ПЛАН

1. Вступ.
2. Основні характеристики технології.

* Загальна характеристика мови програмування JavaScript.
* JavaScript як серверна мова програмування. Node.JS.
* Express,js – фреймворк для написання веб-застосунків.
* Загальна характеристика REST API, та її призначення в сучасному програмному забезпеченні.
* Загальна характеристика GraphQL, його призначення у сучасному програмному забезпеченні.

1. Визначення технології REST API:

* Основні поняття REST API.
* Семантика протоколу HTTP.

1. Визначення технології GraphQL:

* Основні поняття GraphQL.
* Основні задачі GraphQL, та підходи до їх розв’язку.
* Порівняльна характеристика REST технології з GraphQL, їх переваги та недоліки.

1. Реалізація технології GraphQL, як альтернативи REST API.

* Основні переваги GraphQL, над REST API.
* Способи доступу до даних за допомогою GraphQL.

1. Висновок.
2. Список використаної літератури.

**ВСТУП**

У 2012 році всередині компанії Facebook розробники під час створення мобільного додатку Facebook для iOS та Android користувачів зіштовхнулись з новою для них проблемою. Проблема полягала в тому, що чим складнішими ставали їхні додатки, тим частіше вони виходили з ладу, також значно сповільнювалась робота самого додатку. Їм був потрібен достатньо потужний API для отримання інформації, який би дав змогу описати весь Facebook, але в той же час достатньо простий для вивчення та використання їхніми розробниками. Основною причиною, чому REST технологія не давала змоги вирішити їхні проблеми це те, що виникала занадто велика відмінність між даними, які вони хотіли використати в їхніх додатках, та запитами до сервера яких потребувала дана технологія. Крім цього виникали мережеві проблеми стосовно отримання даних такі, як: переповнення даними та недовантаження даних. Таке розчарування надихнуло трьох розробників цієї компанії Лі Байрона, Ніка Шрока та Дена Шафера на створення нового проекту, яким в решті і став GraphQL.

GraphQL був створений в 2012 році, і протягом трьох років активно використовувався компанією Facebook, а вже в 2015 році відбувся його перший публічний реліз. Дана технологія була розроблена, як альтернатива REST API, та іншим спеціальним веб-сервісним архітектурам. Основними причинами, чому на даний момент багато компаній відмовляються від REST технології, є наступні:

* Їх REST API став настільки складним, що це затримує та ускладнює розробку додатку.
* Вони хочуть відокремити Front-End та Back-End для пришвидшення розробки додатку.
* У компанії є мобільний клієнт, і вони турбуються про швидкість відповіді клієнту на його запит.
* Чіткість формування запиту, тобто клієнт отримає лише те, що йому потрібно і ніякої надлишкової інформації.

В даний момент існує декілька різних реалізацій даної технології. Головними конкурентами в цій сфері є Facebook та Apollo.

**Основні характеристики технології**

* 1. Загальна характеристика мови програмування JavaScript.

**Javascript** (**JS**) – це динамічна, об’єктно орієнтована, прототипна мова програмування, яка приносить інтерактивність веб-сайтам. Дана мова найчастіше використовується для створення сценаріїв веб-сторінок, що дає можливість на стороні клієнта взаємодіяти з користувачем. Також за допомогою javascript можна виконувати такі дії, як: керування браузером, асинхронний (паралельний) обмін інформацією з сервером, зміна структури та зовнішнього вигляду веб-сторінок.

Класифікують javascript, як прототипну, скриптову мову програмування з динамічною типізацією. Також дана мова, окрім прототипної підтримую також і інші парадигми програмування, а саме: імперативну та частково функціональну.

Основними напрямками в яких використовується javascript є наступні:

* Написання сценаріїв веб-сторінок для надання їм інтерактивності;
* Створення односторінкових веб-додатків;
* Створення мобільних додатків;
* Програмування на стороні сервера за допомогою Node.js.

На даний момент javascript є однією з найпопулярніших мов програмування в інтернеті. Це пояснюється зручним та багатим функціоналом, якими володіє дана мова, а саме:

* Об’єкти, з можливістю динамічної зміни типу завдяки механізму прототипів;
* Функції, як об’єкти першого класу;
* Обробка винятків;
* Автоматичне приведення типів;
* Автоматичне прибирання сміття;
* Анонімні функції.

Основну гнучкість даній мові надають такі властивості, як: виринання, замикання, контекст(контекст виконання), та делегування подій.

Всі ці властивості перетворюють дану мову на потужний механізм для розробки веб-застосувань.

Для зручності javascript містить декілька вбудованих об’єктів. До них належать: Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Також дана мова містить набір вбудованих операцій, які, грубо кажучи, не обов’язково є функціями або методами, а також набір вбудованих операторів, що керують логікою виконання програм.

Оскільки javascript надає програмісту досить великі можливості, то при розробці великих веб-додатків з використанням даної технології, критично важливим є доступ до інструментів налагодження програми. Ця необхідність потрібна тому, що браузери від різних виробників , дещо відрізняються у поведінці javascript коду, і реалізації DOM(Document Object Model) моделі. Сьогодні майже всі браузери підтримують інструменти налагодження програм, користуючись якими відстеження помилок при написанні коду значно спрощується. Також оскільки javascript є дуже популярною існують такі зручні інструменти, як:

1. ESLint – перевірка якості коду. Користуючись даним інструментом можна здійснити сканування коду, та знайти деякі проблеми в ньому.
2. Prettier – даний інструмент дозволяє програмістам забути про погано відформатований, нечитабельний код. Prettier зробить все форматування автоматично, потрібно лише надати ресурс з кодом.
3. Babel – це перекладач javascript коду до більш старих версій. Завдяки цьому застосуванню у розробників є можливість користуватись найновішим функціоналом мови, не турбуючись про те, що оточення не встигло реалізувати найновіший стандарт.

Оскільки js є інтерпретатором, без строгої типізації, то кожен блок сценарію написаний даною мовою інтерпретатор розбиває окремо. На веб-сторінках, коли треба комбінувати блоки js та HTML(Hypertext Markup Language) коду, то синтаксичні помилки знайти легше, якщо тримати увесь javascript код не в одному файлі, розбити його на багато малих пов’язаних між собою js файлів. Користуючись таким способом синтаксична помилка не спричиняє падіння всієї сторінки, а також при такій структуризації коду будь-які помилки відстежуються набагато простіше.

* 1. JavaScript як серверна мова програмування. Node.JS.

**Node.js –** це програмна платформа з відкритим кодом, для створення високопродуктивних мережевих додатків написаних мовою js. Дана технологія заснована на движку V8, який був розроблений компанією GOOGLE, для трансляції javascript коду в машинний код.

Node.js додає javascript можливість взаємодіяти з пристроями вводу-виводу через власний АРІ, підключати інші зовнішні бібліотеки, що були написані різними мовами програмування та забезпечувати звернення до них безпосередньо з js-коду. Найчастіше Node.js використовується для написання серверної частини додатку, виконуючи роль веб-сервера, але за допомогою даної технології можна створювати також і віконні додатки.

Завдяки Node.js додаток легше масштабується. Це все досягається завдяки тому, що при одночасному підключенню великої кількості користувачів до серверу, Node працює асинхронно. Тобто дана технологія розставляє пріоритети стосовно обробки запитів користувачів, завдяки чому ресурси розподіляються грамотніше. Така ідея розділу пріоритетів прийшла в голову розробнику Раяну Далу, оскільки виділення окремого потоку під кожне під’єднання було досить затратним відносно ресурсів. Дал придумав використовувати систему, що була орієнтована на події. Інакше кажучи, дана система повинна реагувати на дію або бездіяльність користувача і виділяти під це ресурс.

На даний момент дана технологія розвивається в дуже швидкому темпі, нараховуючи близько 200 000 пакетів.

Ще однією з причин чому Node.js є настільки популярною, це те що в ній реалізований дуже зручний пакетний менеджер npm, завдяки якому прописавши лише одну строку ми можемо довантажити всі необхідні для нашого проекту пакети. Цей пакетний менеджер є настільки гнучким, що дозволяє керувати як пакетами, які є локальними залежностями певного проекту, так і глобально встановленими інструментами js. Всі залежності, що інстальовані для локального проекту, зберігаються у файлі package.json. У цьому файлі кожна залежність (dependencies) може визначити діапазон дійсних версій, використовуючи схему семантичної версії, це дозволяє уникнути небажаних змін, а також надає розробникам змогу автоматично оновлювати пакети.

Отже, Node.js є хорошою альтернативою іншим серверним мовам програмування, оскільки асинхронне подієве оточення дає змогу зменшити затрати ресурсів на обслуговування великої кількості клієнтів. Також дана технологія дозволяю створювати масштабовані системи, завдяки тому, що жодна з функцій Node не працює напряму з потоками вводу-виводу.

* 1. Загальна характеристика REST API, та її призначення в сучасному програмному забезпеченні.

REST API (Representational state transfer) – це підхід до архітектури мережевих протоколів, що здійснюють доступ до інформаційних ресурсів. Даний підхід був описаний Роєм Філдінгом (одним з творців протоколу HTTP) у 2000 році. В своїй дисертації “Архітектурні стилі та дизайн мережевих програмних архітектур” він визначив теоретичну основу, під спосіб взаємодії клієнтів та серверів у Всесвітній павутині. Філдінг описав концепцію побудови розподіленого додатка, при якій кожен запит клієнта до сервера містить в собі вичерпну інформацію щодо бажаної відповіді сервера, а сервер в свою чергу не зобов'язаний зберігати інформацію про стан клієнта. Користуючись даним архітектурним стилем дані повинні передаватися у вигляді невеликої кількості стандартних форматів, таких як: JSON, XML, HTML та ін. Але як і будь-який архітектурний стиль REST API також має список власних обмежень. За Філдінгом є п’ять обов’язкових обмежень для побудови розподілених REST -додатків:

1. Модель клієнт-сервер;
2. Відсутність стану;
3. Кешування;
4. Однорідність інтерфейсу;
5. Шари абстракції;
6. Код на запит.

Останнє обмеження в списку не є обов’язковим, і якщо воно не надає переваг для конкретного застосування, то немає необхідності його реалізовувати. До прикладу, дозвіл на завантаження стороннього коду може бути небажаним з точки зору безпеки. Якщо ж не дотриматись хоча б одного з інших обмежень, то дану систему вже не можна вважати RESTful системою. В іншому випадку, якщо будуть дотримані всі обов’язкові обмеження, то система отримає ряд наступних переваг:

1. Надійність (за рахунок відсутності необхідності зберігати інформацію про стан клієнта);
2. Продуктивність (завдяки кешуванню даних);
3. Простота уніфікованого інтерфейсу;
4. Портативність компонентів системи шляхом перенесення програмного коду разом з даними;
5. Простота внесення змін (навіть при працюючому додатку);
6. Прозорість системи взаємодій між компонентами системи для сервісних служб;
7. Масштабованість для великої кількості компонентів та взаємодій між компонентами;
8. Здатність еволюціонувати, пристосовуючись до нових вимог (на прикладі Всесвітньої павутини).

REST технологія набула популярності не лише завдяки вищевказаним можливостям, а ще й тому, що є дуже простим інтерфейсом управління інформацією, без використання яких-небудь додаткових внутрішніх прошарків. Тобто дані не загортаються в XML, як це відбувається в SOAP та XML-RPC, а передаються в тому вигляді, в якому вони знаходяться.

Кожна одиниця інформації визначається глобальним ідентифікатором, таким як URL-адреса. В свою чергу кожна URL-адреса має суворо заданий формат. До прикладу вам потрібно взяти книгу, яка знаходиться на 3-ій позиції на книжній полиці, тоді ключ до неї, тобто її URL матиме наступний вигляд /book/3. Уявіть, що тепер вам потрібно відкрити дану книгу на 41 сторінці, тоді URL виглядатиме так /book/3/page/41. Завдяки уніфікованим глобальним ідентифікаторам доступ до даних здійснюється неймовірно просто. Причому абсолютно не важливо в якому форматі знаходяться дані за адресою /book/3/page/41 ­– це може бути як відсканована сторінка даної книги в форматі jpeg, так і текстовий документ Microsoft Word.

Управління інформацією сервісу цілком ґрунтується на протоколі передачі даних. Найбільш розповсюдженим протоколом звичайно ж є HTTP протокол. Для даного протоколу дії над даними здійснюється за допомогою наступних методів POST, GET, PUT, DELETE.

Отже, основною причиною чому технологія REST є такою популярною – це простота її використання. Адже по запиту, що надійшов на сервер можна одразу зрозуміти, що він робить. Також REST вважається менш ресурсо-затратним ніж вищезгадані SOAP та XML-RPC, оскільки запити не потрібно парсити для того, щоб зрозуміти що саме вони повинні робити, та не потрібно переводити дані з одного формату в інший.

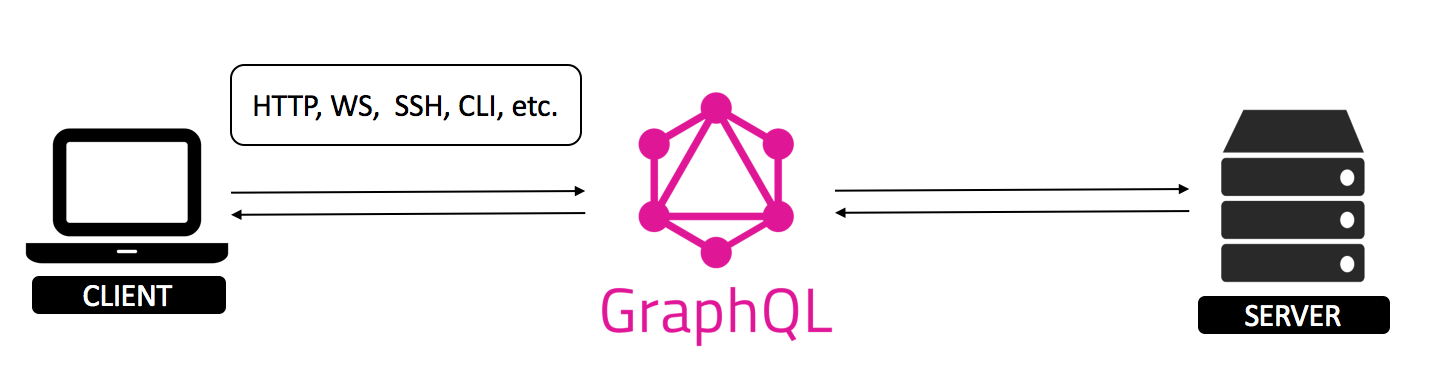
* 1. Загальна характеристика GraphQL, його призначення у сучасному програмному забезпеченні.

GraphQL - це мова запитів, призначена для побудови клієнтських додатків, синтаксис GraphQL інтуїтивно зрозумілий, та гнучкий у розумінні опису своїх вимог до даних та їх взаємодій.

GraphQL має три основні характеристики, які використовуються клієнтом для завантаження даних з сервера:

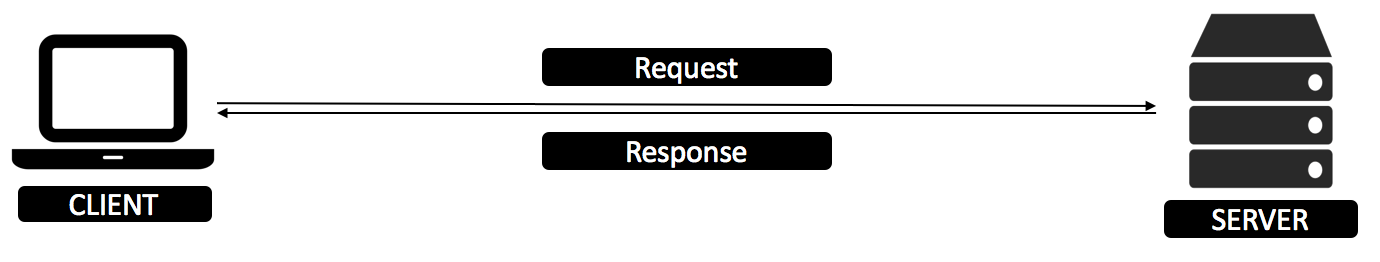
* GraphQL дозволяє клієнту чітко вказати, які дані йому потрібні.
* Полегшує агрегацію даних взятих з декількох джерел.
* Використовує систему типів для опису даних.

Основними причинами, чому дана технологія є хорошою альтернативою REST, це те, що використовуючи GraphQL клієнтам більше не має потреби декілька разів звертатись до сервера за різними даними, оскільки він дозволяє отримати всю необхідну для клієнта інформацію за один запит. Також використовуючи GraphQL клієнт спілкується з сервером за допомогою унікальної мови запитів, яка в свою чергу відміняє необхідність для сервера жорстко задавати структуру або склад даних, що повертаються сервером, а також не прив’язує клієнта до конкретного сервера. Фактично GraphQL знаходиться в середині між клієнтом і сервером. Оскільки клієнт запрошує в сервера дані, а той в свою чергу повинен відповісти на цей запит актуальними даними.



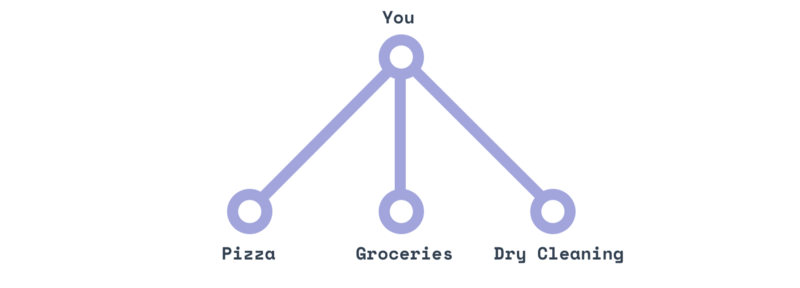
Тобто GraphQL в цьому випадку виступає посередником між клієнтом і сервером, або навіть особистим помічником клієнта стосовно запитів до сервера.

Саме в даному випадку у більшості виникає запитання, чому клієнт не може спілкуватися з сервером на пряму без посередників?

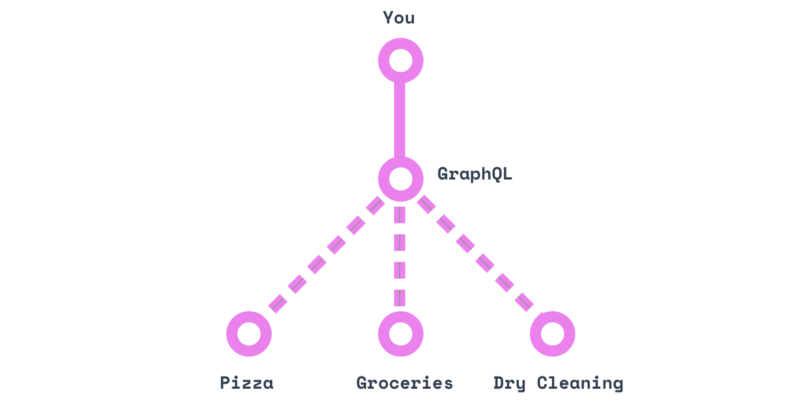


Відповідь очевидна, звичайно може, але використання GraphQL в цьому випадку значно спростить життя клієнту. Оскільки клієнт на сервері зазвичай отримує цілу купу ресурсів, але сервер віддає лише один ресурс за раз. Саме тому клієнт повинен багаторазово звертатись до сервера, щоб отримати всі необхідні йому дані. Користуючись GraphQL такої проблеми не виникає оскільки в цьому випадку він виступає особистим помічником клієнта. Наглядно це демонструє наступний приклад:

Уявіть, що вам потрібно замовити піцу, доставку продуктів, та забрати речі з хімчистки. Користуючись старим REST подібним підходом клієнту довелося б зробити три окремі запити по кожній з позицій, це виглядало б так.



На щастя, у нас є особистий помічник, якому лише потрібно передати адреси місць де знаходяться ваші дані, а потім ви можете запросити в нього всі дані одночасно. Це виглядає наступним чином.



Саме завдяки цим можливостям GraphQL і витісняє стару REST технологію.

**Визначення технології REST** **API**

2.1. Основні поняття REST API.

Як вже зазначалось, щоб додаток вважався RESTful-додатком, то в ньому повинні бути дотримані певні обмеження. Визначимо сутність кожного з цих обмежень.

1. **Модель клієнт-сервер.**

Оскільки першою архітектурою від якої REST успадковує обмеження є клієнт-серверна архітектура, то основним принципом, що лежить в основі даного обмеження є розмежування потреб. Суть його полягає у відмежуванні потреб інтерфейсу клієнта від потреб сервера, на якому зберігаються дані. Саме принцип розмежування підвищує переносимість клієнтського коду на інші платформи, в той час як спрощення серверної частини покращує масштабованість. Отже, розмежування дає змогу окремим частинам розвиватися незалежно одна від одної.

1. **Відсутність стану.**

Принцип даного обмеження полягає в тому, що взаємодії між сервером і клієнтом не мають стану. Кожен запит містить в собі всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту. Поняття відсутності стану не означає, що його немає, а означає лише те, що сервер не знає про стан клієнта. Саме тому в проміжок часу між запитами немає жодної інформації про стан клієнта, що зберігається на сервері. Наприклад клієнт може завантажити сторінку сайту, тоді сервер обробить цей його запит і забуде про клієнта. Через декілька хвилин клієнт може натиснути на інше посилання і тоді сервер знову обробить його запит і знову забуде про клієнта. В цей час сервер може обробляти запити інших клієнтів, але для нашого клієнта це немає ніякого значення. Завдяки даному обмеженню дані про стан сесії зберігаються на стороні клієнта, і передаються з кожним запитом. Сервер в свою чергу після завершення обробки запиту може звільнити всі ресурси, що були задіяні для цієї операції, без жодного ризику втратити цінну інформацію. Спрощується моніторинг, бо для того аби розібратись, що відбувається в певному запиті, достатньо подивитись лише на цей запит. Також збільшується надійність, оскільки помилка в одному запиті не зачіпає інші. Окрім плюсів дане обмеження має великий мінус. Даний мінус полягає в тому, що в кожен запит доводиться додавати дані сесії з клієнта, що значно знижує продуктивність. Також збереження стану на різних клієнтах складно підтримувати, бо реалізації клієнтів можуть відрізнятись, в той час як середовище сервера цілком контролює розробник.

1. **Кешування.**

Це обмеження полягає в тому, що дані, які передаються сервером повинні містити інформація про те, чи можна їх кешувати, і якщо можна, то як довго. Завдяки кешуванню збільшується продуктивність оскільки відбувається уникання зайвих запитів до сервера, але це в сою чергу зменшує надійність тому, що дані в кеші можуть бути застарілими.

1. **Однорідність інтерфейсу.**

Наявність уніфікованого інтерфейсу є фундаментальною вимогою REST-сервісів. Дане обмеження дозволяє кожному з сервісів розвиватися незалежно, саме тому їх легко модифікувати при потребі. Також на однорідні інтерфейси накладається ще чотири обмеження:

1. Ідентифікація ресурсів.

Це означає, що всі ресурси ідентифікуються в запитах. Наприклад з використанням URL в інтернет-системах.

1. Маніпуляція ресурсами через представлення.

Суть полягає в тому, що якщо клієнт зберігає представлення ресурсу з метаданими включно, то він має достатньо інформації для того, щоб модифікувати або видалити ресурс.

1. Самоописові повідомлення.

Кожне повідомлення містить достатньо інформації, щоб зрозуміти яким чином його можна обробити.

1. Гіпермедіа, як засіб зміни стану додатку.

Клієнти змінюють стан системи тільки через дії, які динамічно визначені в гіпермедіа на сервері. Самостійно клієнт не може визначити, що доступна якась операція над якимось ресурсом, якщо він не отримав інформацію про це в попередніх запитах до сервера.

1. **Шари абстракції.**

Зазвичай клієнт не може визначити, чи взаємодіє він напряму з сервером, чи з проміжним вузлом, у зв’язку з ієрархічною структурою мереж. Мається на увазі, що така структура утворює шари. Кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі над ним або в шарі під ним. Дане обмеження покращує масштабованість за рахунок збалансованого навантаження та розподіленого кешування. Обмеження знання системи одним шаром зменшує складність компонентів.

1. **Запитування коду.**

Це обмеження дозволяє розширити функціональність клієнта за рахунок завантаження коду з серверу у вигляді аплетів або сценаріїв. Філдінг стверджує, що дане обмеження не є обов’язковим, але дотримуючись його можна спроектувати архітектуру, що підтримує бажану функціональність.

Оскільки REST є архітектурним стилем, то він має власні архітектурні елементи.

Одним з цих понять є елементи даних. Компоненти REST системи спілкуються передаючи один одному представлення ресурсу в форматі, що вибирається з набору стандартних форматів даних. Даний формат обирається динамічно відповідно до бажань клієнта та можливостей сервера.

Ключовим елементом даних в REST є ресурс. Ресурс може бути представлений чим завгодно, це може бути текстовий файл, зображення, якесь динамічне значення, навіть щось з реального світу.

Для того, щоб мати змогу посилатись на ресурси, існує поняття ідентифікатора ресурсів. Компонент, що надав ресурсу ідентифікатор, і дозволяє доступитись до даного ресурсу за даним ідентифікатором, відповідає за збереження незмінного стану функції приналежності. Якщо подивитись на ресурс з іншого боку, то він і є цією функцією приналежності, що відображає моменти в часі на множину однотипних сутностей.

Ще одним елементом даних є представлення. Представлення – це послідовність байтів, та метадані представлення, які потрібні для опису цих байтів. Часто представлення знаходяться у вигляді документів, файлів, HTTP-повідомлень тощо. Також досить часто трапляється така ситуація, що метадані є не лише в представлення ресурсу, а й в самого ресурсу. Прикладом метаданих ресурсу може бути посилання на джерело в html тезі.

Контрольні дані – це дані, які описують ціль повідомлення між компонентами в представленні. До прикладу це може бути прохання про дію таке, як створення, зміна або видалення ресурсу. Також контрольні дані можуть виступати, як значення відповіді, тобто можуть містити інформацію про стан ресурсу, або ж описувати помилку. Ті контрольні дані, які включені в запити чи відповіді – можуть керувати поведінкою кеша.

Також в елементах даних існує таке поняття, як медіа типи. Медіа тип – це формат даних представлення. Одні медіа типи краще підходять для автоматичної обробки, в той час, як інші ­– для того, щоб бути показаними користувачу. Для того, аби поєднати кілька видів представлення в одному, можна скористатись композитними медіа типами. Від формату даних залежить прихованість застосунку, яку сприймає безпосередньо користувач. До прикладу браузер може почати відображення веб-сторінки, ще до того, як HTML код повністю завантажиться, це збільшує видиму швидкість роботи додатку.

Конектори в REST надають інтерфейс для комунікації між компонентами в даній системі. Завдяки конекторам є можливість приховати реалізацію ресурсів, та механізм комунікації. За своєю сутністю конектори подібні, до виклику віддалених процедур (RPC – Remote Procedur Call). RPC – полягає в тому, що дозволяє програмі запущеній на одному комп’ютері, звертатись до функцій програми, що виконується на іншому комп’ютері. Головний нюанс в якому полягає відмінність між конекторами та RPC – це передача параметрів та результат виклику. Параметри в свою чергу складаються з ідентифікатора ресурсу, контрольних даних та представлення (не є обов’язковим), а результат складається з контрольних даних відповіді та безпосередньо представлення. Найважливішими типами конекторів є клієнт та сервер. Їхня відмінність полягає в тому, що клієнт створює запит, в той час як сервер очікує на запити і відповідає на них відкриваючи доступ до своїх сервісів. Існує також додатковий тип конектора – кеш. Кеш може знаходитись як і на клієнтський стороні, для уникнення створення зайвих запитів, так і на серверній – для уникнення зайвого обчислення відповіді на певний запит. Оскільки одним з обов’язкових обмежень в REST є однорідність інтерфейсу, то кеш може легко дізнатись чи можна кешувати запит. За промовчанням, відповідь на запит отримання ресурсу кешувати можна, а запити зміни ресурсу кешувати заборонено. Проте всі ці налаштування можна змінити за допомогою контрольних даних.

Резолвер – це ще один тип конектора. Резолвер перетворює ідентифікатори ресурсів в інформацію про мережеві адреси, яка є необхідною для компонентів щоб отримати цей ресурс. Наприклад можна розглянути URL в якому міститься доменне ім'я, і для доступу до відповідного домена, потрібно дізнатись адресу в DNS-сервера. В цьому конкреному випадку DNS грає роль резолвера. Використання одного або декількох резолверів здатне збільшити життєздатність ідентифікатора ресурсу, оскільки він не вказує на фізичне розташування ресурсу, яке може змінитися.

Останньою формою конектора в REST є тунель. Основним завданням тунеля є проведення запитів через межу системи, наприклад через фаєрвол. Головною причиною чому тунелі включені до REST архітектури є те, що певні компоненти системи можуть перетворюватися в тунелі по запиту. Це чітко можна побачити на прикладі HTTP-тунелю, який активується при отриманні запиту з методом CONNECT.

2.2. Семантика протоколу HTTP.

HTTP (Hyper Text Transfer Protocol) – це протокол передачі даних, що використовується в комп’ютерних мережах. Даний протокол належить до протоколів моделі OSI 7-го прикладного рівня.

В REST архітектурі ресурс може бути представлений чим завгодно, але дії які можна виконувати над ресурсом визначаються в повідомленнях які описуються стандартним протоколом. В системі WWW цей протокол – HTTP, хоча існують REST архітектури на основі інших мережевих протоколів.

Стандарт HTTP визначає 8 типів повідомлень.

**GET** – це тип повідомлення, який запрошує вміст вказаного ресурсу. Ресурс, що запитується може приймати параметри. Усі параметри передаються за допомогою URL. Згідно з HTTP стандартом запити типу GET вважаються ідемпотентними. Це означає, що багаторазове виконання одного й того самого запиту повинно вертати однаковий результат, за умови, що в проміжок часу між запитами ресурс не змінювався. Завдяки даній властивості дозволяється кешувати відповіді на запити типу GET.

**OPTIONS** – повертає HTTP методи, які підтримуються веб-сервером. Зазвичай цей метод використовується для визначення можливостей веб-сервера.

**HEAD** – це метод який за своїм призначенням аналогічний до методу GET, за винятком того, що у відповіді сервера на даний метод відсутнє тіло. Цей метод корисний для витягнення мета-інформації, що задана в заголовках відповіді, без пересилання всієї інформації. Тобто завдяки цьому методу, клієнт може отримати лише заголовки які б відсилалися разом з представленням ресурсу, але не саме представлення ресурсу.

**POST** – цей метод передає призначені для користувача дані заданому ресурсу. Тобто даний метод створює новий ресурс, використовуючи передане в тілі представлення.

**PUT** – цей метод змінює стан поточного ресурсу представленням, що передається.

**DELETE –** цей метод видаляє вказаний ресурс. Також належить до ідемпотентних методів. Хоча після другого виклику даний метод поверне статус 204 No Content, а потім і 404 Not Found, але ресурсу не буде, як після першого видалення так і після 10-го.

**TRACE** –повертає отриманий запит таким чином, що клієнт може подивитися, що саме проміжні сервери додають в запит або змінюють в ньому.

**CONNECT** – це метод, який призначений для використання разом з проксі-серверами, які мають змогу динамічно перемикатися в тунельний режим.

Існує також ще 9-ий метод, який щоправда не описаний в HTTP, але описаний в його додатку RFC 5789.

**PATCH –** це метод, який змінює лише частину ресурсу на основі переданого представлення. Якщо якась частина ресурсу не згадується в переданому представленні, то це означає що її змінювати не потрібно. Завдяки даному методу зменшується кількість інформації, яку потрібно передати для зміни ресурсу.

В протоколі HTTP використовуються такі коди статусів:

* **1хх** – інформаційний. Запит прийнято, продовжуй процес. Сервери не повинні відсилати клієнтам 1хх відповіді, за виключенням експериментальних умов.
* **2хх** – успіх. Цей клас кодів стану вказує на те, що запит клієнта було успішно прийнято і зрозуміло.
* **3хх** – перенаправлення. Даний клас кодів вказує на те, що майбутні дії повинні бути виконані клієнтом, для успішного завершення запиту.
* **4хх** – помилка на стороні клієнта. Цей клас кодів призначений для визначення помилок на стороні клієнта (окрім випадку коли методом запиту був HEAD).
* **5хх** – помилка на стороні сервера. Даний клас кодів вказує на те, що сервер знає про те, що виникла помилка, або у випадку коли сервер не має змоги опрацювати запит клієнта.

Основними статусами, що використовуються в даному протоколі є:

* 200 (**ОК**) – запит виконано успішно. Інформація, яка вертається у відповіді напряму залежить від методу, що використовувався в запиті.
* 301 (**Moved Permanently**) – ресурс переміщено. Даний код вказує на те, що ресурсу було назначено нову URL-адресу, і всі наступні звернення до цього ресурсу повинні здійснюватися по новій адресі. Новий URL вказується в полі Location заголовка.
* 403 (**Forbidden**) – доступ до запитуваного ресурсу заборонений. Сервер зрозумів запит, але відмовляється його виконувати через обмеженість доступу для клієнта, який звертається до даного ресурсу.
* 404 (**Not Found**) – ресурс не знайдено. Сервер не знайшов жодних ресурсів по вказаному URL, також немає жодних вказівок про те, чи це постійний стан, чи ні. Слід використати статус 410 (**Gone**), якщо серверу відомо, що старий ресурс недоступний постійно і сервер немає адреси для пересилання.
* 503 (**Service Unavailable**) – немає доступу до сервісу. Сервер не може опрацювати запит через тимчасове перевантаження, або ж з приводу технічних робіт. Це тимчасовий стан з якого сервер вийде через деякий проміжок часу. Якщо цей проміжок часу відомий, то його можна вказати в заголовку **Retry-After**.