments = await Promise.all(
  posts.map(p => fetch(`/posts/${p.id}/comments`).then(r => r.json()))
);
```

Figure 3. Client-side code sequence of requests executed in REST API

Figure 4 shows a code sequence taken from Node.js, implemented in Apollo Server, where a graphql server is created, the data schema is defined and how the same will be retrieved from the server. The endpoint is opened on port 4000, the library where Apolloserver creates the GraphQL server is imported, and the query that is written is actually the input to the API, where the client can request the data later, on the client side.

```
// GraphQL (Apollo Server)
const { ApolloServer, gql } = require("apollo-server");

const typeDefs = gql`
  type User {
    id: ID!
    name: String!
    posts: [Post]
  }
  type Post {
    id: ID!
    title: String!
    comments: [Comment]
  }
  type Comment {
    id: ID!
    text: String!
  }
  type Query {
    user(id: ID!): User
  `;

const resolvers = {
  Query: {
    user: (_, { id }) => db.users.findById(id),
  },
  User: {
    posts: (user) => db.posts.find({ userId: user.id }),
  },
  Post: {
    comments: (post) => db.comments.find({ postId: post.id }),
  },
};

const server = new ApolloServer({ typeDefs, resolvers });
server.listen({ port: 4000 });
```

Figure 4. Server-side code sequence that integrates the GraphQL API

Figure 5 shows a code sequence, where the client sends a request to a GraphQL API and receives data related to a single query. From the code sequence we see that the user with id 1 is requested, and from the same user the name, posts are requested, where in each post the title, comments are requested, and for each comment the text that is stored on the server side is requested. The response received at the end is converted to JSON, which is actually a structured form of data representation.

```graphql
// Client (GraphQL)
const query = `
query {
  user(id: "1") {
    name
    posts {
      title
      comments {
        text
      }
    }
  }
}`;
const data = await fetch("/graphql", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ query })
}).then(r => r.json());
```

Figure 5. Client-side code sequence for executing a query in the GraphQL API

Discussion and future trends in API

Data read and modify operations in REST API and GraphQL differ in the way the client and server interact. In REST, operations are performed through get, post, and put methods, where each endpoint represents a separate resource. This makes REST simpler and more standard for uses where performance and time efficiency are not required. GraphQL uses query and mutation to provide more flexible services, where the client determines the exact structure of the response. In other words, GraphQL offers greater control over data, and reduces the complexity of operations in mobile applications (Nnamdi, 2024). GraphQL performs much better in complex scenarios where complex data is requested from the client, while REST often requires consecutive requests to retrieve data from different sources from the same client, which directly impacts network latency and bandwidth (Jin, 2025). Regarding mutations, REST uses separate queries for CRUD operations, while GraphQL uses mutation as a centralized mechanism, which although it gives greater flexibility, on the other hand increases the risk of uncontrolled complexities and DoS attacks if restrictions and security mechanisms are not implemented (Tsai et al., 2025). Recent research shows increasing adoption of GraphQL in mobile applications, especially those with dynamic interfaces and architectures such as microservices and serverless (Devalla, 2024). In other words, REST is suitable for simpler and more stable systems, since REST implementation is much longer than GraphQL, while GraphQL offers advantages in performance-oriented mobile services. Although there is an attempt to move REST towards GraphQL, both are still used today, creating a hybrid ecosystem where both technologies interact in hybrid environments (Amareen et al., 2026). Figure 6 presents future implementations of APIs, which, among other things, shows the combination of REST, GraphQL and gRPC architectures in one ecosystem, implementation of AI tools for automatic generation, documentation and testing, respectively for intelligent query optimization, real-time monitoring, focus on optimization and bandwidth, and more.

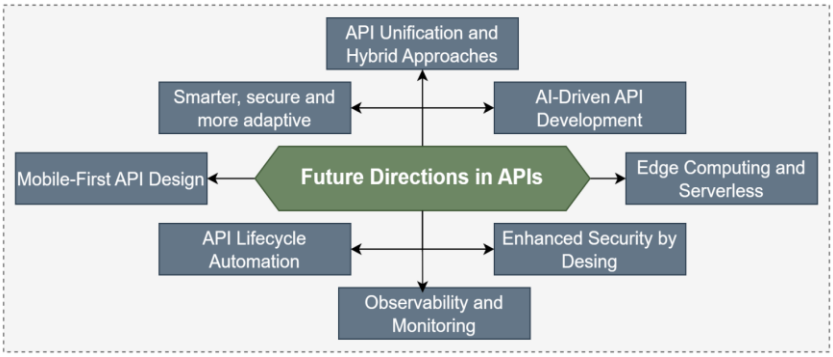


Figure 6. Future trends of implementing APIs

5. Conclusion

The development and combination of contemporary technologies play an important role in creating flexible and more easily integrated ecosystems with the demands of users and technologies in general. The development of APIs and the adoption of hybrid architectures is an important process in adapting to the different requirements of systems that change depending on the circumstances and environment in which they are implemented. REST API and GraphQL offer different models that directly affect how data is managed and protected. REST is characterized by security mechanisms such as authentication, authorization, and caching. GraphQL requires more advanced control mechanisms, input validation, and complexity constraints. In terms of privacy, both face different challenges in terms of processing personal data. GraphQL, because it precisely specifies fields, reduces the risk of accessing private data, but at the same time, even a weak configuration can lead to unauthorized access to data. REST API, because of its flexibility and fixed structure, often offers the risk of over-sharing information. In terms of ethical and legal issues, the design of architectures that use APIs must respect principles such as data transparency, minimization, and control of the information that is generated. GraphQL is considered more ethical because it provides a more ethical and consolidated response compared to REST which can distribute information that is in conflict with the client's ethical issues. Both paradigms offer good opportunities for accessing data through APIs, but also risks that must be carefully managed, and greater ethical awareness in the development of modern systems. Through the paper, in addition to describing the advantages and disadvantages of these two paradigms, we also provided some practical implementation sequences, including experimental analysis to reinforce the evidence that GraphQL has advantages in some aspects compared to REST API, but that both paradigms are still widely used by developers today.

References

[1]. Akamai Technologies. (2022). Unlocking mobile application performance [White paper]. https://www.akamai.com/site/en/documents/white-paper/2022/akamai-unlocking-mobile-application-performance-white-paper.pdf

[2]. Alozie, C. E., Ajayi, O. O., & Chukwurah, N. (2025). Comparative Analysis of APIs and Frameworks in Mobile Application Development: Insights from Industry Practices. International Journal of Academic and Applied Research (IJAAR), Vol. 9 Issue 4 April - 2025, Pages: 97-105.

[3]. Amareen, S., Rahman, A., Rahaman, S., & Bosu, A. (2026). GraphQLify: Automated and Type Safety-Preserving GraphQL API Adoption. arXiv preprint arXiv:2604.15465.

[4]. Devalla, S. (2024). Enterprise-scale evaluation of REST and GraphQL: Balancing performance, scalability, and resource utilization. International Journal of Core Engineering & Management, 7(12), 396-416.
[5]. Haider, S., Hameed, S., Zubair, S., & Saddiqa, A. (2026). Mobile application performance metrics and security integration. Contemporary Journal of Social Science Review, 4(1), 464–472. https://doi.org/10.63878/cjssr.v4i1.2002.
[6]. Hakia. (2025, December). GraphQL vs REST: When to use which. Hakia. https://hakia.com/engineering/graphql-vs-rest/
[7]. Iyer, A. (2026, January 27). GraphQL vs REST — Which is better? DZone. https://dzone.com/articles/graphql-vs-rest-which-is-better
[8]. Jin, R. (2025). GraphQL vs. REST: Performance and Scalability Analysis for Serverless Applications (Master's thesis, University of Washington).
[9]. Jin, R., Cordingly, R., Zhao, D., & Lloyd, W. (2024, December). GraphQL vs. REST: A performance and cost investigation for serverless applications. In Proceedings of the 10th International Workshop on Serverless Computing (pp. 37-42).
[10]. Kanthed, S. (2023). Rest vs. GraphQL: Comparative Analysis of API Design Approaches. International Journal of Multidisciplinary Research and Growth Evaluation, 4(1), 984-991.
[11]. Liu, P., & Li, Y. (2022). Response time evaluation of mobile applications combining network protocol analysis and information fusion. Information and Software Technology, 145, 106838.
[12]. Niswar, M., Safruddin, R. A., Bustamin, A., & Aswad, I. (2024). Performance evaluation of microservices communication with REST, GraphQL, and gRPC. International Journal of Electronics and Telecommunication, 70(2), 429-436.
[13]. Nnamdi, C. (2024, September 12). *GraphQL vs REST – Key differences and use cases*. Refine. https://refine.dev/blog/graphql-vs-rest/
[14]. Quiña-Mera, A., García, J. M., Fernández, P., Vega-Molina, P., & Ruiz-Cortés, A. (2022, September). GraphQL or REST for Mobile Applications?. In International Conference on Advanced Research in Technologies, Information, Innovation and Sustainability (pp. 16-30). Cham: Springer Nature Switzerland.
[15]. Rustemi, A., Atanasovski, V., & Risteski, A. (2023, September). Design of the Blockchain System for the Generation and Verification of Diplomas. In 2023 XXXII International Scientific Conference Electronics (ET) (pp. 1-6). IEEE.
[16]. Stępień, K., & Skublewska-Paszkowska, M. (2025). Performance evaluation of REST and GraphQL API aproaches in data retrieval scenarios using NestJS. Journal of Computer Sciences Institute, 36, 350-356.
[17]. Tsai, O., Li, J., Cheung, T. T., Huang, L., Zhu, H., Xiao, J., ... & Tayebi, M. A. (2025). Graphqler: Enhancing graphql security with context-aware api testing. arXiv preprint arXiv:2504.13358.