УДК 004.4’2,004.75
DOI: 10.18523/2617-3808.2024.7.4-10

Олецький О. В., Моголівський В. О.

КООРДИНАЦІЯ МІКРОСЕРВІСІВ ІЗ ВИКОРИСТАННЯМ МАШИН СТАНІВ

У статті розглянуто підхід до координації мікросервісів на основі використання машин станів як абстракції для управління розподіленим потоком виконання програми з мікросервісною архітектурою. Реалізовано прототип бібліотеки, загальну архітектуру і принципи роботи якого описано в статті.

У разі застосування описаного підходу полегшується та упорядковується робота з паралельними потоками виконання і довготривалими фоновими задачами. Крім того, покращується прозорість процесів, що полегшує їх моніторинг, візуалізацію, тестування і відлагодження. Для використання машин станів у мікросервісному середовищі реалізовано інтеграцію з брокером повідомлень і належне довготривале зберігання їхнього стану.

Ключові слова: мікросервісна архітектура, машини станів, координація мікросервісів, подійно-орієнтований підхід, управління потоком виконання, моніторинг мікросервісів, візуалізація мікросервісів, тестування мікросервісів.

1. Вступ і постановлення задачі

Від моменту своєї появи у 2011 р. [6] мікросервісна архітектура набула значного поширення завдяки масштабованості, гнучкості та надійності. Згідно з опитуванням, проведеним O’Reilly серед своїх підписників у 2020 р., 77 % респондентів зазначили, що їхні компанії використовують мікросервісну архітектуру [7]. Більшість респондентів, а саме 55 %, вважають, що впровадження мікросервісів є доволі успішним, однак зауважили про труднощі, з якими доводиться стикатися [7]. На складність управління багатьма сервісами скаржаться 29 % опитаних, на непрозорість і труднощі у моніторингу — 27 %, на клопоти з тестуванням — 20 %, на складність відлагодження — 16 % [7]. Інші дослідження також виявляють схожі проблеми [5; 9–11].

Отже, актуальною є проблема пом’якшення та/або усунення цих та інших проблем, характерних для мікросервісної архітектури. Якщо зробити певне узагальнення, то слід звернути увагу на те, що ці проблеми значною мірою пов’язані з недостатнім розвитком засобів координації мікросервісів. У цій царині переважають евристичні підходи.

У цій статті розвинуто підхід до координації мікросервісів на основі використання машин станів.

2. Забезпечення належного функціонування машин станів у мікросервісній архітектурі

Машину станів слід розглядати як абстракцію для управління розподіленим потоком виконання програм [13], і для застосунків із мікросервісною архітектурою це набуває особливого значення. При цьому важливими є питання задовільного вибору моделі та реалізації машини станів, забезпечення механізмів комунікації між мікросервісами і машинами станів, а також забезпечення належного рівня персистентності, тобто довготривале і надійне зберігання як станів розподіленої системи, так і власне машини станів.

Як реалізацію машини станів було обрано бібліотеку XState [13]. Ця бібліотека спирається на класичний теоретичний апарат на основі скінченних автоматів Мілі та Мура, але доповнює їх підходами, запропонованими Девідом Харелем, Граді Бучем, а також використовує W3C стандарт SCXML [13]. Ідеться про механізми на основі вкладених і паралельних (складених) станів, внутрішніх і зовнішніх подій, умовних переходів, контекстної залежності (по суті — на основі пам’яті) та навіть побічних ефектів [3; 4; 12]. Маючи таку багату функціональність, машини станів на цій основі можуть бути максимально корисними під час розроблення сучасного програмного забезпечення, зокрема на основі мікросервісів.

Для забезпечення комунікації між машинами станів і мікросервісами було вирішено використати подійно-орієнтований підхід, який можна застосовувати як у машинах станів, так і у мікросервісах [8; 12]. Було реалізовано можливість надсилати події з машини станів до мікросервісів і навпаки. Для цього було використано брокер повідомлень Apache Kafka. Такий вибір обґрунтовано двома аспектами. По-перше, брокер повідомлень — це необхідний інструмент у мікросервісній архітектурі, який забезпечує надійність, відновлюваність і масштабованість [6]. По-друге, було використано таку властивість Apache Kafka, як гарантований порядок повідомлень з однаковим ключем партиції [1]. Для усіх подій, пов'язаних із певною машиною станів, як ключ партиції встановлюється унікальний ідентифікатор машини станів. У такий спосіб певною мірою забезпечується порядок повідомлень, що суттєво спрощує опис усіх потенційних подій для кожного зі станів у машині станів.

Читання повідомлень (подій) може відбуватися в одному з двох режимів, а саме зі збереженням відступу від початку черги для кожної події або для групи подій. Перший режим спрямований на максимальну надійність, адже за критичного збою доведеться повторно обробляти тільки одну подію. Другий режим дає змогу пришвидшити роботу системи, зберігаючи відступ після оброблення певної кількості подій, проте призводить до надмірної складності у разі критичного збою. Саме задля подолання цієї складності режимом за замовчуванням було обрано перший режим — збереження відступу для кожної події. Цей режим дає змогу значно легше зробити машину станів, наскільки це можливо, ідемпотентною, оскільки значно легше забезпечити ідемпотентність для кожної окремо взятої події, ніж для серії подій. Якби брокер повідомлень міг гарантувати семантику доставлення повідомлень «рівно один раз», то не було б потреби забезпечувати ідемпотентність оброблення подій, проте жодний із на сьогодні відомих комерційних брокерів повідомлень не надає такої гарантії у контексті мікросервісної архітектури. А вона необхідна для забезпечення відмовостійкості.

Для довготривалого зберігання стану машини станів було використано багатомодельну СУБД ArangoDB [2]. У ній зберігається метаінформація про машину станів і її серіалізований стан. У принципі можна використовувати будь-яку СУБД, але вибір саме ArangoDB пов'язаний із її перспективністю для подальших досліджень. Ця перспективність полягає у багатомодельності цієї бази даних і наявної підтримки у ній графів [2]. Машини станів легко моделюються графами, що може дати змогу додатково їх аналізувати для надання корисних порад розробникам.

Отже, для використання машин станів у мікросервісній архітектурі було створено бібліотеку з архітектурою, зображеною на рис. 1.

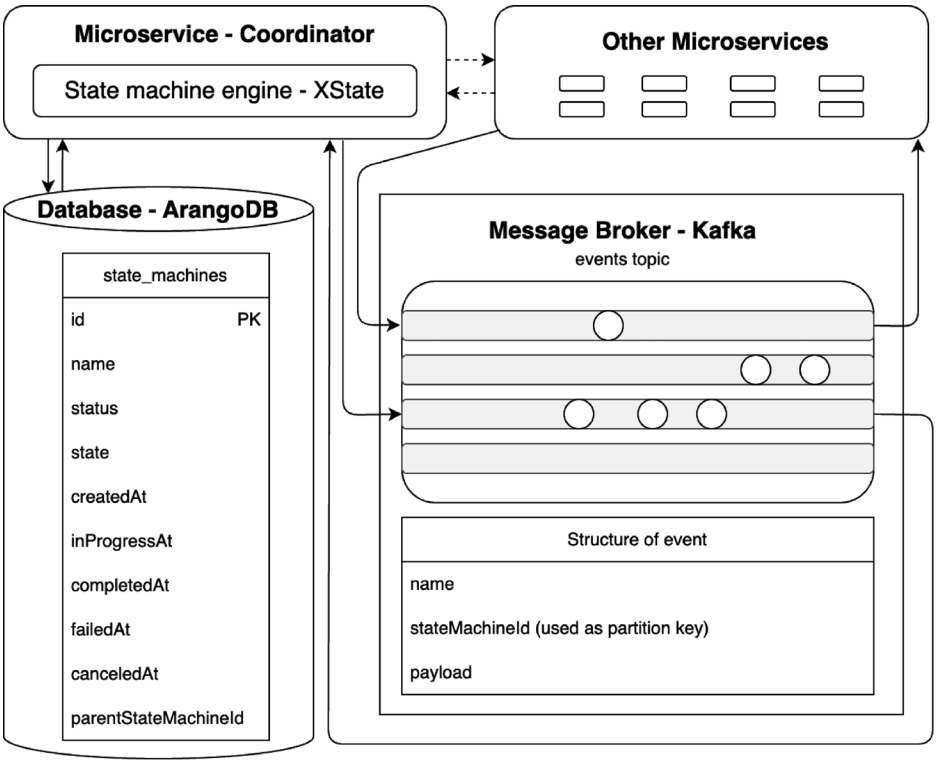

Рис. 1. Реалізація мікросервісної архітектури з використанням машини станів

У базі даних ArangoDB зберігається представлення машини станів, яке складається з унікального ідентифікатора, назви, статусу, стану та ідентифікатора батьківської машини станів (якщо така є). Також зберігається дата і час створення, запуску, помилки, завершення чи скасування виконання машини станів. Поле «назва» по суті зберігає тип машини станів, як-от GDPRCleanUpMachine чи MonthlyReportMachine. З назвою пов'язана визначена розробником машина станів (стани, переходи, дії тощо). Статус зберігає високорівневий стан виконання, він може мати такі значення, як NEW, IN_PROGRESS, COMPLETED, FAILED, CANCELLED. У полі «стан» зберігається серіалізований стан, створений бібліотекою XState. У ньому міститься стан, у якому перебуває машина, поточний контекст (об'єкт, який є пам'яттю машини станів) та історія попередніх станів.

Під час старту застосунку буде ініціалізовано екземпляр машини станів, після чого його буде збережено у базу даних із початковим станом. Далі брокеру повідомлень буде надіслано подію STATE_MACHINE_START, яку пізніше прочитає один із мікросервісів, який буде у ролі координатора; він десеріалізує машину станів із бази даних в оперативну пам'ять і надішле їхню початкову подію. Машина станів почне своє виконання, під час якого вона змінюватиме стани й надсилатиме події для інших мікросервісів до брокера. Вона продовжуватиме своє виконання, допоки не досягне кінцевого стану або не вичерпає усі можливі переходи. Коли це станеться, вона буде серіалізована та збережена у базу даних.

Якщо усі можливі переходи вичерпано, а кінцевого стану не досягнуто — це означає, що машина очікує на події від інших мікросервісів, якщо, звісно, розробник не припустився помилок при оголошенні машини станів. Коли інші мікросервіси оброблять раніше надіслані події з машини станів мікросервісом- координатором, вони надішлють свою відповідь у вигляді подій аналогічних до першої розглянутої події STATE_MACHINE_START, так коло замкнеться: машину станів буде знов десеріалізовано й запущено. Цей цикл відбуватиметься до того моменту, поки машина станів не досягне свого кінцевого стану.

3. Переваги застосування машин станів у мікросервісній архітектурі

Центральною ідеєю мікросервісної архітектури є розподіл логіки застосунку між мікросервісами. Такий підхід підвищує гнучкість, масштабованість і надійність системи, але водночас і ускладнює управління даними та координацію процесів.

Прикладом типової задачі, для якої проявляються зазначені проблеми, є видалення даних для дотримання регламенту GDPR. При проєктуванні мікросервісів основну увагу приділяють забезпеченню високої продуктивності системи, що є звісно цілком виправданим, але це своєю чергою призводить до розпорошення даних між мікросервісами. Через це виникають складнощі при імплементації дотримання GDPR, оскільки дані можуть бути розкидані по багатьох мікросервісах. Інформація про те, які дані мають бути видалені, може міститися на одному наборі мікросервісів, а самі дані на іншому. Тому може постати потреба у забезпеченні розподіленого потоку виконання між мікросервісами, за якого спершу буде зібрано інформацію про дані, які потребують очищення, а уже потім буде видалено дані з усіх мікросервісів, які їх містять. Приклад такого потоку виконання зображено на рис. 2.

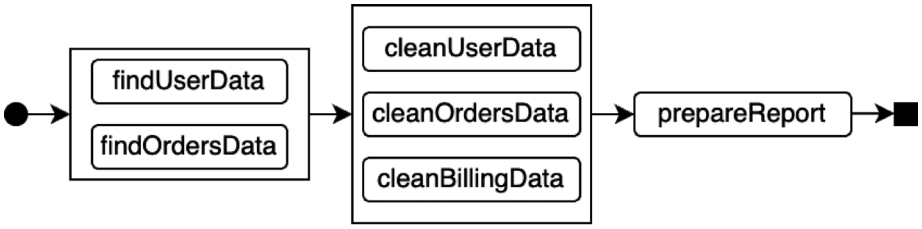

Рис. 2. Приклад потоку виконання мікросервісного застосунку при видаленні даних згідно з GDPR

Спершу необхідно знайти інформацію, яка потребує видалення. Для цього, як правило, потрібно залучити декілька мікросервісів. Далі необхідно дочекатися, поки кожен із залучених мікросервісів виконає пошук. Коли пошук буде завершено, необхідно здійснити видалення даних, у якому теж, як правило, мають бути залучені декілька мікросервісів. Більше того, процес очищування може бути тривалим і відбуватися протягом декількох годин, а то й днів. І тільки коли всі залучені мікросервіси видалять дані, можна перейти до завершальних дій, наприклад формування звіту. За це завдання, як правило, відповідає тільки один із мікросервісів.

Для вирішення таких задач типовим є застосування подійно-орієнтованого підходу, за якого мікросервіси надсилають події, щоб обмінюватися інформацією для координації своєї роботи. За такого підходу перед розробниками постають дві основні проблеми:

1. Координація очікування мікросервісами один одного:
a) один чекає на багатьох;
b) багато чекають на одного;
c) багато чекають на багатьох.
2. Забезпечення контролю та прозорості поточного статусу системи, особливо для довготривалих завдань.

Доповнення подійно-орієнтованого підходу машинами станів може сприяти вирішенню цих проблем, оскільки сучасні реалізації машин станів легко оперують вкладеними та паралельними (складеними станами), а також чітко зберігають поточний стан, що вирішує першу та другу проблеми відповідно.

Іншою значною перевагою запропонованого підходу є наявність візуалізаторів машин станів, які дають змогу уявляти логіку застосунку у формі схем та чітко ідентифікувати поточний стан системи. Така наочність значно збільшує зрозумілість логічних потоків і станів системи. Графічні діаграми станів можуть слугувати відмінною технічною документацією, яка ніколи не втрачає актуальності, оскільки автоматично генерується з програмного коду. Така документація може бути особливо корисною для нових розробників у команді, яким зазвичай доводиться витрачати багато часу для ознайомлення з усіма наявними подіями у подійно-орієнтованому підході. Для розуміння подій і взаємозв'язків між ними доводиться повністю перечитувати програмний код їх опрацювання. Натомість із діаграмами станів вивчати взаємозв'язки між подіями значно легше. Також візуальне представлення може значно посприяти покращенню комунікації між різними учасниками проєкту. Воно буде корисним не тільки для технічних, а й для нетехнічних спеціалістів: проєктних менеджерів, бізнес-аналітиків, спеціалістів із підтримки та ін. Особливо корисною візуалізація буде для тестувальників, які зможуть чітко бачити, у якому стані перебувала система під час виявлення вади, та зможуть легко ділитися цією інформацією з розробниками. Приклад такої візуалізації зображено на рис. 3. Її було автоматично згенеровано з програмного коду.

Візуалізація показує діаграму станів машини станів, яка містить спрощений приклад процесу очищення даних. Спершу відбувається збір інформації про дані, які мають бути видалені, це робить один із мікросервісів. Далі паралельно відбуваються два процеси: видалення даних користувачів і видалення платіжної інформації, пов'язаної з цими користувачами. Цими задачами займаються два різні мікросервіси відповідно до своєї предметної області. Коли обидва завершать виконання, буде сформовано звіт і машина станів перейде у стан «completed». Також наявне оброблення помилок на кожному з кроків виконання (станів). У разі помилки інформацію про помилку буде записано в контекст машини станів (внутрішню пам'ять), і машина завершить своє виконання у стані «errored». Отже, усі процеси, які відбуваються під час очищення даних, є наочними та зрозумілими. Чим більші масштаби процесів, які відбуваються між мікросервісами, тим кориснішими будуть такі візуалізації.

Також запропонований підхід покращує можливості модульного і ручного тестування. Перевіряти взаємодію між мікросервісами досить складно. Найчастіше для досягнення цієї мети використовують інтеграційне або системне тестування, яке є значно складнішим, ніж модульне тестування. Покрити модульними тестами інтеграційну взаємодію мікросервісів досить складно, оскільки модульні тести будуть розкидані по всіх мікросервісах і будуть ніяк не пов'язані між собою, що робить їх написання та підтримку затратним завданням. Та якщо взаємодія мікросервісів буде представлена машиною станів, тоді таку взаємодію досить легко протестувати модульними тестами, оскільки машина станів є незалежною сутністю, яку може бути протестовано окремо (звісно, за умови використання заглушок для логіки, яка виходить за рамки машини станів).

Також виконання машини станів може бути симульовано, тобто можливе ручне тестування різних сценаріїв подій без побічних ефектів та необхідності запуску мікросервісів. Запуск машини станів у такому режимі може зекономити багато часу розробників, оскільки вони зможуть переконатися, що їхній логічний ланцюг потоку виконання програми правильний і не містить помилок. За використання традиційного подійно-орієнтованого підходу такого домогтися дуже складно; у переважній більшості випадків, щоб протестувати взаємодію між всіма мікросервісами, вони мають бути запущені. Водночас у разі використання машин станів переважну частину взаємодії можна перевірити, тільки симулюючи виконання машини станів, без необхідності запуску мікросервісів.

Рис. 3. Візуалізація машини станів, автоматично згенерована за допомогою візуалізатора бібліотеки XState

4. Висновки й обговорення

У статті описано підхід до інтеграції машин станів і мікросервісів із метою управління потоком виконання застосунку з мікросервісною архітектурою. Було створено бібліотеку, яка інтегрує між собою три ключові компоненти: реалізацію машин станів XState, брокер повідомлень Kafka і багатомодельну базу даних ArangoDB.

Створено також прототип, який ілюструє використання створеної бібліотеки для низки типових задач. Роботу прототипу проілюстровано на прикладі задачі видалення даних відповідно до регламенту GDPR. Показано шляхи до вирішення деяких із цих проблем в рамках розглянутої архітектури.

Запропонований підхід спрощує розроблення логіки виконання мікросервісного застосунку, а саме полегшує та упорядковує роботу з паралельними потоками виконання та довготривалими фоновими задачами. Крім того, він покращує прозорість процесів у мікросервісній архітектурі, а саме їх моніторинг, візуалізацію, тестування та відлагодження.

Водночас необхідно звернути увагу на певні проблемні аспекти, пов'язані з удосконаленням і практичним використанням розробленої бібліотеки. Очевидною є потреба в розширенні спектра типових задач, для яких її може бути застосовано. Вище йшлося про те, що здійснюється оброблення кожної окремої події, але це може створювати потенційні ускладнення при обробленні логічно пов'язаних послідовностей подій, тобто транзакцій, особливо за високого рівня завантаженості. Корисним видається також підвищення рівня формалізації моделей, що лежать в основі підходу. Все це має стати предметом подальших досліджень.

Список літератури

1. Apache Kafka documentation [Electronic resource]. — Mode of access: https://kafka.apache.org/documentation/ (date of access: 01.06.2024). — Title from screen.
2. ArangoDB documentation [Electronic resource]. — Mode of access: https://docs.arangodb.com/stable/ (date of access: 01.06.2024). — Title from screen.
3. Booch G. Unified modeling language user guide, the (2nd edition) (the addison-wesley object technology series) / Grady Booch, James Rumbaugh, Ivar Jacobson. — 2nd ed. — [S. l.] : Addison-Wesley Professional, 2005. – 496 p.
4. Harel D. Statecharts: a visual formalism for complex systems [Electronic resource] / David Harel // Science of computer programming. — 1987. — Vol. 8, no. 3. — Pp. 231–274. — https://doi.org/10.1016/0167-6423(87)90035-9.
5. Kurian George Cheripurathu. Integrating microservices and microfrontends: a comprehensive literature review on architecture, design patterns, and implementation challenges [Electronic resource] / Kurian George Cheripurathu, Sanjeev Kulkarni // Journal of scientific research and technology. — 2024. — Pp. 1–12. — https://doi.org/10.61808/jsrt115.
6. Lewis J. Microservices [Electronic resource] / James Lewis, Martin Fowler. — Mode of access: https://martinfowler.com/articles/microservices.html (date of access: 01.06.2024). — Title from screen.
7. Loukides M. Microservices adoption in 2020 [Electronic resource] / Mike Loukides, Steve Swoyer // O'Reilly Media. — Mode of access: https://www.oreilly.com/radar/microservices-adoption-in-2020/ (date of access: 02.07.2024). — Title from screen.
8. Michelson B. Event-Driven architecture overview [Electronic resource] / Brenda Michelson. — Boston, MA: Patricia Seybold Group, 2006. — https://doi.org/10.1571/bda2-2-06cc.
9. Microservices in practice: a survey study / Markos Viggiato [et al.] // VI workshop on software visualization, evolution and maintenance, São Carlos, 19 September 2018. — [S. l.].
10. Montelius A. An exploratory study of micro frontends [Electronic resource]: thesis / Montelius Anna. — [S. l.], 2021. — Mode of access: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-176963 (date of access: 11.09.2024). — Title from screen.
11. Söylemez M. Challenges and solution directions of microservice architectures: a systematic literature review [Electronic resource] / Mehmet Söylemez, Bedir Tekinerdogan, Ayça Kolukısa Tarhan // Applied sciences. — 2022. — Vol. 12, no. 11. — Pp. 5507. — https://doi.org/10.3390/app12115507.
12. State chart XML (SCXML): state machine notation for controlabstraction [Electronic resource] // W3C. — Mode of access: https://www.w3.org/TR/scxml/ (date of access: 01.06.2024). — Title from screen.
13. XState documentation [Electronic resource] // XState — JavaScript State Machines and Statecharts. — Mode of access: https://xstate.js.org/docs/ (date of access: 01.06.2024). — Title from screen.

References

Apache Kafka documentation. (n.d.). Apache Kafka. https://kafka.apache.org/documentation/.

ArangoDB documentation. (n.d.). ArangoDB. https://docs.arangodb.com/stable/.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified modeling language user guide, the (2nd edition) (the addison-wesley object technology series)* (2nd ed.). Addison-Wesley Professional.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, *8* (3), 231–274. https://doi.org/10.1016/0167-6423(87)90035-9.

Kurian, George Cheripurathu, & Kulkarni, Sanjeev. (2024). Integrating microservices and microfrontends: A comprehensive literature review on architecture, design patterns, and implementation challenges. *Journal of Scientific Research and Technology*, 1–12. https://doi.org/10.61808/jsrt115.

Lewis, J., & Fowler, M. (2014, March 25). *Microservices*. martinfowler.com. https://martinfowler.com/articles/microservices.html.

Loukides, M., & Swoyer, S. (2020, July 15). *Microservices adoption in 2020*. O'Reilly Media. https://www.oreilly.com/radar/microservices-adoption-in-2020/.

Michelson, B. (2006). *Event-Driven architecture overview*. Patricia Seybold Group. https://doi.org/10.1571/bda2-2-06cc.

Montelius, A. (2021). *An exploratory study of micro frontends* [Thesis, Linköpings universitet, Programvara och system]. http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-176963.

Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Challenges and solution directions of microservice architectures: A systematic literature review. *Applied Sciences*, *12* (11), 5507. https://doi.org/10.3390/app12115507.

State chart XML (SCXML): State machine notation for controlabstraction. (n.d.). W3C. https://www.w3.org/TR/scxml/.

Viggiato, M., Terra, R., Rocha, H., Tulio Valente, M., & Figueiredo, E. (n. d.). Microservices in practice: A survey study. In *VI workshop on software visualization, evolution and maintenance*.

XState documentation. (n.d.). XState — JavaScript State Machines and Statecharts. https://xstate.js.org/docs/.

O. Oletsky, V. Moholivskyi

COORDINATION OF MICROSERVICES USING STATE MACHINES

The article describes the use of state machines to coordinate microservices. The lack of well-developed microservice coordination tools presents a significant number of challenges for developers building applications with a microservice architecture. Among these challenges are the difficulty of managing multiple services and their distributed execution flow, the lack of transparency in monitoring and debugging, complications in testing, and other issues. These challenges can be addressed by utilizing state machines.

State machines serve as an abstraction to control the distributed execution flow of a program in a microservice architecture. This approach simplifies working with parallel execution flows and long-running background tasks. It also improves process transparency, as well as monitoring, visualization, testing, and debugging.

Integration with a message broker and adequate long-term state storage has been implemented to use state machines in a microservice environment. The paper describes the developed library, which connects three key components required for the effective functioning of state machines within microservices. These components include the XState implementation of the state machine, the Kafka message broker, and the ArangoDB multi-model database. Additionally, a prototype has been created to illustrate the usage of the developed library for a number of typical tasks. The prototype showcases the process of deleting data in accordance with GDPR regulations.

The proposed approach simplifies the development of execution logic in a microservice application, specifically by facilitating and organizing the management of parallel execution flows and long-running background tasks. In addition, it improves process transparency in the microservice architecture, namely in monitoring, visualization, testing, and debugging.

Keywords: microservice architecture, state machines, coordination of microservices, event-oriented approach, execution flow management, monitoring of microservices, visualization of microservices, testing of microservices.

Матеріал надійшов 12.08.2024