DOI 10.34185/1562-9945-5-124-2019-04
УДК 004.75

О.С. Волковський, В.С. Павелко

ОРГАНІЗАЦІЯ БЕЗСЕРВЕРНИХ ОБЧИСЛЕНЬ ДЛЯ РІЗНОПЛАТФОРМНИХ КЛІЄНТІВ

Анотація. Зростання інтересу до безсерверних систем обчислень призвело до стрімкого розвитку можливостей, які надають хмарні провайдери. Але специфічність цієї сфери розробки програмного забезпечення поки що створює деякі проблеми, серед яких:

- *слідування вимогам архітектури серверного додатку не дозволяє розширювати sdk таким чином, щоб новий функціонал працював автоматично;*
- *відсутність API для ефективної синхронізації пристроїв;*
- *відсутність глобального тригеру для збереження історії змін об'єктів системи.*

Метою роботи є вирішення проблем з можливістю розширення існуючого sdk BaaS провайдеру Parse Server автоматичного запуску нового функціоналу.

В ході роботи було використано системний та аналітичний методи дослідження.

У роботі запропоновані алгоритми реалізації синхронізації даних різноплатформних пристроїв та принципи збереження знімків (коммітів) змін об'єктів в системах, у яких в ролі серверу використовується безсерверна система обчислення, реалізована на базі моделі BaaS. Для вирішення задачі були використані принципи прототипного об'єктно-орієнтованого принципу програмування, а також патерни проектування: декоратор, стратегія, event-sourcing, builder, factory. За BaaS провайдер було обрано Parse Server. Розроблені та програмно реалізовані наступні алгоритми: ChangeLogSpy – збереження знімків об'єктів системи без додавання додаткової логіки для класу об'єкта; SyncProvider - реалізація ефективної синхронізації сесій різноплатформних клієнтів; GetAllResultsForQuery - реалізація алгоритму асинхронного отримання всіх результатів запиту.

Ключові слова - безсерверні системи обчислень, синхронізація сесій різноплатформних пристроїв, знімки стану об'єкта.

Постановка проблеми. В ході розвитку безсерверних технологій було отримано достатньо досягнень, які привели до зменшення кінцевої вартості продукту за рахунок використання BaaS провайдерів.

Синхронізація сесій гарантує забезпечення постійного надання актуальних даних незалежно від платформи пристрою, з якого була

зареєстрована нова сесія. Одним з важливих аспектів такого процесу є селекція даних, оскільки важливо відсіяти не актуальні по змісту дані для цього пристрою (на мобільні пристрої не передавати повні версії зображень, та нехтувати даними об'єктів, що не використовується на поточному типі клієнта.

Збереження знімків стану об'єктів - один з різновидів event sourcing, при якому зберігаються не події, що призвели до зміни стану об'єкта, а конкретні стани об'єктів. Такий підхід дозволяє використовувати переключення між станами об'єктів без зміни актуального стану або регулювати бізнес-логіку по версіям продукту або користувацьким планам.

У роботі пропонується реалізація необхідних систем відповідно до тенденції поточної екосистеми, при якій використання описаних функцій стає прихованим для розробників частин продукту.

Було визначено доцільним реалізувати розширення поточного функціоналу sdk хмарного провайдера для підвищення ефективності синхронізації сесій, незалежно від платформи клієнта.

Аналіз останніх досліджень. Існує достатньо зарубіжних матеріалів стосовно принципів реалізації складних систем на основі BaaS провайдерів, але лише в деяких з них розглядалося питання ефективної синхронізації. Особливо можна відокремити документ “Sync Efficient” Р. Лина. В ньому було розглянуто основну ідею реалізації ефективної синхронізації сесій різноплатформних клієнтів додатку “Athlas Guidses” [1].

У вітчизняній літературі не було знайдено джерел стосовно BaaS провайдерів та вирішення проблем підвищення ефективності обробки даних.

Формулювання цілей статті. Виходячи з вищенаданої інформації були сформульовані наступні завдання:

• розробити прихований від користувачів sdk алгоритм, який виконує зберігання знімків об'єктів автоматично, під час зміни будь-якого з них;

• розробити алгоритм ефективної синхронізації даних, враховуючи поточну сесію та платформу пристрою, що здійснив підключення;

• розробити алгоритм для отримання всіх результатів запиту, оскільки стандартний засіб sdk дозволяє отримати максимум 1000 записів, які задовольняють умовам запиту.

Основна частина. Baas провайдер Parse Server надає sdk для розробників серверу додатку, за допомогою якого можна реалізувати бізнес-логіку на стороні серверу на мові програмування JavaScript. Отже, є можливість розширення функціоналу sdk таким чином, щоб уникнути явних викликів методів sdk.

Основна ідея розширення полягає в підстановці замість базових об'єктів sdk - розширених. Провайдер Parse Server імпортує глобальний об'єкт Parse в кожен файл додатку як посилання на всі доступні методи sdk (як об'єкти window, document). Якщо в базовий файл додатку (файл main) в першу чергу імпортувати файли з розширенням цих об'єктів та передекларувати посилання, ми можемо отримати в усіх наступних файлах новий функціонал.

Розглянемо базові алгоритми.

GetAllResultsForQuery - розширення методу для отримання всіх рядків, що задовольняють умовам запиту. Йог необхідно реалізовувати без приховування, оскільки отримання всіх результатів потрібно не в усіх випадках.

Основний порядок дій для цього запиту (Рис 1.):

1. Отримати, як аргумент запит (екземпляр класу Parse.Query).
2. Підрахувати кількість елементів, що задовольняють запиту.
3. Розділити на сторінки по 1000 штук вибірку всіх результатів.
4. Асинхронно отримати результати для всіх сторінок.
5. Змонтувати елементи всіх сторінок в один массив.

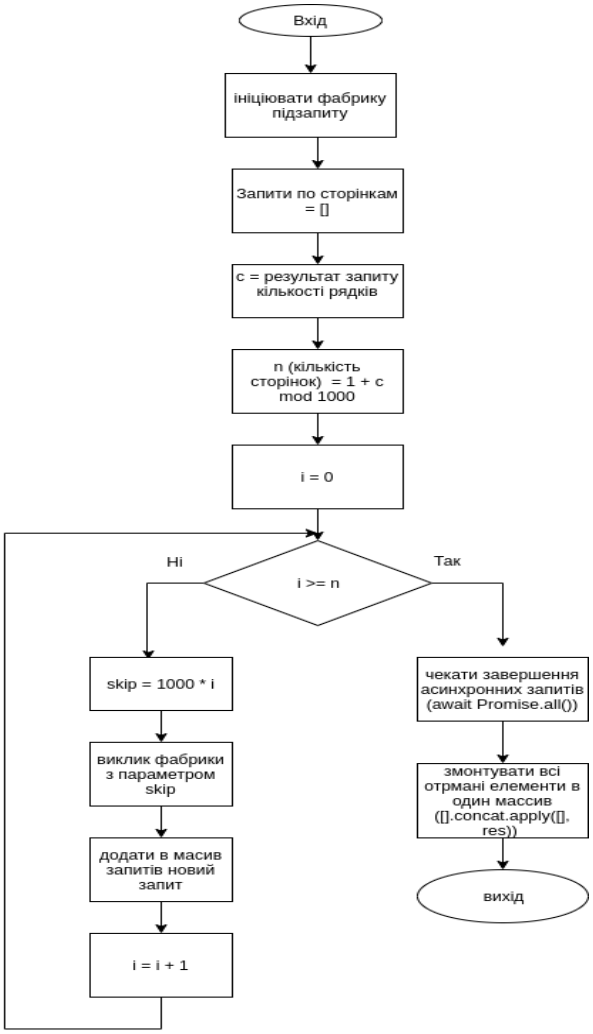

Рисунок 1 - Блок-схема алгоритму GetAllResultsForQuery

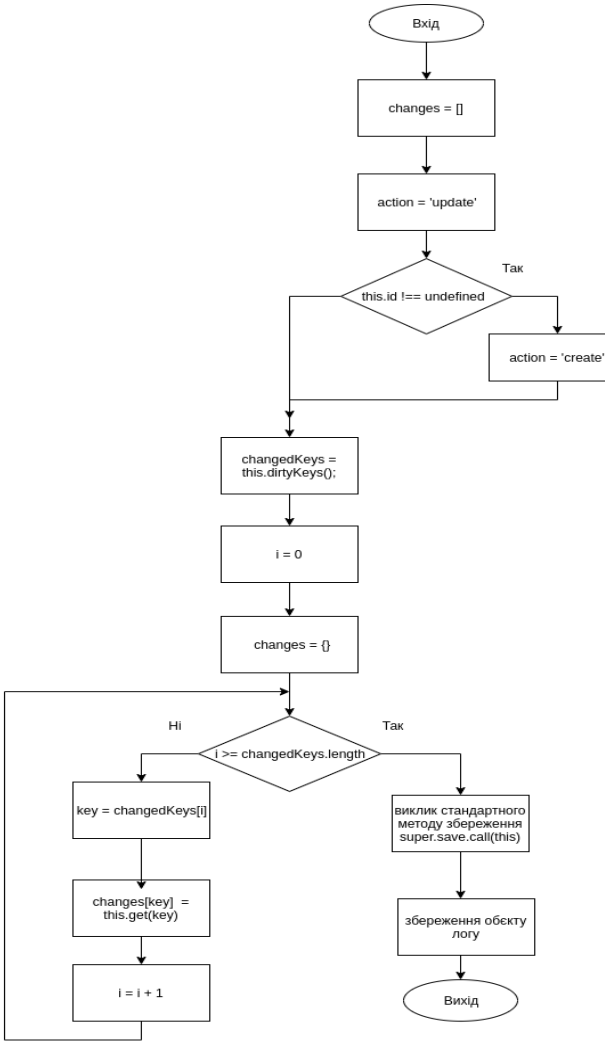

Рисунок 2 - Блок-схема алгоритму ChangeLogSpy

ChangeLogSpy – алгоритм збереження історії змін об'єктів для ведення логу змін та можливості відновлення попередніх станів об'єктів.

Для роботи з об'єктами провайдер Parse Server надає глобальні об'єкти Parse.Object - для створення нових класів системи, та Parse.Query - для роботи з об'єктами, що збережені до бази даних. При цьому, Parse.Query повертає з бази екземпляри класу Parse.Object. Кожен экземпляр класу Parse.Object має методи save та destroy - збереження та видалення відповідно.

Також, в глобальному об'єкті Parse доступні методи saveAll та destroyAll - для массового збереження/видалення об'єктів.

Виходячи з вказаного, було вирішено розширити базові класи Parse.Object та Parse.Query наступним чином:

1. При викликах методів save/saveAll зберігається інформація в лог про створення або редагування об'єкта.

2. При викликах методів destroy/destroyAll виконується зберігання логу про видалення об'єктів.

3. Методи класу Parse.Query повертають розширені об'єкти нового класу, наслідуваного від Parse.Object.

Оскільки мова JavaScript реалізує ідею прототипного наслідування, для об’єктів, що були отримані з методів Parse.Query, достатньо прилаштувати, як прототип, розширений клас.

Для аналізу та збереження логу змін було реалізовано алгоритм ChangeLogSpy (Рис. 2).

Під час виклику, згідно з реалізованим алгоритмом, відбувається перевірка, чи існує в поточному єкземплярі id: якщо id існує, це означає, що об'єкт оновлюється, якщо ні - то об'єкт, тільки що створено.

Щоб встановити, які поля об'єкту були оновлені, можна використати базовий метод Parse.Object.prototype.dirtyKeys. Даний метод повертає масив рядків з іменами полів, які були змінені. Така поведінка доступна, оскільки розробниками було використано патерн Proxy, який слідкує за зверненням до полів об'єкта.

SyncProvider - алгоритм ефективної синхронізації (Рис. 3) великих об'єктів, що враховує поточну сесію пристрою, який її ініціював.

Для кожного об'єкта в системі створюється дата останнього редагування. Спираючись на цю дату, можна зробити припущення, про актуальність версії об'єкта на клієнті. Для встановлення актуальності версії з клієнта достатньо відправити дані з id об'єкта та датою його оновлення.

Для кожного об'єкта необхідно створити конфігурацію, на основі якої можна запустити фабрику створення запиту. Основна ідея полягає в створенні оточення та функціоналу, на основі якого можна лише конфігурувати умови для створення необхідних об'єктів.

Оскільки sdk пропонує використовувати патерн builder для створення запиту, можна реалізувати набір стратегій та фабрик, що створюють необхідні об'єкти запитів. Враховуючи реалізацію обробників запитів в екосистемі Parse Server та здійснивши реалізацію декораторів для різнотипних об'єктів, можна обгорнути канали передачі об'єктів, які враховують актуальний стан даних на клієнті та повертають тільки ті дані, які необхідно оновити.

Оскільки, функція обробки запиту є простим экземпляром middlewares,існує можливість декорувати таку функцію іншою функцією, для слідування принципа модульності. Тобто, реалізувавши необхідні набори декораторів, можна реалізувати гнучкий інтерфейс, елементи якого мінімально залежні один від одного. Також, така функція буде доступна всім користувачам sdk, а отже немає ніякої проблеми для клієнтів з різною платформою.

Рисунок 3 - Блок-схема алгоритму ефективної синхронізації

Висновки. Використання BaaS є перспективним, оскільки дозволяє створювати гнучкі рішення за короткий час. Розлянуте sdk BaaS провайдера Parse Server дозволяє розширювати існуючий функціонал простим та єффективним шляхом. В ході роботи були створені алгоритми синхронізації даних для різноплатформних клієнтів, алгоритм отримання всіх результатів для запиту, приховане ведення логу змін об'єктів системи.

Результати досліджень можуть бути застосовані при створенні безсерверних обчислювальних систем для додатків, які портуються на різні платформи.

ЛИТЕРАТУРА / ЛІТЕРАТУРА

1. Ryan Linn, Athlas Guides: Sync Efficient [Електронний ресурс]. Режим доступу: https://atlasguides.com/parse-sync-efficient/
2. Parse Cloud Code Simple [Електронний ресурс]. Режим доступу: https://www.back4app.com/
3. Parse Cloud Code Hell [Електронний ресурс]. Режим доступу: https://www.eidel.io/2015/11/24/parse-com-cloud-code-hell/
4. Parse Platform. Docs - [Електронний ресурс]. Режим доступу: https://docs.parseplatform.org/cloudcode/guide/

REFERENCES

1. Ryan Linn, Athlas Guides: Sync Efficient [Електронний ресурс]. Режим доступу: https://atlasguides.com/parse-sync-efficient/
2. Parse Cloud Code Simple [Електронний ресурс]. Режим доступу: https://www.back4app.com/
3. Parse Cloud Code Hell [Електронний ресурс]. Режим доступу: https://www.eidel.io/2015/11/24/parse-com-cloud-code-hell/
4. Parse Platform. Docs - [Електронний ресурс]. Режим доступу: https://docs.parseplatform.org/cloudcode/guide/

Received 11.10.2019.
Accepted 16.10.2019.

Организация безсерверных вычислений для разноплатформенных клиентов

В работе предложены алгоритмы синхронизации данных разноплатформенных устройств и принципы сохранения снимков (коммитов) изменений состояний объектов в системах, у которых в роле сервера выступает безсерверная система вычисления, реализованная на базе модели BaaS. Для решения задачи были использованы принципы прототипного объектно-ориентированного принципа программирования, а также паттерны

проектирования: декоратор, стратегия, event-sourcing, builder, factory. В качестве BaaS провайдера был выбран Parse Server. Разработаны и программно реализованы следующие алгоритмы: ChangeLogSpy – сохранение снимком объектов системы без добавления дополнительной логики для класса объекта; SyncProvider – реализация эффективной синхронизации сессий разноплатформных клиентов, GetAllResultsForQuery - реализация алгоритма асинхронного получения всех результатов запроса.

Organization of serverless computing for multi-platworm clients

Growing of interest to serverless have leaded to extending ability of serverless. But development of this software has many problems such as:

- there are not ability to autorun extended functional because application should be implemented according to serverless architecture;

- serverless doesn't have API for efficient sync;

- serverless doesn't have global trigger for saving history of object changes.

The main goal of this article is resolving of Parse Server issue with autorun of new functional.

The systemic and analytic methods were used during researching.

This article proposes the sync efficient algorithms for multi-platform devices and the principles of saving changes of object states related to systems which was implemented as serverless solution via BaaS model. Patterns such as decorator, strategy, event-sourcing, builder, factory were used for resolving this task. Also, prototype-oriented principle was used. Parse Server was used as BaaS provider. These algorithms were implemented: ChangeLogSpy – for saving objects snapshots without adding additional logic for the object class; SyncProvider – implementation of efficient session synchronization for multiplatform clients; GetAllResultsForQuery - implementation of algorithm for async fetching all query results.

Павелко Виталий Сергеевич – магистр Днепровского Национального Университета имени Олеся Гончара.

Волковский Олег Степанович - кандидат технических наук, доцент кафедры компьютерных наук и информационных технологий Национального Университета имени Олеся Гончара.

Павелко Віталій Сергійович - магістр Дніпровського Національного Університету імені Олеся Гончара.

Волковський Олег Степанович - кандидат технічних наук, доцент кафедри комп'ютерних наук та інформаційних технологій Дніпровського Національного Університету імені Олеся Гончара.

Pavelko Vitalii - Master of the Dnipro National University named after Oles Honchar.

Volkovskyy Oleg - Candidate of Technical Sciences, Associate Professor of the Department of Computer Science and Information Technology of the Dnipro National University named after Oles Honchar.