**Лабораторна робота 5**

**Знайомство з архітектурним стилем REST та мовою запитів GraphQL**

**Мета роботи:** ознайомитися з архітектурним стилем REST і технологією GraphQL. Отримати практичні навички створення GraphQL-запитів.

1. **Теорія**

**1.1 Архітектурний стиль REST**

Широко відома клієнт-серверна архітектура в даний час не прийнятна для побудови систем комерційної спрямованості на її основі, зважаючи на значну кількість потенційних користувачів таких систем (інтернет-магазини, портали, пошукові системи і т.д.). Основна причина – незадовільні показники продуктивності систем на основі цієї архітектури через надмірності обчислювального навантаження і потоків трафіку. Рішення полягає в використанні сучасного архітектурного стилю REST (Representational State Transfer). Цей термін був введений в 2000-му році Роєм Філдінгом одним з творців протоколу HTTP. Системи, побудовані на основі цього архітектурного стилю прийнято називати RESTful-системами. Архітектурний стиль REST призначений для проектування розподілених систем. Під "розподіленістю" розуміється, перш за все, територіальна розподіленість компонентів системи, а також супутні цьому витрати (тимчасові, обчислювальні). Разом з тим, архітектурний стиль REST також оперує поняттями "клієнта" і "сервера". Відмінні риси REST:

– відсутність станів (statelessness) - ні клієнт не запитує стан сервера, ні сервер не цікавиться станом клієнта. На практиці це означає, що розуміння клієнтом і сервером повідомлень один від одного ніяк не залежить від кількості і складу раніше прийнятих ними повідомлень. Таким чином, в контексті розподілених систем, стан компонента системи визначається кількістю і складом прийнятих ним повідомлень: вхідне повідомлення спонукає виклики відповідних обробників повідомлень, що змінюють стан. Ця властивість досягається завдяки оперуванню поняттям "ресурс". Під "ресурсом" розуміється поняття предметної області Web – деякий об'єкт, який підлягає зберіганню або пересиланню деякого Web-сервісу. Взаємодія RESTful-систем здійснюється за допомогою операцій над ресурсами. З цієї причини немає потреби в прив'язці до реалізації деякого інтерфейсу. У цьому ключі варто відзначити, що взаємодію компонентів розподіленої системи можна реалізувати двома принципово різними шляхами: шляхом обміну повідомленнями і за допомогою використання поділюваних змінних. У цьому ключі поняття «ресурс» співвідноситься з поділюваною змінною;

– відділення реалізацій клієнта і сервера – реалізація сервера може бути виконана незалежно від клієнтської частини, і навпаки. Це означає, що зміна коду клієнтської частини не впливає на функціонування сервера, і навпаки.

Перераховані вище особливості дозволяють характеризувати RESTful-додатки наступним чином: надійні, продуктивні, масштабовані. Разом з тим, REST-технологія характеризується рядом недоліків, критичних для мобільних додатків. Деякі з цих недоліків наведені нижче:

– при запиті складних об'єктів клієнтським додатком у серверного додатка виникає істотний (найчастіше надлишковий) трафік;

– при зміні вимог до додатка (системи) REST endpoints доповнюються ("обростають") все новими даними. Для клієнтських застосунків, створених раніше, ці дані є надлишковими, породжують надлишковий трафік і зростання часу відгуку при взаємодії клієнтської і серверної частин застосунку;

– REST endpoints слабо типізовані і характеризуються недоліком метаданих. Це ускладнює процедуру розвитку і супроводу відповідних програмних продуктів.

Докладніше про парадігму REST та принципи формування запитів наведено [за посиланням](https://www.codecademy.com/article/what-is-rest). Наведені недоліки технології REST подолані в рамках прогресивної технології GraphQL, мова про яку піде далі.

**1.2 Мова запитів GraphQL**

GraphQL – це мова запитів, створена компанією Facebook і призначена для отримання даних програми уніфікованим чином (шляхом). Цей засіб було представлено світовій громадськості в 2012 році в рамках технології Relay. GraphQL – це мова запитів (query language) для заданих API, а також серверне середовище виконання для здійснення запитів шляхом використання системи типів, визначеної для даних користувача.

Засоби GraphQL представляються у вигляді бібліотеки: вона не асоціюється з деякою базою даних або системою зберігання, а грунтується лише на наявних в розпорядженні коді і даних.

Базові принципи GraphQL:

– ієрархічність – GraphQL-запит – це ієрархічна множина полів; структура запиту відповідає структурі даних, яку ми запитуємо;

– продукто-центровані (мається на увазі цільовий програмний продукт) – характеристики GraphQL визначені вимогами front-end інженерів з точки зору view-складової схеми MVC (Model-View Controller);

– обернено сумісний;

– структурований – гранулярність на рівні полів;

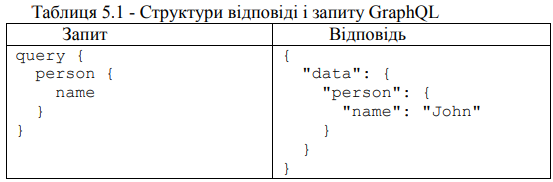
– є протоколом прикладного рівня;

– строго типізований;

– інтроспективний – клієнтські та інші програмні компоненти можуть запитувати систему типів безпосередньо на основі синтаксису GraphQL.

Сервіс GraphQL створюється шляхом визначення типів, полів в рамках цих типів, а також функцій для кожного поля заданого типу. Іншими словами, GraphQL – це засіб запитування окремо взятих полів об'єктів.

Характерна особливість GraphQL - структури запиту і JSONвідповіді (JavaScript Object Notation) збігаються (табл. 5.1).

****

У табл. 5.1 для поля "name" вертається рядок "John".

**1.3 Порівняння REST і GraphQL**

Відмінні риси технологій REST і GraphQL наведені у вигляді порівняльної таблиці (табл. 5.2).

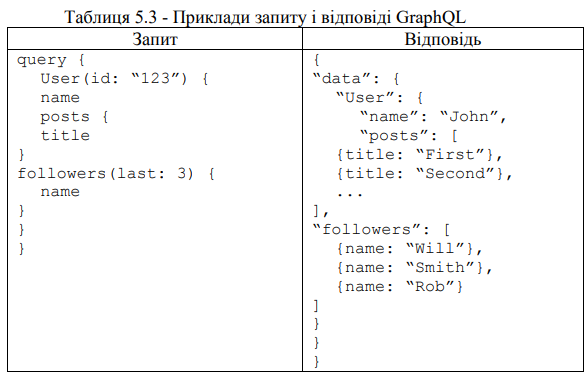
Таблиця 5.2 - Порівняння технологій REST і GraphQL

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Параметри порівняння** | **Основоположні принципи** | |
| *REST* | *GraphQ* |
| 1. | Відмінні риси | Відсутність станів, структурований доступ до ресурсів. | Перехід від концепції множини точок доступу до ресурсів (API Endpoint) до концепції єдиної точки доступу. |
| 2. | Переваги | Підвищення продуктивності і спрощення архітектури розподілених систем, в порівнянні з технологією RPC (Remote Procedure Call). | Забезпечує гнучкість і ефективність клієнт-серверної взаємодії, зокрема, за рахунок зниження надмірності інформації, що передається; немає потреби приводити API згідно зі зміною вимог на етапі проектування, що істотно сприяє зручності внесення змін при розробці програми. |
| 3. | Недоліки | Орієнтований на статичну природу клієнтської частини | Порівняно складніше в розумінні, ніж REST. |

Використання GraphQL замість REST дозволяє підвищити продуктивність (ефективність) клієнт-серверної взаємодії через те, що в першому випадку у відповідях на HTTP-запити не передається надлишкова інформація. Мова GraphQL може використовуватися з будь-якими backend-інструментами (frameworks) і мовами програмування.

Ключова ідея GraphQL – отримати всю цікаву для нас інформацію одним HTTP-запитом. Даним шляхом вирішуються дві характерні для REST проблеми: Overfetching - завантаження надлишкових даних, що веде до підвищення обсягу трафіку і зниження чутливості застосунку; Underfetching - отримання недостатньої кількості даних, що веде до потреби створення додаткових запитів до серверної частини, а це також загрожує вищенаведеними наслідками.

Таким чином, GraphQL – це наступний крок розвитку технологій реалізації розподілених систем (веб-застосунків), який дозволяє досягти більшої продуктивності і зручності реконфігурування таких рішень, ніж традиційна технологія REST. Розглянемо приклад. Нехай ми хочемо по користувачеві з ідентифікатором "123" отримати інформацію наступного характеру: ім'я користувача, заголовки всіх його публікацій (постів), а також імена останніх трьох його підпісників. Тоді HTTP-POST GraphQL-запит і відповідна відповідь можуть мати вигляд, наведений в табл. 5.3.



Докладніше про принципи формування запитів GraphQL наведено [за посиланням](https://graphql.org/learn/).

**2 Опис змісту лабораторної роботи**

**Заняття 1:**

1. Дана робота носить індивідуальний характер та спрямована на знайомство з принципами взаємодії клієнта та сервера (між сервісами) за допомогою запитів у REST-архітектурі та у мові запитів GraphQl. Ознайомитись з теоретичним матеріалом.
2. Ознайомитись з ресурсом [SWAPI.dev](https://swapi.dev) та [його документацією](https://swapi.dev/documentation) для виконання HTTP-запитів
3. Ознайомитись з ресурсом [SWAPI GraphQL](http://graphql.org/swapi-graphql) та його [модифікацією](https://swapi-graphql.eskerda.vercel.app/), його [документацією](https://github.com/graphql/swapi-graphql), [схемою](https://github.com/graphql/swapi-graphql/blob/master/schema.graphql) та [її графічним відображенням](https://graphql-kit.com/graphql-voyager/) для формування graphQL запитів.
4. Виконати завдання у гугл-формі з лабораторної роботи 5, отримавши необхідну інформацію для відповіді на питання, наведені у формі, за допомогою HTTP-запитів на ресурсі [SWAPI.dev](https://swapi.dev) та GraqphQL-запитів на ресурсі [SWAPI GraphQL](http://graphql.org/swapi-graphql) чи його [модифікації](https://swapi-graphql.eskerda.vercel.app/). Порівняти обидва методи, необхідну кількість запитів для отримання потрібної інформації, складність формування запиту, кількість зайвої інформації у відповіді та зробити висновки щодо переваг та недоліків обох методів.
5. Форма до лабораторної роботи 5 повинна бути відправлена до дедлайну (до другого заняття з лабораторної) та зараховується як звіт з її виконання. При заповненні форми зверніть увагу, щоб обрана Вами форма відповідала Вашій групі, щоб вона не загубилась при оцінюванні.

**Заняття 2:**

* + - 1. Це заняття присвячено захисту лабораторної роботи 5 у форматі тестування.