*UDC 004.852*

*https://doi.org/10.52058/2786-6025-2023-13(27)-718-733*

**Kosenyuk Hryhoriy Volodymyrovych** PhD in Engineering, Associate Professor, Bohdan Khmelnytsky National University of Cherkasy, Cherkasy, https://orcid.org/0000-0003-2103-3904

**Koseniuk Oleksii Hryhorovych** IT director, Antal trading LTD, Cherkasy, https://orcid.org/0009-0009-2027-6833

# AUTOMATED TESTING OF WEB APPLICATIONS USING GRAPHQL QUERIES: APPROACHES AND TOOLS

**Abstract.** An effective language for manipulating APIs in web services is called Graph Query Language, or GraphQL. It has just lately been made public as a substitute method for resolving RESTful API constraints. This article presents an automated approach to testing GraphQL APIs. We offer an overarching framework for testing automated APIs, including test case development and other tools. In this paper, we perform a comprehensive analysis of the GraphQL field by first outlining the GraphQL concept and its structure, followed by an organized mapping analysis of 84 main studies chosen from a total of 525. Through a generalization of the studies and a particular classification of this research, our work examines trends or knowledge gaps regarding GraphQL. The primary findings of the study indicate that the community is increasingly adopting GraphQL as a powerful substitute for implementing APIs. However, we found that applicable business and government research needed to increase the quantity and quality of empirical evidence collected. Furthermore, we identified the need for targeted research on the majority of GraphQL components, particularly the use of GraphQL API services. GraphQL provides a single endpoint and lets clients describe exactly what data they require, in contrast to REST, which depends on several APIs to obtain data. As a result, less data is sent over the network, performance is enhanced, and client-side development is given greater latitude. Testing with GraphQL is now primarily concerned with making sure the queries and schema are accurate, while the API reacts to client requests as intended. This may be accomplished by using both human and automated testing techniques to ensure the queries produce the desired results and validate the schema. The choice between REST and GraphQL depends on the specific needs of an application and the capabilities of the API. This necessitates test engineers to possess the capability to adapt to any type of API. Fortunately, with the assistance of appropriate tools and approaches, the complexity of underlying technicalities can be alleviated, making GraphQL testing a more accessible and efficient process. This article delves into the evolving landscape of API development and highlights the importance of adept testing in the GraphQL era.

**Keywords:** GraphQL, API, automated test generation, API testing, schema.

**Косенюк Григорій Володимирович** кандидат технічних наук, доцент, Черкаський національний університет імені Богдана Хмельницького, м. Черкаси, https://orcid.org/0000-0003-2103-3904

**Косенюк Олексій Григорович** IT директор, ТОВ «Антал трейдінг», м. Черкаси, https://orcid.org/0009-0009-2027-6833

# АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ ІЗ ВИКОРИСТАННЯМ ЗАПИТІВ GRAPHQL: ПІДХОДИ ТА ІНСТРУМЕНТИ

**Анотація.** Ефективна мова для роботи з API у вебсервісах називається Graph Query Language, або GraphQL. Нещодавно він був оприлюднений як альтернативний метод для вирішення обмежень RESTful API. У цій статті представлено автоматизований підхід до тестування GraphQL API. Ми пропонуємо загальну структуру для тестування автоматизованих API, включно з розробленням тестів та інші інструменти. У цій статті ми виконуємо комплексний аналіз поля GraphQL, спочатку окреслюючи концепцію GraphQL та її структуру, а далі організовуючи картографічний аналіз 84 основних досліджень, вибраних із загальної кількості, 525. Завдяки узагальненню досліджень і конкретній класифікації цього дослідження наша робота вивчає тенденції або прогалини в знаннях щодо GraphQL. Основні результати дослідження свідчать про те, що спільнота все більше приймає GraphQL як потужну заміну впровадження API. Однак, ми з'ясували, що для збільшення кількості та якості зібраних емпіричних доказів необхідні відповідні дослідження бізнесу та уряду. Крім того, ми визначили потребу в цілеспрямованому дослідженні більшості компонентів GraphQL, зокрема використання сервісів GraphQL API. GraphQL передбачає єдину кінцеву точку та дає клієнтам змогу точно описувати, які дані їм потрібні, на відміну від REST, що залежить від кількох API для отримання даних. Як результат, менше даних надсилається через мережу, продуктивність підвищується, а розроблення клієнтської частини отримує більше можливостей. Тестування за допомогою GraphQL тепер здебільшого пов'язане з впевненістю в тому, що запити та схема точні, тоді як API реагує на клієнтські запити за призначенням. Це може здійснюватися за допомогою як людських, так і автоматизованих методів тестування, щоби переконатися, що запити надають бажані результати та підтверджують схему. Вибір між REST і GraphQL залежить від конкретних потреб програми та можливостей API. Це вимагає від інженерів-тестувальників здатності адаптуватися до будь-якого типу API. На щастя, за допомогою відповідних інструментів і підходів можна зменшити складність основних технічних деталей, зробивши тестування GraphQL більш доступним і ефективним процесом. Ця стаття заглиблюється в зміни характеру розроблення API та підкреслює важливість вмілого тестування в епоху GraphQL.

**Ключові слова:** GraphQL, API, автоматизована генерація тестів, тестування API, схема.

**Problem statement.** This article focuses on the operation of GraphQL, its significance, and recommended techniques for GraphQL testing. The lack of research on GraphQL query techniques is a problem for developers and businesses looking to fully utilize this technology. A concentrated effort to record and distribute best practices, optimization techniques, and practical use cases is needed to close this gap. In addition, encouraging cooperation only within the GraphQL community can help to create thorough resources that are up to date with the constantly changing GraphQL ecosystem It guarantees that developers have the direction and expertise required to build effective and high-performing GraphQL APIs. Reducing this disparity will support GraphQL's continuous development and prosperity as a potent and adaptable API technology. When it comes to addressing best practices and tactics for efficiently formulating inquiries, this gap in the literature is very noticeable. Although GraphQL's core concepts, schema design, and usage are widely available, few comprehensive resources offer deep insights into query speed optimization, data fetching efficiency enhancement, and frequent query issues in GraphQL.

There is a literature gap that emphasizes the approaches of queries. It reduces information over- and under-fetching by enabling customers to request just the required data. However, even with its broad use and copious documentation, there remains a significant literature vacuum about the query strategies used in the GraphQL ecosystem. When it comes to sharing the best practices, tactics, and resources for efficiently designing queries, this disparity is very noticeable.

There aren't many real-world instances of best practices and query optimization in action. GraphQL query optimization tools that can evaluate, display, and provide recommendations for developers and organizations would be very useful, helping close the knowledge gap by illustrating the practical applications of query optimization techniques.

Because of the size and diversity of the GraphQL community, it may be possible to build query optimization and efficiency analysis tools through cooperation. Community-driven tools can encourage the dissemination of best practices and offer approachable solutions to common problems. Developers can contribute to accessible projects aimed at improving GraphQL query performance.

GraphQL Ecosystem Evolution and Tool Adaptation: GraphQL is constantly changing, with new extensions, libraries, and tools appearing regularly. These advancements affect the way queries are designed and optimized. Tools need to change and grow with the GraphQL ecosystem to give developers current and useful solutions.

Literature Gap in Methods for GraphQL Testing: There is a noticeable vacuum in the literature about the approaches and techniques for automated tools of GraphQL-based web apps, even despite the growing popularity of GraphQL in the development of web applications. Current materials mostly concentrate on fundamental ideas and GraphQL schema creation, and there is no extensive advice on testing methodologies.

Need for Specialized Testing Tools: The requirement for specialist testing tools and frameworks is necessary to handle the complexities of GraphQL-based web apps. There may be a gap in the test automation ecosystem as GraphQL queries, mutation, and subscriptions may not be completely supported by the testing tools available today, which are mostly made for RESTful APIs.

Test Automation for Reliability and Efficiency: Continuous integration and delivery (CI/CD) is made possible by automated testing, a fundamental component of contemporary software development. Unfortunately, the effectiveness and dependability of development pipelines are hampered by the absence of strong automated testing methodologies and tools for GraphQL-based web applications, which may result in quality regressions and deployment delays.

**Analysis of recent research and publications.** A few research provide methods for automatically generating tests for GraphQL.

Deviation testing is a test-generating method for GraphQL developed by Vargas et al.  mentioned in [1]. Deviation testing, a type of test amplification [2], starts with a single test and generates what are known as deviations—variations of the test. Fields Deviation, Not Null Deviation, Type Deviation, and Empty Fields Deviation are the four methods used to produce these deviations. Using the GraphQL schema as a guide, the Fields Deviation technique adds or eliminates features from the original query. The Not Null Deviation method allows an error to be raised and changes a not null variable with null. By substituting an input that is different from the intended type, the Type Deviation technique anticipates an error. With the Empty Fields Deviation technique, a query's subfields and fields are all deleted, and a syntax mistake is anticipated. Three research individuals were used to test their method, and the findings indicated that two of them had trouble with the Empty Fields Deviation. This method's ease of use and low effort requirements for developers make it ideal for expanding the current test suite with the addition of new test cases. FrontApp is unable to employ this method, though, as it lacks test cases for its GraphQL API and still needs pre-existing manually created test cases.

Based on GraphQL schemas, Karlsson et al. developed a test-generating tool for GraphQL [3]. Based on the fields and objects in the schema, the produced tests' queries are generated at random. Additionally, the tool features an argument generator, producing arguments either entirely at random or based on predetermined categories. For instance, the tool will generate a random integer rather than a random text if a parameter is declared to be an integer. The HTTP status message and the kinds of returned fields are the two main properties that are tested by the suggested method. They run the query and first assess the status code that is returned. The kinds of the provided fields are then confirmed by comparing the answers obtained from the query's execution with the schema. The publication, however, is not clear about how specific their tests are about the attributes that comprise the returning objects that are confirmed by assertions. By creating scenarios for tests based on real queries and the responses recorded from production, our study seeks to solve the shortcomings of the previous research by eliminating the necessity for test creation to be done randomly.

Eventually, utilizing preexisting unit test cases, Abdi et al. [4] create new integration test cases. They employ the knowledge from unit test cases, such as how to create classes, build parameters for method calls, and what the anticipated outcome is to create more complicated test cases. Rather than concentrating on calling individual methods, these test cases highlight class relationships. This investigation shows promising results in identifying software bugs. The earlier studies offer new methods or standards for creating test cases that increase coverage or identify errors.

Web applications may have complicated query patterns, including dynamic data fetching and highly nested inquiries, thanks to GraphQL's flexibility. The specific issues these complexities present for automated testing include verifying query correctness, data integrity, and overall system stability.

**The purpose of the article:** The goal of this work is twofold: first, it presents the unique GraphQL paradigm by conceptualizing, illustrating, and exemplifying its constituent parts. Next, to organize the field of interest of the scientific community and provide an overview of the topic using a publication categorization scheme, a systematic mapping study (SMS) of GraphQL should be carried out. Researchers and practitioners are becoming more interested in GraphQL, a revolutionary way to develop APIs. To provide a broad overview of this field, we begin this work by introducing a conceptual framework that describes the referred to GraphQL framework from its formal specification. We then go on to illustrate and exemplify its various components, which will serve as the foundation for developing an in-depth knowledge of the GraphQL paradigm and connecting the different elements that have been studied. As the primary focus of our study, we then carried out a systematic literature review (SMS) to present an overview of the field's research and identify gaps and trends in GraphQL. Our study was specifically concerned with identifying the who, where, when, what, and why of GraphQL research, in addition to placing the various research topics in the context of the GraphQL paradigm that we presented. Our study's findings support the notion that, despite GraphQL's increasing popularity as a substitute for API development, there is not a strong scientific community supporting it. While publishing in high-profile publications is becoming more common, there are still gaps in the literature from recent studies, particularly when it comes to the development of empirical evidence, validation in practical use cases, and assessment of GraphQL's underutilized features and extra-quality attributes.

**Presenting main material.** In the context of API-first development, GraphQL adoption and GraphQL testing implementation have become essential. The necessity of functional, scalable, and maintainable code has increased dramatically due to the increasing need for contemporary software. For SOAP and REST architectural styles, human and automated API testing is required to facilitate autonomous growth and isolate the front end from the backend [5].

GraphQL is being used by several digital businesses, including Facebook, Amazon, AirBnB, GitHub, and others, in place of more conventional REST and SOAP APIs.

GraphQL APIs are being used by major players in the industry, such as Amazon (AWS), Twitter, Facebook, GitHub, and others, for both internal and external operations.

GraphQL, also referred to as "SQL for APIs," is a runtime that effectively pulls data from databases rather than just a query language. Whereas RESTful APIs sometimes need to combine information from several server answers, GraphQL enables apps to retrieve all relevant data with only one request.

Despite a noticeable lack of resources for GraphQL testing, this transition is impressive. Although testing RESTful API endpoints is generally simple, there are special issues with GraphQL because many tools now in use do not completely support it [6].

Ensuring that GraphQL architectures are comprehensively, thoroughly, and autonomously tested is essential to maintaining business processes and performant APIs as the technology continues to gain adoption.

**What is GraphQL?** GraphQL functions as an API query language, enabling the retrieval of data from databases to streamline the query execution process. GraphQL uses types and fields rather than separate endpoints with predetermined replies to get all necessary data in a single request, in contrast to REST APIs, that rely on numerous resource URLs.

Resolvers, or GraphQL services, can be built in any programming language, and language independence is guaranteed by the GraphQL schema language. Object types are defined by schemas, and each one represents a fetch-able object with related data. This provides developers with a recognizable framework [7].

**Benefits of using GraphQL**

Instead of creating many REST calls, the user may utilize GraphQL to make a single call to retrieve the desired data. Here are a few benefits of utilizing GraphQL [8].

- Describe a data type: It is simple to predict the format of the data returned from an inquiry since GraphQL queries replicate their answers. Writing a question following your application's needs is aided.
- Hierarchical by design: GraphQL naturally follows the exchange between objects, while a RESTful service could need a SQL fancy join statement or several round trips. Ultimately, this hierarchy works well with hierarchical user interfaces and graph-structured information repositories.
- Firmly Typed: Each GraphQL query level correlates to a certain type, which defines a collection of fields that may be obtained. This allows GraphQL to provide informative error warnings before running code, much like SQL does.
- Considerate: An IDE such as GraphiQL or Relay, or statically typed languages, can generate code on a secure platform made available by a GraphQL server. Developers may quickly learn and inspect an API using GraphiQL without having to dig through the codebase.
- No iterations: Because the client's query determines the form of the return data only, servers become clear-cut and easy to understand. Typically, more fields are introduced to the server while adding new features to the product. The related server fields are obsolete while unsetting earlier functionality. With this slow, backward-compatible method, an increasing version number is no longer required.

**REST vs. GraphQL**

Two well-liked methods for creating APIs are GraphQL and REST (Representational State Transfer), each having advantages of its own.

An established standard called REST exposes data via a set of specified endpoints. It works well for a wide range of applications because of its reputation for ease of use and simplicity of caching.

However, GraphQL provides greater freedom. Clients may use GraphQL to precisely request the data they want, which minimizes data over- and under-fetching. It's perfect in scenarios where maximizing productivity and reducing network queries are essential [9].

Which REST or GraphQL is better for your project will depend on its particular needs. For more straightforward apps, REST is a good option, but GraphQL excels when you want exact control for your API interactions. Comprehending the distinct advantages of each might assist you in selecting the best option for your API design [8].

**Schema and Queries**

The first step in implementing a GraphQL API is to define a schema using the GraphQL schema language. This schema is a multi-graph, with objects serving as nodes that specify types and hold lists of fields. As soon as a type specifies one of its variables as another object, the multigraph edges become visible [10].

**Querying GraphQL API**

The following are the steps used how to interact with a GraphQL API via HTTP before we can begin testing the various GraphQL components. There are several methods available for initiating an API. I've included a few simple, often-used methods below [5]:

- cURL: For processing data across the HTTP (and many more) protocols, Curl is a well-liked command-line utility and library. Sending a curl command with three arguments is required.

1) To start, since the question is a JSON string, the content-type is application/json.

2) The second thing that will be supplied is an actual query, such as {"query":" {movies {name}}"}.

3) Lastly, here is the GraphQL 4) endpoint: The URL is n7b67.sse. codesandbox.io/graphql. Furthermore, the majority of GraphQL requests would be made using the POST HTTP verb.

**Components to Test in GraphQL**

Here are several test kinds that might be utilized [10]:

- Query tests: Verify that a given query and its associated variables provide the intended result.
- Ensure that a particular query and its associated variables correctly keep information inside the database by running mutation tests.
- Load tests: Ensure the API continues functioning (per SLAs) even in the face of a high volume of queries.
- Security tests: Verify that sensitive data is not returned via APIs without taking the required security measures.

You should mimic replies when employing GraphQL for testing a third-party web service (such as GitHub V4). It can shorten test run durations and help you prevent unused consumption. To mimic these services, you may occasionally use fixtures and mocks. But in other situations, virtualizing services could be necessary to examine consumption and additional metrics.

**GraphQL testing tools**

A GraphQL server may be tested using a variety of instruments. For instance, GraphQL servers may be tested using some of the same frameworks used for Node.js servers. One well-liked JavaScript testing framework for testing GraphQL servers in Node.js is called Mocha. With Mocha, you can create asynchronous tests, sequentially execute tests, provide reports, and map the exceptions to the test cases https://www.apollographql.com https://mochajs.org/#getting-started. There are more frameworks available, such as Sails.js, Chai, and SuperTest. The schema itself may be evaluated in addition to the query testing process. With Apollo's Mocking8, it is possible to create tests using actual queries that concentrate on the schema's type definition. By employing mocks, these tests help prevent any type of conflict. Simulating inquiries and seeing the answer is a crucial activity. It is one benefit of utilizing Graphene9's Test Client. It enables testing to ensure that a Django template renders a query request with certain values. The tools listed above make it easier to write tests for GraphQL servers [11].

**Test generation techniques**

Model-based testing (MBT) is one method of automated test creation where a test oracle is created using a model of the system being tested to determine if the test passes or fails. Various models can be employed to depict distinct components of the system. In engineering and architecture, physical models, like models of structures or bridges, have always been used for testing. Software system models can characterize several facets of the system that is being tested [9]. A requirements data model, for example, specifies the range of values that can be assigned to a parameter. The set of acceptable and unacceptable variables that will be produced for that variable in a test is defined by a test-generating method based on this model. Another typical illustration of a model is the state machine, which describes a system's behavior in terms of legitimate states that the system can reach depending on actions taken on it. As a result, automated test creation using models like this is feasible, where the test input is the entrance action to the state that represents the anticipated behavior and the subsequent state is the behavior that is expected. The GraphQL schema may be thought of as a system model for a GraphQL API [12].

**New techniques or criteria for automated testing**

Using a method that builds functional abstracts from test suites, authors [13] accumulate cases till the operational abstract becomes static. They choose the test cases using the operational difference approach after they have been prepared to enhance fault detection.

Later, Greca et al [14] examined the variables influencing test suite augmentation in his research and found that the approach was one of them. Xu describes his approach for guided test suite augmentation and conducts concolic testing as a tool for test case generation. Based on the code coverage, the findings indicate that his approach performed more effectively and efficiently than the concolic technique.

To separate the test code from the test input and to extend pre and post-conditions, Lam et al. [15] propose test generation and mutation. It provides the opportunity to use symbolic parameters to abstract a sizable portion of the code input. Additionally, they use variety to determine the relevant behavior; the more mistakes a post-condition stores, the more relevant it is.

A testing strategy based on black-box properties was presented by Karlsson et al. [11]. The following stages make up the procedure. Initially, every type specification together with its relationship is taken out of the schema. Using user-provided "data generators" that are tailored to their needs, data is created at random by the schema. Furthermore, the authors provide two approaches that may be used as automated oracles: the initial one checks the HTTP status codes that are returned, and the second one confirms that the returned data is consistent with the specified schema.

**Test GraphQL API Implementations**

Front-end systems and back-end APIs are separated by an abstraction layer called GraphQL. Because of this, GraphQL is necessary for testing. Multiple backend resources may be accessed and combined into a single, meaningful result using GraphQL queries.

Because they facilitate the creation of new building pieces that may be used for various applications, backend APIs are frequently granular. This does not imply, however, that the intended front-end operations are carried out. GraphQL makes working with backend data easier. A connection with schemas that define system behavior is used to do this. Then, you may use APIs to obtain effective data.

Every GraphQL schema translates to functions, which call your backend in turn. Calls are conducted to databases, REST APIs, and other resources necessary for gathering the needed data by business logic.

The functions then put all the pieces together to create a response that keeps the structure of the request. This facilitates the process of determining which data is related to each request element.

In addition, GraphQL may be configured to call different backend services as it puts together a query answer. This can shorten the time a user needs to spend reading and comprehending the information returned by a request by reducing the amount of time they spend browsing through API documentation [16].

**Using GraphQL Playground**

A GraphQL client is called GraphQL Playground [17]. It may be used to test various queries, create distinct playgrounds within GraphQL IDEs, and organize them thematically or by giving them names. Similar to GraphiQL, Playground may generate documentation automatically without requiring you to submit and handle introspection queries and responses by hand. Another fantastic benefit is that it doesn't require the availability of the GraphiQL interface. The tool may be used locally using a data file, or it can be directed to the GraphQL nodes through a URL. You may use GraphQL Playground to immediately test for vulnerabilities instead of sending HTTP queries through a personal proxy. This implies that you may evaluate and interact with GraphQL in a basic way using this tool. Use an individual proxy for additional, more sophisticated payloads.

Keep in mind that you might occasionally require altering the headers of the HTTP request at the bottom to include an authentication method, such as a session ID. To confirm whether authorization problems actually exist, this nevertheless permits the creation of numerous "IDEs" with various rights.

**EasyGraphQL**

Most functional tests in GraphQL are optimized to make sure that the requests, modifications, and schema perform as intended on the client side. For this kind of testing, a plethora of security testing technologies are available. You may select the ones that work best for your language, platform, test infrastructure, and specific testing needs.

For example, the most popular tool for functional GraphQL testing for creating JavaScript APIs is EasyGraphQL. As a component of your automation test toolset, you may link it with a library like Mocha and then test assertions to assess API answers [18].

**Apollo GraphQL** is a full-stack platform designed for GraphQL API development [17]. It offers frameworks and tools that make creating, maintaining, and using GraphQL APIs easier.

The Apollo Server, an efficient and adaptable server that makes it simple to create scalable and effective GraphQL APIs, is the central component of the Apollo GraphQL platform. The Apollo Server facilitates seamless integration with pre-existing systems by supporting a broad spectrum of data sources, such as databases, REST APIs, and other services.

Apollo also offers several client libraries that make using GraphQL APIs easier, such as the Apollo Client for web and mobile. The Apollo Client offers sophisticated functionalities including caching, an optimistic user interface, real-time updates, and an easy-to-use query and mutation tool for data.

Apollo offers several more tools and services besides the Apollo Server and Apollo Client, such as a schema maintenance system, a GraphQL analytics service, and a collection of developer tools for creating and debugging GraphQL APIs.

**The Katalon Platform**

QA teams may plan, develop, execute, and diagnose automation API and UI integration test cases with the aid of Katalon [19], a test automation platform. Furthermore, users may create queries and changes in Java/Groovy using Script Mode. Strong API testing capabilities are offered by Katalon, which allows you to test REST, SOAP, and GraphQL APIs. You can test your new API schema on the same platform as traditional API assets and import current API specifications thanks to its connections with GraphQL tools like Postman, Swagger, and SoapUI.

- Using RESTful methods, GET and POST to create test requests and execute;
- Complete visualization of the headers, content, and status code;
- Parameterization of query variables; and execution against the JSON schema.

**UI integration testing**

Web components like buttons, dropdown menus, and input fields are arranged inside an object repository. This is essential for updating properties and locators globally throughout test cases.

Additionally, there is a test recorder and keyword library for simple drag and drop. Imagine having the ability to quickly create an automation script from your manual testing.

**Data-driven testing**

Because Katalon supports several databases (MySQL, Oracle, and SQL Server) and data formats (CSV, Excel), it makes it possible to automate API testing with a variety of data situations. In order to link variables to the appropriate data elements from external data sources, testers might create parameters in test scripts, objects, or request attributes.

**Create, write, execute, and evaluate**

All the capabilities required for end-to-end continuous automated testing are integrated into Katalon: Writing tests in either full-code or low-code formats, running them in CLI mode between environments, and getting visibility through thorough reporting are all examples of this. There's no requirement for installs or workarounds for integration.

**Auto-triggered on CI**

Katalon's tool connections with Circle CI, Jenkins, and GitLab facilitate the automation and execution of test cases within the CI/CD pipeline. Setting up UI and API tests to run automatically guarantees that integration problems are found and fixed quickly.

**Karate**

Karate is a framework for GraphQL testing that simplifies the process of testing GraphQL APIs. It makes it simple to add variables to your searches and supports JSON. You may use Karate's match assertions to check the answer and pass the GraphQL query exactly as is. Among this tool's primary characteristics are [20]:

- Simple Execution: Java Runner files and maven commands may be used to run test suites.
- Adaptability: 1. Reusable feature files and scripts that can be called by other feature files and scripts. 2. An integrated JavaScript engine that enables the creation of reusable JavaScript functions. Payload information and user-defined routines can be reused throughout many tests.
- Built-In Assertions: Return response code, reaction response time, headers, and other properties may all be validated using built-in assertions.
- Parallel Execution: Using Karate DSL, built-in multi-threaded execution in parallel is supported.
- Integration: JUnit, TestNG, and other unit testing frameworks are simple to integrate with.
- Performance Testing: Capability to use Gatling to test an application's performance.
- Custom Code: Java/JS user-defined functions may be written in Karate.

- Markers: Use markers such as #ignore, #null, #notnull, and #boolean to demonstrate assertion capabilities.

**Testing schema**

For schema testing, we may make use of test queries, fake schemas, and static type checks. Schema testing may be done with the libraries graphql-schema-linter and eslint-plugin-graphql6.

A tool to check the schema specification is graphql-schema-linter, and a linter to expand ESLint rules is Est-lint-plugin-graphql. Schema testing will benefit from mocking the schema using a mock-server using graphql-tools (a collection of npm packages), in addition to static type checks, and paying attention to potential combinations [21].

**Testing mutations and queries**

Easygraphql-tester allows us to mimic queries and modifications, even though we may utilize an automated technique for query testing by utilizing the libraries "request" and "supertest.". Alternatively, we may use easygraphql-tester to explicitly construct test assertions rather than using mocking [14].

**Testing Resolvers**

Since resolvers are pure functions, testing them is simpler. Resolvers are merely javascript functions or functions written in any programming language the GraphQL server depends on, therefore we don't require packages to test them. Schemas provide the information that resolvers gather, thus it's important to test them early to prevent expensive mistakes [21].

**Conclusions.** In 2023, GraphQL is becoming more and more popular as well-known digital companies—not only Facebook—use it for internal as well as external APIs. GraphQL testing doesn't have to be difficult; all it takes is the appropriate equipment and procedures. Developers have been utilizing the classic REST architectural API for many years, but GraphQL is rapidly taking the lead as an alternative. It allows front-end developers to use a single API to query only the required data. Back-end developers are using industry-standard techniques to ensure they create scalable and fluid APIs in GraphQL due to its many advantages.

The needs of the app and the capabilities of the API are two factors that will determine which protocol is best suited for a particular application. The ability to deal with any kind of API is a prerequisite for a test engineer. By using an aid, the overall burden of underlying technicalities can be reduced.

***References:***

1. EMB: A Curated Corpus of Web/Enterprise Applications And Library Support for Software Testing Research / A. Arcuri et al. *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Dublin, Ireland, 16–20 April 2023. 2023. URL: https://doi.org/10.1109/icst57152.2023.00047 (date of access: 04.11.2023).

2. Cheng S., Hartig O. LinGBM: A Performance Benchmark for Approaches to Build GraphQL Servers. *Web Information Systems Engineering – WISE 2022*. Cham, 2022. P. 209–224. URL: https://doi.org/10.1007/978-3-031-20891-1_16 (date of access: 04.11.2023).

3. Karlsson S., Causevic A., Sundmark D. Automatic Property-based Testing of GraphQL APIs. *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, Madrid, Spain, 20–21 May 2021. 2021. URL: https://doi.org/10.1109/ast52587.2021.00009 (date of access: 04.11.2023).

4. Adding dynamically-typed language support to a statically-typed language compiler / K. Ishizaki et al. *ACM SIGPLAN Notices*. 2012. Vol. 47, no. 7. P. 169–180. URL: https://doi.org/10.1145/2365864.2151047 (date of access: 04.11.2023).

5. Belhadi A., Zhang M., Arcuri A. Evolutionary-based automated testing for GraphQL APIs. *GECCO '22: Genetic and Evolutionary Computation Conference*, Boston Massachusetts. New York, NY, USA, 2022. URL: https://doi.org/10.1145/3520304.3528952 (date of access: 04.11.2023).

6. Belhadi A., Zhang M., Arcuri A. Random Testing and Evolutionary Testing for Fuzzing GraphQL APIs. *ACM Transactions on the Web*. 2023. URL: https://doi.org/10.1145/3609427 (date of access: 04.11.2023).

7. Zetterlund L., Tiwari D., M. Monperrus, Baudry B. Harvesting production graphql queries to detect schema faults, *IEEE Conference on Software Testing, Verification and Validation (ICST).* 2022. P. 365-376. URL: https://arxiv.org/pdf/2112.08267.pdf (date of access: 04.11.2023).

8. Sridharamurthy S. Types of API protocols- REST, SOAP, graphQL, gRPC. *ACCELQ Inc*. 2023. URL: https://www.accelq.com/blog/types-of-api-and-its-usage/ (date of access: 04.11.2023).

9. Brito G., Valente M. T. REST vs GraphQL: A Controlled Experiment. *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil, 16–20 March 2020. 2020. URL: https://doi.org/10.1109/icsa47634.2020.00016 (date of access: 04.11.2023).

10. Sharma S. Modern API Development with Spring and Spring Boot: Design Highly Scalable and Maintainable APIs with REST, GRPC, GraphQL, and the Reactive Paradigm. Packt Publishing, Limited, 2021. 582 p.

11. Krishna V. V., Gopinath G. Agile test automation for web application using testing framework with random integration algorithm in machine learning to predict accurancy and response time on automated test results. *Journal of Theoretical and Applied Information Technology*. 2022. Vol.100, no.16. URL: https://www.jatit.org/volumes/Vol100No16/2Vol100No16.pdf (date of access: 04.11.2023).

12. Can GraphQL Replace REST? A Study of Their Efficiency and Viability / S. L. Vadlamani et al. *2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, Madrid, Spain, 4 June 2021. 2021. URL: https://doi.org/10.1109/ser-ip52554.2021.00009 (date of access: 04.11.2023).

13. Lin J.-W., Salehnamadi N., Malek S. R oute : Roads Not Taken in UI Testing. *ACM Transactions on Software Engineering and Methodology*. 2022. URL: https://doi.org/10.1145/3571851 (date of access: 04.11.2023).

14. Greca R., Miranda B., Bertolino A. State of Practical Applicability of Regression Testing Research: A Live Systematic Literature Review. *ACM Computing Surveys*. 2023. URL: https://doi.org/10.1145/3579851 (date of access: 04.11.2023).

15. Dependent-test-aware regression testing techniques / W. Lam et al. *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Virtual Event USA. New York, NY, USA, 2020. P. 298-311.URL: https://doi.org/10.1145/3395363.3397364 (date of access: 04.11.2023).

16. Lawi A., Panggabean B. L. E., Yoshida T. Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System. *Computers*. 2021. Vol. 10, no. 11. P. 138. URL: https://doi.org/10.3390/computers10110138 (date of access: 04.11.2023).

17. GraphQL Playground. *Apollo Docs*. URL: https://www.apollographql.com/docs/apollo-server/v2/testing/graphql-playground/ (date of access: 04.11.2023).

18. Vashishth R. *AI Powered Quality Engineering Services - Software Testing Company*. 2020. URL: https://www.bugraptors.com/blog/testing-graphql-to-leverage-api (date of access: 04.11.2023).

19. Katalon. What is GraphQL Testing? How To Test GraphQL APIs?. *katalon.com*. URL: https://katalon.com/resources-center/blog/graphql-testing (date of access: 04.11.2023).

20. Gentile V. GraphQL Testing With Karate. 2020. *Codemotion Magazine*. URL: https://www.codemotion.com/magazine/frontend/web-developer/graphql-testing-karate/ (date of access: 04.11.2023)

21. GraphQL Testing in 2023: Benefits, Types & Components. *AIMultiple*. URL: https://research.aimultiple.com/graphql-testing/ (date of access: 04.11.2023).

***Література:***

1. EMB: A Curated Corpus of Web/Enterprise Applications And Library Support for Software Testing Research / A. Arcuri et al. *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Dublin, Ireland, 16–20 April 2023. 2023. URL: https://doi.org/10.1109/icst57152.2023.00047 (date of access: 04.11.2023).

2. Cheng S., Hartig O. LinGBM: A Performance Benchmark for Approaches to Build GraphQL Servers. *Web Information Systems Engineering – WISE 2022*. Cham, 2022. P. 209–224. URL: https://doi.org/10.1007/978-3-031-20891-1_16 (date of access: 04.11.2023).

3. Karlsson S., Causevic A., Sundmark D. Automatic Property-based Testing of GraphQL APIs. *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, Madrid, Spain, 20–21 May 2021. 2021. URL: https://doi.org/10.1109/ast52587.2021.00009 (date of access: 04.11.2023).

4. Adding dynamically-typed language support to a statically-typed language compiler / K. Ishizaki et al. *ACM SIGPLAN Notices*. 2012. Vol. 47, no. 7. P. 169–180. URL: https://doi.org/10.1145/2365864.2151047 (date of access: 04.11.2023).

5. Belhadi A., Zhang M., Arcuri A. Evolutionary-based automated testing for GraphQL APIs. *GECCO '22: Genetic and Evolutionary Computation Conference*, Boston Massachusetts. New York, NY, USA, 2022. URL: https://doi.org/10.1145/3520304.3528952 (date of access: 04.11.2023).

6. Belhadi A., Zhang M., Arcuri A. Random Testing and Evolutionary Testing for Fuzzing GraphQL APIs. *ACM Transactions on the Web*. 2023. URL: https://doi.org/10.1145/3609427 (date of access: 04.11.2023).

7. Zetterlund L., Tiwari D., M. Monperrus, Baudry B. Harvesting production graphql queries to detect schema faults, *IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2022. P. 365-376. URL: https://arxiv.org/pdf/2112.08267.pdf (date of access: 04.11.2023).

8. Sridharamurthy S. Types of API protocols- REST, SOAP, graphQL, gRPC. *ACCELQ Inc*. 2023. URL: https://www.accelq.com/blog/types-of-api-and-its-usage/ (date of access: 04.11.2023).

9. Brito G., Valente M. T. REST vs GraphQL: A Controlled Experiment. *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil, 16–20 March 2020. 2020. URL: https://doi.org/10.1109/icsa47634.2020.00016 (date of access: 04.11.2023).

10. Sharma S. Modern API Development with Spring and Spring Boot: Design Highly Scalable and Maintainable APIs with REST, GRPC, GraphQL, and the Reactive Paradigm. Packt Publishing, Limited, 2021. 582 p.

11. Krishna V. V., Gopinath G. Agile test automation for web application using testing framework with random integration algorithm in machine learning to predict accurancy and response time on automated test results. *Journal of Theoretical and Applied Information Technology*. 2022. Vol.100, no.16. URL: https://www.jatit.org/volumes/Vol100No16/2Vol100No16.pdf (date of access: 04.11.2023).

12. Can GraphQL Replace REST? A Study of Their Efficiency and Viability / S. L. Vadlamani et al. *2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, Madrid, Spain, 4 June 2021. 2021. URL: https://doi.org/10.1109/ser-ip52554.2021.00009 (date of access: 04.11.2023).

13. Lin J.-W., Salehnamadi N., Malek S. R oute : Roads Not Taken in UI Testing. *ACM Transactions on Software Engineering and Methodology*. 2022. URL: https://doi.org/10.1145/3571851 (date of access: 04.11.2023).

14. Greca R., Miranda B., Bertolino A. State of Practical Applicability of Regression Testing Research: A Live Systematic Literature Review. *ACM Computing Surveys*. 2023. URL: https://doi.org/10.1145/3579851 (date of access: 04.11.2023).

15. Dependent-test-aware regression testing techniques / W. Lam et al. *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Virtual Event USA. New York, NY, USA, 2020. P. 298-311.URL: https://doi.org/10.1145/3395363.3397364 (date of access: 04.11.2023).

16. Lawi A., Panggabean B. L. E., Yoshida T. Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System. *Computers*. 2021. Vol. 10, no. 11. P. 138. URL: https://doi.org/10.3390/computers10110138 (date of access: 04.11.2023).

17. GraphQL Playground. *Apollo Docs*. URL: https://www.apollographql.com/docs/apollo-server/v2/testing/graphql-playground/ (date of access: 04.11.2023).

18. Vashishth R. *AI Powered Quality Engineering Services - Software Testing Company*. 2020. URL: https://www.bugraptors.com/blog/testing-graphql-to-leverage-api (date of access: 04.11.2023).

19. Katalon. What is GraphQL Testing? How To Test GraphQL APIs?. *katalon.com*. URL: https://katalon.com/resources-center/blog/graphql-testing (date of access: 04.11.2023).

20. Gentile V. GraphQL Testing With Karate. 2020. *Codemotion Magazine*. URL: https://www.codemotion.com/magazine/frontend/web-developer/graphql-testing-karate/ (date of access: 04.11.2023)

21. GraphQL Testing in 2023: Benefits, Types & Components. *AIMultiple*. URL: https://research.aimultiple.com/graphql-testing/ (date of access: 04.11.2023).