DOI 10.34185/1562-9945-3-158-2025-17
УДК 004.65

А.С. Степаненко, І.О. Саяпіна

МОДИФІКОВАНИЙ МЕТОД КУРСОРНОЇ ПАГІНАЦІЇ ТА ФІЛЬТРАЦІЇ ДАНИХ У ВИГЛЯДІ ПАКЕТУ NPM

Анотація: Ця стаття присвячена розробці модифікованого методу курсорної пагінації, що має підтримку основних баз даних та ORM систем, надає можливість використання комплексних фільтрації та сортування даних, індексації запитів тощо. Проаналізовані наявні рішення, випадки їх використання, переваги та недоліки. Сформовано модифікований метод, який містить в собі видозмінені та покращені функціональні можливості існуючих NPM пакетів, разом з новими впровадженнями, що зосереджені на збільшені ефективності та універсальності запитів до баз даних. Проведено аналіз запропонованого рішення. Наведено потенційні напрями подальших розширень та поліпшень методу.
Ключові слова: курсорна пагінація, база даних, ORM система, фільтрація даних, індексація запитів, NPM пакет, нечіткий пошук, сортування злиттям та купою, бінарне дерево, хеш-функція.

Постановка проблеми. Кожного дня все більше й більше електронних магазинів, каталогів та додатків з'являється в мережі Інтернету. Більшість з подібних ресурсів містить доволі велику кількість даних (наприклад, перелік інструментів та автомобільних запчастин на відповідному вебсайті), і є необхідність в їх зручному відображенні на вебсторінці. Одним зі способів розбиття даних на невеликі «набори» є пагінація. Однак при виборі методу пагінації, що найкраще підходить під задачі конкретного вебресурсу, виникає проблема, пов'язана зі складністю імплементації. Сьогодні найпопулярнішими методами є офсетна та курсорна пагінації [1]. Проте, на відміну від офсетного підходу, методи реалізації якого доволі широко відомі та розповсюджені, реалізація курсорної пагінації (або пагінації за токеном, як її ще називають) вимагає більш глибоких знань та вмінь розробника [2]. Окрім того, на цей момент існує дуже обмежена кількість пакетів даних, які надають вже готовий функціонал для імплементації курсорної пагінації, і жоден з них не є достатньо універсальним: підтримується тільки одна ORM (Object-Relational Mapping – об'єктно-реляційне відображення) система, або одна нативна база даних. До того ж подібні рішення не мають в собі імплементацію індексації запитів для підвищення ефективності; відсутня опціональна фільтрація/сортування даних за обраним параметром.

Таким чином, проблема у відсутності універсального та оптимізованого методу пагінації є актуальною та потребує дослідження. Тому у даній роботі пропонується ро-

зробити модифікований метод курсорної пагінації та фільтрації даних у вигляді пакету NPM, який вирішить недоліки наявних методів.

Аналіз останніх досліджень і публікацій. Існує два методи використання курсорної пагінації в проєктах будь-якого рівня складності: написання її вручну розробником та налаштування всіх необхідних функціональних можливостей опісля; або ж використання вже існуючого рішення, в основному, у вигляді пакету NPM (Node Package Manager), оскільки NPM є найпопулярнішим пакетним менеджером платформи Node.js, яка, в свою чергу, є одним з лідерів серед платформ для бекенд частин вебзастосунків [3].

Оскільки реалізація курсорної пагінації вручну може зайняти чимало часу навіть у досвідчених розробників, подібний підхід використовується доволі рідко. Використання ж існуючих готових рішень підходить далеко не для всіх проєктів через низьку універсальність наявних методів, практично повну відсутність опцій для фільтрації/сортування даних за заданим параметром та не найкращу швидкодію запитів (в наслідок відсутності індексації запитів).

Розглянемо детальніше найбільш популярні пакети NPM, що містять в собі функціональні можливості курсорної пагінації даних.

Mongodb-cursor-pagination – як можна здогадатись з назви, головною та єдиною задачею цього пакету є робота з базою даних MongoDB, використовуючи нативний драйвер. Основними перевагами є простота у використанні та детальний опис функціональних можливостей. Серед недоліків спостерігається відсутність підтримки інших баз даних, повільна робота над запитами з комплексним унікальним ключем та відсутність можливості обрання поля для сортування [4].

Mongoose-relay-paginate – пагінація з використанням ORM системи Mongoose, та, відповідно, бази даних MongoDB. Переваги: зрозуміла документація, швидкі запити до бази даних незалежно від складності даних. Недоліки: аналогічно до вище розглянутого рішення відсутність підтримки інших баз даних та ORM, комплексних фільтрації та сортування тощо.

Typeorm-cursor-pagination – пагінація з використанням ORM системи TypeORM з підтримкою більшості реляційних (SQL) баз даних. Переваги: чудова документація, підтримка чималої кількості баз даних, наявна можливість базового налаштування сортування. Недоліки: відсутність підтримки MongoDB (хоч сама ORM і працює з цією базою, проте в цьому пакеті вона не доступна з певних причин), налаштування фільтрації [5].

Sequelizejs-cursor-pagination – пагінація з використанням ORM системи Sequelize з підтримкою деяких реляційних баз даних (таких як PostgreSQL, MySQL, SQLite). Переваги: простота в реалізації, можливе базове налаштування фільтрації та сортування даних. Недоліки: підтримка обмеженої кількості баз даних, відсутність комплексних налаштувань фільтрації та сортування даних, посередня документація [6].

Мета дослідження. Метою даного дослідження є підвищення ефективності та універсальності курсорної пагінації шляхом розробки та програмної реалізації методу,

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

у якого час запитів буде суттєво скорочений, та кількість баз даних, що підтримуються, буде більшою, аніж в існуючих рішеннях. Відповідно до вказаної мети необхідно розв'язати такі задачі: оглянути наявні методи реалізації курсорної пагінації, запропонувати новий метод з кращою швидкодією та більшою універсальністю, експериментально перевірити ефективність розробленого методу.

Викладення основного матеріалу дослідження. Однією з перших речей, які варто обдумати, є вибір формату реалізації запропонованого методу. Оскільки JavaScript є однією з найпопулярніших мов програмування для реалізації вебзастосунків, Node.js – це платформа для імплементації бекенд частини додатків, а NPM – найпопулярніший пакетний менеджер Node.js, то було обрано саме реалізацію модифікованого методу курсорної пагінації та фільтрації даних у вигляді пакету NPM [7].

Пакет NPM побудований з використанням фреймворку Nest.js. Його було обрано, оскільки він використовує статично типізовану мову TypeScript, має чітку модульну структуру, надає можливість інкапсуляції методів, та, найголовніше, має підтримку усіх необхідних баз даних та ORM систем.

Основними функціональними можливостями запропонованого методу є:

- підтримка багатьох баз даних (PostgreSQL, MySQL, SQLite, MongoDB тощо) та ORM (TypeORM, Mongoose, Sequelize тощо);
- власне курсорна пагінація у вигляді методу класу, що імпортується з NPM пакету;
- опціональна комплексна фільтрація даних;
- опціональне комплексне сортування даних;
- опціональна індексація запитів у вигляді методу класу, що імпортується з NPM пакету.

Розглянемо детальніше кожен з наведених пунктів.

Задля забезпечення роботи більшості баз даних та ORM систем було обрано класову систему, де кожен метод відповідає за імплементацію курсорної пагінації в певній базі даних чи ORM. Такий підхід дозволяє виконати імпорт лише одного сервісу, та надає доступ до всіх його функціональних можливостей. Для нативного підключення до баз даних було використано існуючі пакети даних, що надають можливість використання необхідних драйверів баз даних та їх методів. Для використання ORM систем, аналогічно до нативного підключення баз даних, було використано пакети даних NPM, які містять в собі потрібні сервіси та функції роботи з системою. Оскільки в більшості випадків користувач запропонованого модифікованого методу курсорної пагінації у вигляді пакету NPM не має необхідності у використанні всіх наявних баз даних та ORM, було використано динамічне підвантаження пакетів – тобто, наприклад, якщо користувач (програміст) використовує для курсорної пагінації лише базу даних PostgreSQL та ORM систему TypeORM, лише відповідні пакети даних будуть додані в структуру NPM пакету запропонованого рішення.

Процес курсорної пагінації в запропонованому методі є більш комплексним, ніж традиційне пагінування, оскільки включає не лише просте розділення великого набору даних за контрольними точками (так званими курсорами), а й додаткову логіку філь-

трації та сортування. Запропоноване рішення має алгоритм, який складається з таких частин:

- валідація даних, що надходять до методу;
- визначення ключа сортування даних;
- створення об'єкта фільтрації даних;
- декодування отриманого курсора;
- створення запиту до бази даних, що містить в себе сортування та фільтрацію;
- виконання запиту та отримання відповіді, обробка даних;
- сформування об'єкта відповіді.

Розглянемо детальніше кожну частину.

Усі методи, незалежно від бази даних чи ORM, приймають майже однаковий набір параметрів. Обов’язковими є: limit – кількість даних, яку необхідно повернути; назва первинного ключа сутності/документа (в разі відсутності ключа сортування, сортування відбувається саме за первинним ключем); назва сутності/таблиці та відповідний метод драйвера бази даних/репозиторій ORM системи. Опціональними є: курсор (точка відліку, що розділяє дані), може бути відсутнім у разі завантаження початкового набору даних; тип курсора (next або previous), що вказує на напрям (отримання набору даних після курсора чи до), у разі відсутності використовується next; ключ сортування – поле сутності/документа (сутності для нереляційних баз даних, документа для реляційних); об’єкт фільтрації даних (певний набір параметрів). Перед формуванням запиту виконується обробка та валідація вхідних параметрів. Зокрема, перевіряється, що ліміт є натуральним числом, а тип курсора вказано як *next* чи *previous*, без інших значень.

Після валідації даних визначається ключ сортування даних. Для початку перевіряється наявність вказаного ключа сортування. В разі його відсутності сортування відбувається за первинним ключем, назва якого передається в параметрах метода. Якщо ж ключ сортування присутній, сортування даних відбувається за ним. Окрім цього, ключ може бути комплексним (приймати декілька значень водночас), тоді сортування відбувається водночас за всіма переданими полями для збільшення унікальності. Важливо зазначити, що сортування має виконуватись за унікальними значеннями, інакше може виникнути проблема з повторенням записів або некоректною кількістю даних, що повертаються методом.

Далі створюється об’єкт фільтрації даних. У разі передачі певних фільтрів у параметрах методу, об’єкт складатиметься з двох умов: отримання даних після або до заданого курсора (залежно від типу та наявності курсора) та власне переданих в методі даних. В іншому випадку фільтр буде здійснювати пошук лише за курсором. Звісно, якщо необхідно отримати перший набір даних (курсор не заданий), та без додаткових параметрів фільтрації, кінцевий об’єкт фільтра буде переданий в запит до бази даних порожнім. Об’єкт даних, який приймає метод, є статично типізованим та спеціально розроблений для підтримки різних баз даних. Він може як комбінувати декілька умов

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

водночас, так і бути максимально простим, наприклад, зі звичайним повнотекстовим пошуком [8].

Останнім кроком перед формуванням остаточного запиту до бази даних є декодування курсора. Для убезпечення даних при створенні курсора (останній крок методу) використовується кодування за допомогою алгоритму base64. Відповідно, для коректного отримання даних з бази перед формуванням запиту токен декодується за допомогою моделі того ж base64.

Після всіх підготовчих процесів, описаних вище, відбувається створення запиту до бази даних, який містить в собі опції сортування та фільтрації. Запит доволі суттєво відрізняється в залежності від бази даних та ORM системи, тому кожен метод враховує ці специфіки.

Далі відбувається виконання запиту та отримання відповіді від бази даних. Перед надсиланням запиту також відбувається додаткова перевірка на відсутність можливості використання SQL-ін'єкції, для забезпечення недоторканості даних, що зберігаються в базі. Після отримання відповіді відбувається обробка даних, які надходять. Вона є унікальною для кожної бази даних та ORM системи, через різний формат відповіді, тому за допомогою спеціальних функцій отримані дані перетворюються в уніфікований формат, який буде наданий у відповіді. Окрім цього, на цьому кроці відбувається кодування курсорів (про що було описано вище).

Власне, останній крок, в якому отримується відформатований об'єкт та надсилається як відповідь методу. Він має такі параметри як *data* (набір отриманих даних), *cursorNext* (закодований курсор останнього елементу з набору даних), *cursorPrevious* (закодований курсор першого елементу з набору даних), *valuesNumber* (кількість даних, що було отримано).

Розглядаючи детальніше комплексну фільтрацію, можна побудувати таку математичну модель:

$$filter = \lambda f.T(f) \to Q \# \quad (1)$$

де $filter$ виступає в ролі функції, яка фактично є рекурсивним компілятором предикатів, що рекурсивно проходить логічне дерево фільтрів f і застосовує правило відображення для перетворення кожного вузла на необхідну умову. У свою чергу, якщо f є композитним деревом (тобто, на практиці умова фільтру містить водночас декілька операцій, таких як AND, OR, NOT тощо), то функція трансформації даних має такий вигляд:

$$T(f) = T(f_1) \otimes T(f_2) \otimes \dots \otimes T(f_n) \# \quad (2)$$

де $\otimes$ виступає в ролі логічного оператора (AND/OR/NOT і т.п.).

У свою чергу, комплексне сортування даних має такий вигляд. Нехай:

– $D = \{d_1, d_2, \dots, d_n\}$ – множина елементів (наприклад, записів в таблиці чи документі).

– $K = [k_1,\ k_2, \dots, k_m]$ – впорядкований набір ключів сортування (наприклад, поля *id* та *createdAt*).
– $dir_i \in \{\uparrow,\ \downarrow\}$ – напрямок сортування для кожного ключа.
– $c = (v_1,\ v_2, \dots, v_m)$ – курсор.

Тоді для всіх елементів $d \in D$ виконується:

$$\exists t \in [1, m] \text{ таке, що } \left[\begin{array}{c} \forall s < t,\ \ k_s(d) = v_s \\ \text{та } \begin{cases} k_t(d) > v_t, \text{якщо } dir_t = \uparrow \\ k_t(d) < v_t, \text{якщо } dir_t = \downarrow \end{cases} \end{array}\right] \# \quad (3)$$

Наостанок розглянемо індексацію запитів. Загалом, індекс – це окрема структура, яка зберігає копію певних колонок таблиці разом із посиланнями на відповідні записи (рядки). Індекс функціонує подібно до алфавітного покажчика в книзі: він дозволяє швидко знайти потрібні дані без необхідності переглядати весь зміст таблиці. У наведеному методі використовується декілька різних видів індексації: B-Tree, Hash та Compound. B-Tree використовується для індексації запитів за параметрами, які можуть бути різних типів. У своїй основі B-Tree індекс використовує алгоритм пошуку в збалансованому та відсортованому бінарному дереві, де висота кожної гілки залишається майже однаковою. Часова складність запитів, що використовують цей індекс, складає $O(log\ n)$ – себто є логарифмічною. Така швидкодія досягається завдяки тому, що B-Tree зберігає великий обсяг ключів у кожному вузлі. Це знижує кількість рівнів, які потрібно пройти під час пошуку. Hash індекс використовується для пошуку за PK (Primary Key – первинний ключ) та має постійну складність, а саме $O(1)$. Compound індекс використовується для пошуку та фільтрації за кількома параметрами одночасно та має таку ж складність, як і B-Tree, $O(log\ n)$, оскільки також використовує впорядковані дерева [3].

Результати експериментальних досліджень. Дослідження відбувались на спеціально підготовленому проєкті, в якому використовувались всі бази даних та ORM системи, які підтримуються NPM пакетом, що містить в собі запропонований модифікований метод курсорної пагінації даних. Курсорна пагінація пройшла успішно в 100% випадків, коли в методи були передані валідні дані; обробка помилок відпрацювала у випадку некоректних даних. Сортування та фільтрація виконали свою роль коректно в 94% випадків (є доволі складні та малореалістичні кейси, які наразі не покриті логікою). Тестування індексації запитів повернуло такі результати для бази PostgreSQL (рис. 1) та MongoDB (рис. 2).

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

DATABASE	ORM	INDEX	SORT	FILTER	AVG LATENCY (MS)
PostgreSQL	TypeORM	+	-	-	8.4
PostgreSQL	TypeORM	+	+	+	9.7
PostgreSQL	TypeORM	-	-	-	60.7
PostgreSQL	TypeORM	-	+	+	240.8
PostgreSQL	Sequelize	+	-	-	10.8
PostgreSQL	Sequelize	+	+	+	14.1
PostgreSQL	Sequelize	-	-	-	233.2
PostgreSQL	Sequelize	-	+	+	196.9
PostgreSQL	-	+	-	-	6.4
PostgreSQL	-	+	+	+	7.9
PostgreSQL	-	-	-	-	43.2
PostgreSQL	-	-	+	+	190.3

Рисунок 1 – Результати тестування курсорної пагінації в базі даних PostgreSQL

DATABASE	ORM	INDEX	SORT	FILTER	AVG LATENCY (MS)
MongoDB	TypeORM	+	-	-	9.7
MongoDB	TypeORM	+	+	+	11.3
MongoDB	TypeORM	-	-	-	208.6
MongoDB	TypeORM	-	+	+	164.1
MongoDB	Mongoose	+	-	-	10.1
MongoDB	Mongoose	+	+	+	13.0
MongoDB	Mongoose	-	-	-	84.1
MongoDB	Mongoose	-	+	+	398.5
MongoDB	-	+	-	-	5.3
MongoDB	-	+	+	+	6.4
MongoDB	-	-	-	-	60.1
MongoDB	-	-	+	+	242.3

Рисунок 2 – Результати тестування курсорної пагінації в базі даних PostgreSQL

Інші реляційні бази даних, такі як, наприклад, MySQL та SQLite, мають схожі показники до PostgreSQL; нереляційні – подібні до MongoDB.

Висновки. Дослідження присвячене розробці модифікованого методу курсорної пагінації та фільтрації даних у вигляді пакету NPM. Запропонований метод дозволяє підвищити швидкодію запитів до бази даних завдяки використанню індексів. Він є універсальним, оскільки підтримує основні бази даних та ORM, надає можливість фільтрувати та сортувати дані за заданими параметрами тощо.

Результати тестування показують успішність імплементації запропонованого методу для більшості випадків використання курсорної пагінації.

Подальше дослідження полягатиме в можливій оптимізації даних за допомогою кеш-таблиць (які підтримуються певними базами даних), розв’язання питань зі складними випадками комплексних фільтрації та сортування даних, які наразі не підтримуються, покриття коду тестами, та підтримка більшої кількості баз даних та ORM систем.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

ЛІТЕРАТУРА

1.Jason Ngan (2022). Розуміння курсорної та оффсетної пагінацій [Електронний ресурс]. – Режим доступу: https://medium.com/better-programming/understanding-the-offset-and-cursor-pagination-8ddc54d10d98

2.Merge.dev (2023). Курсорна пагінація: як працює, переваги та недоліки [Електронний ресурс]. – Режим доступу: https://www.merge.dev/blog/cursor-pagination

3.Kellton (2024). Найкращі фреймворки для бекенд розробки [Електронний ресурс] / Kellton. – Режим доступу: https://www.kellton.com/kellton-tech-blog/best-backend-web-development-frameworks

4.mongo-cursor-pagination (2025). Модуль для реалізації курсорної пагінації в MongoDB [Електронний ресурс]. – Режим доступу:
https://www.npmjs.com/package/mongo-cursor-pagination

5.typeorm-cursor-pagination (2023). Пакет для курсорної пагінації з використанням TypeORM [Електронний ресурс]. – Режим доступу:
https://www.npmjs.com/package/typeorm-cursor-pagination

6.sequelize-cursor-pagination (2025). Пакет для курсорної пагінації в Sequelize [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/sequelize-cursor-pagination

7.about-npm (2023). Про NPM пакети [Електронний ресурс]. – Режим доступу: https://docs.npmjs.com/about-npm

8.Goodays API Platform (2023). Пагінація, сортування та фільтрація [Електронний ресурс]. – Режим доступу: https://apidocs.goodays.co/docs/sorting-and-pagination

REFERENCES

1. Jason Ngan (2022). Understanding the Offset and Cursor Pagination [Electronic resource]. – URL: https://medium.com/better-programming/understanding-the-offset-and-cursor-pagination-8ddc54d10d98

2. Merge.dev (2023). Cursor pagination: how it works and its pros and cons [Electronic resource]. – URL: https://www.merge.dev/blog/cursor-pagination

3. Kellton (2024). Best Backend Web Development Frameworks [Electronic resource] / Kellton. – URL: https://www.kellton.com/kellton-tech-blog/best-backend-web-development-frameworks

4. mongo-cursor-pagination (2025). Module for cursor pagination integration in MongoDB [Electronic resource]. – URL: https://www.npmjs.com/package/mongo-cursor-pagination

5. typeorm-cursor-pagination (2023). Package for cursor pagination with TypeORM [Electronic resource]. – URL: https://www.npmjs.com/package/typeorm-cursor-pagination

6. sequelize-cursor-pagination (2025). Package for cursor pagination with Sequelize [Electronic resource]. – URL: https://www.npmjs.com/package/sequelize-cursor-pagination

7. about-npm (2023). About NPM packages [Electronic resource]. – URL: https://docs.npmjs.com/about-npm

8. Goodays API Platform (2023). Pagination, sorting and filtering [Electronic resource]. – URL: https://apidocs.goodays.co/docs/sorting-and-paginatio

Received 15.04.2025.
Accepted 18.04.2025.

Modified method of cursor pagination and data filtering as an NPM package

This article is devoted to the development of a modified cursor pagination method that supports major databases and ORM systems, provides the ability to use complex data filtering and sorting, query indexing, etc. Existing solutions, their use cases, advantages and disadvantages are analyzed. A modified method is formed that includes modified and improved functionality of existing NPM packages, along with new implementations focused on increasing the efficiency and versatility of database queries. The proposed solution is analyzed. Potential directions for further expansion and improvement of the method are given.

The purpose of this research is to increase the efficiency and versatility of cursor pagination by developing and software implementing a method in which query time will be significantly reduced and the number of databases supported will be greater than in existing solutions. In accordance with the stated goal, it is necessary to solve the following tasks: to review existing methods for implementing cursor pagination, to propose a new method with better speed and greater versatility, to experimentally verify the effectiveness of the developed method.

At the moment, there is a very limited number of data packages that provide ready-made functionality for implementing cursor pagination, and none of them is universal enough (only one ORM system or one native database is supported). In addition, such solutions do not include the implementation of query indexing to increase efficiency; there is no optional filtering/sorting of data by the selected parameter. Thus, the problem of the lack of a universal and optimized pagination method is relevant and requires research. Therefore, a modified method of cursor pagination and data filtering is proposed in the form of an NPM package, which will solve the shortcomings of existing methods.

Keywords: cursor pagination, database, ORM system, data filtering, query indexing, NPM package, fuzzy search, merge and heap sort, binary tree, hash function.

Степаненко Андрій Сергійович – магістрант кафедри програмного забезпечення комп'ютерних систем, факультет прикладної математики, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

Саяпіна Інна Олександрівна – к.т.н, доцент, доцент кафедри програмного забезпечення комп'ютерних систем, факультет прикладної математики, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

Stepanenko Andrii – student of the Department of Computer Systems Software, Faculty of Applied Mathematics, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

Saiapina Inna – Ph.D., Associate Professor, Associate Professor at the Department of Computer Systems Software, Faculty of Applied Mathematics, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)