Технічні науки

УДК 004.51

Дьомін Богдан Олександрович

студент

Національного технічного університету України

«Київський національний інститут імені Ігоря Сікорського»

Domin Bohdan

Student of the

National Technical University of Ukraine

«Igor Sikorsky Kyiv Polytechnic Institute»

Науковий керівник:

Коваленко Олександр Сергійович

доктор технічних наук, професор кафедри біомедичної кібернетики

Національний технічний університет України

«Київський національний інститут імені Ігоря Сікорського»

Консультант:

Аверьянова Ольга Анатоліївна

старший викладач кафедри біомедичної кібернетики

Національний технічний університет України

«Київський національний інститут імені Ігоря Сікорського»

ПАГІНАЦІЯ ТА ЇЇ ЕФЕКТИВНІ МЕТОДИ ДЛЯ ВЕБ-СЕРВІСІВ RESTFUL

PAGINATION AND ITS EFFICIENT METHODS FOR RESTFUL WEB SERVICES

Анотація. *У цій статті, на основі проаналізованих ресурсів і накопичених ідей IT-індустрії, піднімається проблема загальних підходів до*

пагінації у веб-сервісах RESTful, а також шляхи подолання за допомогою методу, який поєднує звичайні підходи, прикриваючи їх недоліки.

Ключові слова: *пагінація, сторінка, RESTful, дані, кінцева точка.*

Summary. *In this paper on the basis of analyzed resources and accumulated IT-industry discuss the problem of the common approaches to pagination in RESTful web services, as well as the ways of overcoming it by using a method that fuses usual approaches while covering their weaknesses.*

Key words: *pagination, page, RESTful, data, endpoint.*

Вступ. З кожним днем Інтернет-технології все більше інтегруються в життя людей. Постійно з'являється багато нових веб-сервісів, а старі оновлюються. Оскільки більшість цих служб створено для людей, вони мають бути ефективними з точки зору взаємодії з користувачем і зручності використання. Для цього було створено багато методів, які називаються «шаблони проектування інтерфейсу користувача» [1]. Деякі з таких «шаблонів», у тому числі детально розглянутий у цій статті, створені, щоб приховати специфіку або обмеження технічних реалізацій, тоді як інші – спрямовані виключно на збагачення користувацького досвіду. Деякі шаблони можуть призвести до додаткових витрат, якщо з ними поводитися необережно. Прикладом такого випадку є пагінація. У цій статті ми обговоримо потенційні недоліки звичайних методів, що використовуються для розбиття сторінок у веб-службах RESTful, а також способи їх вирішення. Підходи, представлені в цій статті, не є абсолютно новими, однак наразі існує небагато авторитетних ресурсів, які забезпечують баланс між узагальненим агностичним описом мови програмування/технології та достатньою кількістю деталей технічної реалізації щодо методів розбиття на сторінки. Навіть судячи з літературних джерел, використаних у цій роботі, можна побачити, що більшість ресурсів зосереджено на конкретній

технології чи системі, де використовується пагінація, практично обмежуючи аудиторію, яка може споживати та застосовувати цю інформацію, лише тими, хто знайомий із цією конкретною технологією, на яку орієнтований ресурс. Таким чином, мета цієї статті полягає в тому, щоб зібрати, обробити та представити технологічну агностичну інформацію про підходи до розбивки сторінок, які застосовуються в сучасних веб-сервісах RESTful, а також підкреслити сильні та слабкі сторони кожного підходу з точки зору продуктивності та показати можливі шляхи модифікації та поліпшення.

Пагінація — це шаблон проектування, який забезпечує спосіб розбиття дуже довгого списку записів даних (рядків) на сторінки, що відображаються по одній. Пагінація також включає спеціальні кнопки керування (або просто елементи керування), які дають користувачеві можливість переходити до різних сторінок списку.

Приклад нумерації сторінок можна побачити на рис. 1.

Рис. 1. Приклад нумерації сторінок

Пагінація ділить список даних на частини, показуючи лише певні частини користувачам по черзі, щоб не перевантажувати їх надмірною інформацією. Фрагмент даних, який відображається користувачеві, називається «сторінкою». Ця техніка дозволяє користувачеві вирішити, чи хочуть вони бачити більше сторінок, чи їх задовольняють відображені результати.

Пагінація дуже поширена в Інтернеті. Серед величезної кількості веб-додатків розбиття на сторінки також використовується для відображення результатів у пошукових системах (див. рис. 2).

Рис. 2. Приклад елементів керування розбивкою на сторінки Google

Кількість результатів (елементів, рядків, записів), представлених на сторінці, зазвичай залежить від розміру екрана, який можуть мати користувачі, обсягу місця, який кожен елемент результату займає на екрані, часу, витраченого на кожен елемент завантаження та відтворення, а також ймовірність того, що користувач зможе знайти потрібні результати на першій сторінці.

Остання частина важлива: користувачі в ідеалі не повинні переходити далі першої сторінки або перших кількох сторінок, оскільки ймовірність того, що користувачі розчаруються та втратять інтерес, зростає з кожною наступною сторінкою, на яку вони переходять, не знайшовши того, що шукали.

З точки зору корисного та зручного для використання інтерфейсу користувача, розбиття на сторінки має забезпечувати певні елементи керування. Зазвичай вони розташовані внизу сторінки, але іноді вони також можуть з'являтися вгорі.

Елементи керування розбивкою сторінок мають містити такі елементи:

• Посилання «Попередній» і «Далі», які зазвичай мають стрілку або трикутник для виділення. Коли користувач перебуває на першій сторінці, посилання «Попереднє» вимкнено. Якщо користувач перебуває на останній сторінці, посилання «Далі» стає недоступним.

• Посилання на першу сторінку має бути завжди доступним, оскільки перша сторінка має містити найбільш відповідні результати, до яких користувач може захотіти повернутися.

• Послідовність пронумерованих посилань на різні сторінки. Немає необхідності посилати сторінку, на якій зараз перебуває користувач. Замість цього рекомендується відображати його контрастним кольором і іншим розміром шрифту, щоб дати користувачеві підказку про його поточне місцезнаходження.

• Еліпси, щоб пропускати частини послідовності, якщо на панелі керування занадто багато сторінок (наприклад, більше 10). Рекомендується зберегти посилання на першу сторінку, на останню сторінку (якщо остання сторінка відома), посилання на попередню сторінку або дві, і, аналогічно, посилання на наступні. Решту можна опустити.

Крім того, елементи керування розбивкою на сторінки можуть додатково включати загальну кількість сторінок, якщо вона відома. Кількість сторінок може бути представлено посиланням на останню сторінку або також можна показати короткий підпис із зазначенням, наприклад, «Сторінка 1 із 100» або подібне [1].

Деякі приклади елементів керування розбивкою на сторінки представлені на рис. 3–4.

Рис. 3. Приклад елементів керування розбивкою на сторінки. Компонент розбиття на сторінки Vuetify [2]

Рис. 4. Пагінація способом визначення кількості елементів на сторінці

Загальна реалізація пагінації. Реалізація розбиття на сторінки,

безсумнівно, дуже залежить від використовуваних технологій. Незважаючи на те, що теоретично цей шаблон проектування може мати вільну форму реалізації, деякі загальні підходи до розбиття на сторінки, які залежать від сучасних популярних веб-технологій і архітектур, таких як HTTP [3] і REST [4], вже сформувалися протягом досить тривалого часу.

Розглядаючи сучасні веб-розподілені системи, які використовують дворівневу клієнтську архітектуру або більш складні, завжди є сервер, який обробляє та забезпечує доступ до всіх даних [5]. Той факт, що доступ до даних здійснюється виключно через сервер, тоді як клієнт, як правило, є лише середовищем, яке забезпечує перегляд отриманих даних, означає, що якщо запитувані дані з сервера відповідають вимогам можливості розбиття на сторінки (описано вище), сервер відповідає за розбивку даних на сторінки та повернення їх клієнту у вигляді сторінок. Потім клієнт може надати дані та за потреби запитати додаткові сторінки. Отже, для того, щоб такий процес відбувся, повинен існувати заздалегідь визначений набір правил зв'язку між клієнтом і сервером. У випадку архітектури REST це відбувається через кінцеві точки.

У веб-службах RESTful, коли потрібна розбивка даних на сторінки, створюється спеціальний ресурс URI (кінцева точка). Цю кінцеву точку можна встановити будь-яким бажаним способом, але зазвичай вона повертає першу сторінку результатів за першим запитом. Так зване «наступне» посилання [6], яке вказує на наступну сторінку даних, також зазвичай передається разом зі сторінкою даних. Після запиту посилання «наступна» повертається наступна сторінка разом із новим посиланням «наступна», що вказує на наступну послідовну сторінку і так далі. У випадку останньої сторінки даних посилання «наступна» дорівнюватиме порожньому значенню або нулю (оскільки наступна сторінка недоступна).

Прикладом кінцевої точки з розбивкою на сторінки може бути таке посилання: «http://www.someserver-0948.com/items» (це посилання є лише

уявним прикладом і не є справжньою службою RESTful). Якщо запит GET надіслано до даної кінцевої точки, він поверне дані, як показано на рис. 5. Дані представлені у форматі JSON.


```
{
    "count": 12,
    "page_size": 5,
    "next": "http://www.someserver-0948.com/items/2",
    "previous": null,
    "results": [
        {
            "name": "item1",
            "content": "test1"
        },
        {
            "name": "item2",
            "content": "test2"
        },
        {
            "name": "item3",
            "content": "test3"
        },
        {
            "name": "item4",
            "content": "test4"
        },
        {
            "name": "item5",
            "content": "test5"
        }
    ]
}
```


Рис. 5. Приклад розбитої на сторінки відповіді у форматі JSON

Поле «наступне» на рис. 5 вказує на другу сторінку розбитого на сторінки списку даних. На рис. 5 також є: поле «count», що представляє загальну кількість результатів (елементів) у всьому списку даних, «page_size», що відповідає кількості елементів, представлених на кожній сторінці, поле «next», яке вже обговорювалося вище, «previous» – поле, подібне до «next» з тією різницею, що воно представляє попередню сторінку замість наступної, – і «results», що містять 5 елементів (як визначається параметром «page_size») списку даних. Усі ці поля не підпадають під дію жодних стандартів і можуть бути присутніми або відсутніми залежно від реалізації.

Також варто звернути увагу на те, як реалізовано посилання на наступну сторінку. Придивившись до посилання «http://www.someserver-0948.com/items/2», можна помітити, що цифра «2» в кінці означає другу

сторінку. У наведеному зразковому варіанті реалізації, якби клієнт мав потребу запитати певну сторінку заздалегідь, не проходячи всі попередні, йому довелося б змінити останній номер у посиланні на номер потрібної сторінки. Таким чином, посилання на 12-ту сторінку буде «http://www.someserver-0948.com/items/12». Якщо потрібно отримати більше 5 рядків на сторінку, тоді можливо (якщо це визначено реалізацією) надати параметр URL-адреси, наприклад «page_size», який визначатиме новий розмір сторінки. Посилання для отримання 12-ї сторінки з розміром 10 рядків на сторінці може виглядати так: «http://www.someserver-0948.com/items/12?page_size=10». Однак точний спосіб отримання номера або розміру сторінки суворо залежить від реалізації. Він може існувати, а може й ні, або мати іншу структуру чи назву.

Іншим прикладом реалізації кінцевої точки може бути кінцева точка на основі зміщення [7] зі сторінками, визначеними таким чином:

- "http://www.someserver-0948.com/items?limit=5&offset=0" – отримати першу сторінку
- "http://www.someserver-0948.com/items?limit=5&offset=5" – отримати другу сторінку
- "http://www.someserver-0948.com/items?limit=5&offset=10" – отримати третю сторінку

У цьому випадку параметр URL-адреси «limit» визначає кількість рядків, які відображаються на одній сторінці з розбивкою на сторінки, тоді як offset визначає кількість або рядки, які потрібно пропустити з початку списку даних. Наприклад, якщо розмір сторінки становить 5 рядків на сторінці, то для відображення третьої сторінки перші 10 рядків списку даних потрібно пропустити, а наступні 5 рядків повернути [8].

Види реалізації пагінації. Як описано в попередньому розділі, пагінація зазвичай виконується на стороні сервера. Це називається серверною пагінацією.

Діаграма розмітки сторінок на стороні сервера представлена на рис. 6. Вона показує, що для того, щоб отримати кілька сторінок, необхідно відправити кілька запитів до сервера.

З іншого боку, деякі інтерфейсні фреймворки (фреймворки на стороні клієнта) мають вбудовану опцію розбиття на сторінки, яка передбачає певний вид розбиття сторінок на клієнті. Наприклад, Vuetify Material Framework [9] має компоненти розбивки на сторінки та автоматичну розбивку на сторінки таблиць даних. Крім того, Vuetify у своїй документації, а також інші фреймворки, такі як Material UI [10], відрізняють пагінацію на стороні сервера від пагінації, виконаної на клієнті, яка далі в цій статті називається «пагінацією на стороні клієнта».

Пагінація на стороні сервера є простою – клієнт запитує певну сторінку від сервера та отримує лише цю сторінку в результаті. Уся логіка, яка бере список даних і розбиває його на сторінки, які обслуговують лише запитані, виконується на сервері. У разі пагінації на стороні клієнта, серверу пропонується отримати та повернути весь список даних (а не лише окрему сторінку), а потім клієнт виконує код, який розбиває дані клієнта на сторінки та показує запитувану сторінку користувачеві. Подальші запити до сервера не надсилаються, оскільки всі дані вже доставлені клієнту (див. рис. 6).

Схема клієнтської пагінації представлена на рис. 7.

З наведеної нижче діаграми можна помітити, що весь список даних повертається після надсилання запиту до певної призначеної кінцевої точки («http://www.someserver-0948.com/items»), а сама розбивка сторінок відбувається на клієнті.

Описавши пагінацію на стороні сервера та клієнта, можна визначити очевидні недоліки кожного з них.

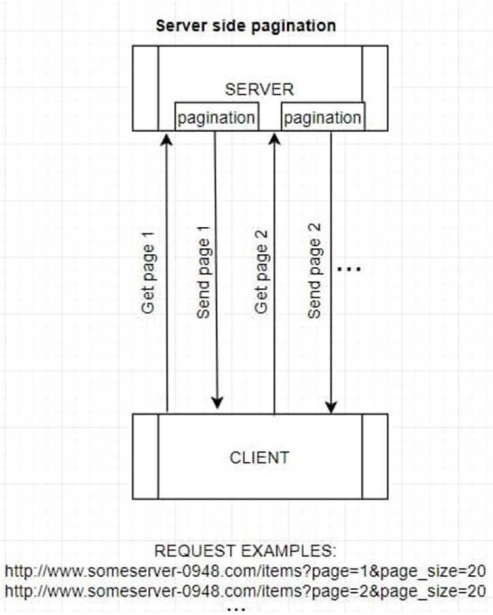

Рис. 6. Діаграма пагінації на стороні сервера

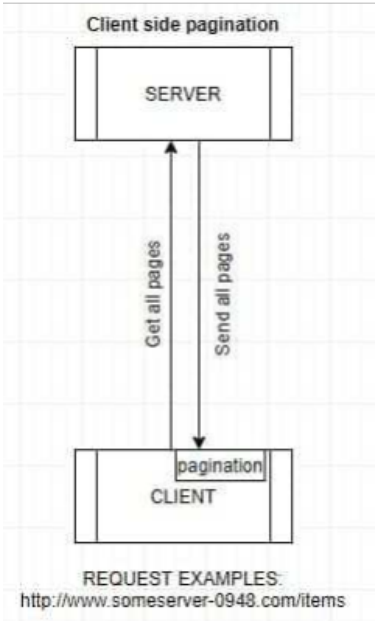

Рис. 7. Діаграма розбивки сторінок на стороні клієнта

У разі розбиття сторінок на стороні сервера – окремий запит надсилається на сервер щоразу, коли клієнт запитує сторінку, як це можна побачити на рис. 6. Після цього клієнту повертається лише одна сторінка даних. Таким чином, кількість запитів може зрости до значної кількості – враховуючи, що користувач може часто перемикається між сторінками і користувачів може бути багато.

Пагінація на стороні клієнта добре працюватиме для невеликих наборів даних. Однак для дуже великих і особливо для «нескінченних» може стати неможливим для завантаження всіх даних одночасно (див. перший розділ статті). Крім того, величезні набори даних займуть багато пам'яті клієнта.

Щоб підвищити ефективність, краще використовувати техніку кешування. Техніку кешування можна використовувати з розбивкою сторінок на стороні сервера. Коли клієнт запитує певну сторінку, сервер надсилає сторінку, а також кешує попередню та наступну сторінки у своїй пам'яті, щоб отримати доступ до них якомога швидше, якщо наступний запит спробує їх отримати.

Хоча кешування на сервері може допомогти зробити пошук сторінки набагато швидшим, це не зменшує кількість запитів. Ще один можливий підхід включає надсилання клієнту більшої кількості даних, ніж було спочатку запитано. Про такий спосіб і піде мова далі.

Змішаний спосіб розміщення сторінок. Ефективним способом розбиття сторінок може бути поєднання клієнтської та серверної розбивки сторінок. Сервер розбиває на сторінки та надає дані, подібно до методу розбиття на стороні сервера. Однак головне, щоб клієнт запросив інформацію не лише для поточної сторінки, а і далі. Наприклад, якщо клієнт запитує 50 рядків даних, тоді у випадку змішаного підходу може бути кориснішим замість цього отримати кожен фрагмент із 500 рядків із сервера. Потім клієнт розбиває на сторінки отриманий набір даних і показує лише ту сторінку, яку вибрав користувач. У випадку, коли користувач намагається відвідати наступну сторінку, запит до сервера не потрібен, оскільки клієнт уже містить інформацію для наступних 9 сторінок (500 рядків у фрагменті поділено на 50 рядків на сторінці мінус 50 рядків першої сторінки). Щойно користувач переходить більше ніж на 10 сторінок вперед, перевищуючи межі сторінок, повернутих із сервера, потрібно зробити додатковий запит до

сервера, щоб отримати ще 500 рядків даних (500 рядків, отриманих із попереднього запиту, можуть бути видалені). Отже, у цьому випадку клієнт надіслав лише два запити, хоча він зміг відобразити 20 сторінок – це вже показує, як можна мінімізувати кількість запитів, що є недоліком розбиття сторінок лише на стороні сервера. Немає навіть необхідності порівнювати це, оскільки у випадку простої (наївної) серверної пагінації кількість запитів сервера на завантаження 20 сторінок буде в 10 разів більша (оскільки кожен запит отримуватиме лише одну сторінку, яку запитував клієнт). Крім того, обсяг даних, що надсилаються назад клієнту у випадку змішаної техніки розбиття на сторінки, становить лише 500 записів на запит. Враховуючи, що записи даних мають простий формат, час завантаження та рендерингу цих даних не займатиме надто багато часу. Однак це, безумовно, залежить від самих даних – тому кількість у 500 рядків було вибрано лише як приклад, і вона може і повинна змінюватися залежно від конкретної веб-служби та форматів даних.

Нижче на рис. 8 наведено діаграму підходу змішаного розбиття на сторінки. Це схоже на діаграму серверної пагінації, але використовує більші фрагменти даних. Крім того, змішаний підхід розбиває на сторінки отриману частину даних на клієнті, оскільки частина даних може бути занадто великою, щоб відобразити її відразу.

Важливим фактом є те, що описаний змішаний підхід можливий і правильний з точки зору цілісності даних і безперебійного відображення даних лише в тому випадку, коли розмір сторінки сервера ділиться на розмір сторінки клієнта без залишку (500 ділиться на 50 без залишку). Ці числа потрібно контролювати, щоб цей підхід не пропускав жодного рядка або не показував незавершені сторінки під час перегляду даних із розбивкою на сторінки. Як, наприклад, якщо кількість сторінок записів на сторінці дорівнює 30, а кількість записів, повернутих із сервера для одного запиту, знову дорівнює 500. Тоді, якщо 500 поділити на 30, це дорівнюватиме 16

повним сторінкам (30 записів даних на кожна) і 20 записів залишаться на 17-й сторінці. Тому після того, як користувач перейшов на 17-ту сторінку, він побачить неповну сторінку, що може викликати деяку плутанину.

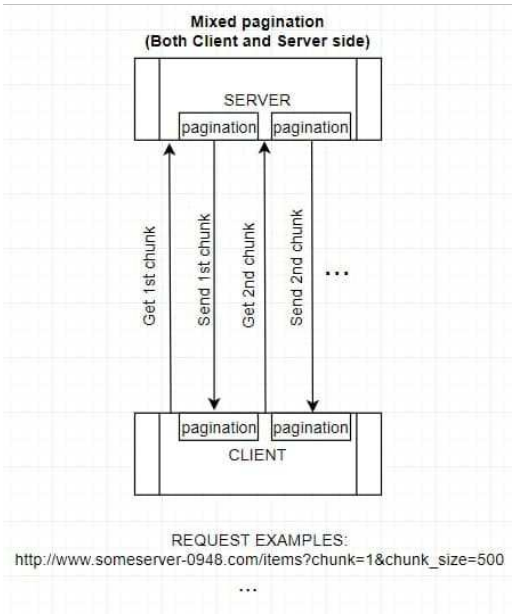

Рис. 8. Діаграма змішаної пагінації

Останнє застереження полягає в тому, що якщо користувач занадто довго залишається в межах сторінок, які були повернуті із сервера, то дані в базі даних можуть тим часом змінитися. Оскільки клієнт не надсилатиме запит до сервера, доки не буде перевищено межі діапазону сторінок (наприклад, доки користувач не перевищить 20-ту сторінку), записи даних залишатимуться застарілими, і користувач не отримає жодного оновлення даних. Щоб подолати це, клієнт має зберегти час останнього запиту сервера, і якщо цей час перевищує поточний час на певну кількість хвилин або секунд, сервер запитується ще раз, щоб отримати найновіші дані.

Загалом, цей метод зменшує навантаження на сервер, яке потенційно може бути спричинене великою кількістю запитів, а також не завантажує всі дані одночасно. Таким чином, це хороший спосіб обійти недоліки підходів до паганіції на стороні сервера та клієнта [11].

Висновок. Пагінація — важливий шаблон у дизайні інтерфейсу користувача, який допомагає обійти технічну нездійсненність величезного

завантаження набору даних. Незважаючи на те, що цей шаблон широко використовувався протягом досить тривалого часу, він не передбачає дотримання жодних стандартів щодо його реалізації, тому розробники можуть вільно створювати його будь-яким способом, який вони вважають за потрібне. Через це не завжди очевидно, які можливі варіанти та який із них найкраще вибрати, особливо з точки зору продуктивності. Крім того, багато зазначених ресурсів надмірно зосереджені на певній технології, коли йдеться про розбиття на сторінки. Тому в цій статті прояснено цю ситуацію шляхом збору та узагальнення наявної інформації. Розглядаючи та аналізуючи поточні тенденції IT-індустрії, було названо два основних підходи до розбиття на сторінки: на стороні клієнта та на стороні сервера. Хоча вищезазначені підходи досить поширені та найбільш інтуїтивно зрозумілі, вони, на жаль, мають свої недоліки. Однак метод, запропонований у цьому документі (змішаний метод реалізації розбиття на сторінки), може бути використаний для усунення таких слабких місць, роблячи більш ефективним і продуктивним розбиття на сторінки, використовуючи підходи до розбиття сторінок як на стороні сервера, так і на стороні клієнта. Цей метод заснований на кешуванні та відповідає пропозиціям, які обговорюються на професійних платформах серед розробників програмного забезпечення та інженерів [11]. Цей метод не залежить від технологій і описаний тут більш конкретно з точки зору основної ідеї, яка забезпечує більш детальне рішення до проблеми продуктивності сторінки.

Література

1. Tidwell J. Designing Interfaces. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2011. P. 224–228.
2. "Veutify documentation," Section: Pagination, URL: https://vuetifyjs.com/en/components/paginations/, May 2021

3. Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. Hypertext Transfer Protocol. Derived from HTTP/1.1, Internet RFC 2616, Fielding, Revision: Dan Connolly, June 1999.
4. Fielding R. T. Representational State Transfer (REST). 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
5. "Distributed Application Architecture". Sun Microsystem. Archived from the original on 6 April 2011. Retrieved 2009-06-16.
6. Richardson L., Amundsen M. "RESTful Web APIs" Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., September 2013. P. 101–102.
7. "BoxDEV documentation", Section: Offset-based Pagination. 2021. URL: https://developer.box.com/guides/api-calls/pagination/offset-based/
8. Gastón C. Hillar BuildingRESTfulPython Web Services. Published by Packt Publishing, Ltd. Livery Place, 35 Livery Street, Birmingham. B3 2PB, UK., ISBN 978-1-78646-225-1. October 2016. P. 140-145.
9. "Veutify documentation", Section: Tables/Data tables. May 2021. URL: https://vuetifyjs.com/en/components/data-tables/#external-sorting/
10. "Material-UI documentation," Section: Data Grid – Pagination. Version 4.11.1. URL: https://material-ui.com/ru/components/data-grid/pagination/,
11. "Pagination: Server Side or Client Side" discussion topic by jrutter. StackOverflow public web platform. URL: https://stackoverflow.com/a/408137