УДК 004.451.26
doi:10.21685/2072-3059-2021-2-2

Проблематика обработки транзакций при использовании микросервисной архитектуры

Д. С. Фомин[1], А. В. Бальзамов[2]

[1,2]Национальный исследовательский Мордовский государственный университет имени Н. П. Огарева, Саранск, Россия

[1]nkfominsa@gmail.com, [2]veron13rus@yandex.ru

Аннотация. *Актуальность и цели.* Объектом исследования является система электронной коммерции, построенная по принципу микросервисной архитектуры. Предметом исследования являются методы обеспечения корректной работы с транзакциями при использовании микросервисной архитектуры. Цель работы – поиск оптимального метода для решения проблемы обработки транзакций при использовании микросервисной архитектуры. *Материалы и методы.* Исследования проводились в области архитектурных решений при построении высоконагруженных систем электронной коммерции. Использовались методы двухфазной фиксации для обработки транзакций и паттерн-компенсационной транзакции – «Сага». *Результаты.* Проведен анализ особенностей работы с транзакциями и предложены методы решения проблемы обработки транзакций в системах, построенных с использованием микросервисной архитектуры. *Выводы.* Рассмотренные подходы, как правило, предполагают введение дополнительного сервиса (Координатора транзакций или Оркестратора Саги), которые управляют жизненным циклом транзакций, что несколько увеличивает затраты на разработку и ее сложность. Применяя описанные методы решения, система становится более отказоустойчивой, масштабируемой.

Ключевые слова: микросервисная архитектура, транзакция, база данных, сервис, бизнес-процесс

Для цитирования: Фомин Д. С., Бальзамов А. В. Проблематика обработки транзакций при использовании микросервисной архитектуры // Известия высших учебных заведений. Поволжский регион. Технические науки. 2021. № 2. С. 15–23. doi:10.21685/2072-3059-2021-2-2

The problem of transaction processing using microservice architecture

D.S. Fomin[1], A.V. Bal'zamov[2]

[1,2]Ogarev Mordovia State University, Saransk, Russia

[1]nkfominsa@gmail.com, [2]veron13rus@yandex.ru

Abstract. *Background.* The object of the research is an e-commerce system built on the principle of microservice architecture. The subject of the research is methods of ensuring correct operation of transactions using a microservice architecture. The purpose of the work is to find an optimal method for solving the problem of processing transactions using a microservice architecture. *Materials and methods.* Research was carried out in the field of architectural solutions for the construction of high-load e-commerce systems. T Two-phase commit methods were used to process transactions and a pattern-compensating transaction – “Saga”. *Results.* The research analyzes the features of working with transactions and pro-

poses methods for solving the problem of processing transactions in systems built using a microservice architecture. *Conclusions.* The approaches considered, as a rule, involve the introduction of additional services (Transaction Coordinator or Saga Orchestrator) that manage the life cycle of transactions, which increases development costs and complexity. Applying the described solution methods, the system becomes more fault-tolerant and scalable.

Keywords: microservice architecture, transaction, database, service, business process

For citation: Fomin D.S., Bal'zamov A.V. The problem of transaction processing using microservice architecture. *Izvestiya vysshikh uchebnykh zavedeniy. Povolzhskiy region. Tekhnicheskie nauki = University proceedings. Volga region. Engineering sciences.* 2021;2:15–23. (In Russ.). doi:10.21685/2072-3059-2021-2-2

Введение

При использовании классической монолитной архитектуры приложения зачастую используется единое централизованное хранилище данных (база данных), где имеется доступ сразу ко всем бизнес-объектам системы и при выполнении сложных бизнес-процессов можно использовать транзакции для повышения безопасности и целостности данных [1].

Как пример можно рассмотреть (рис. 1) использование транзакции в монолитной системе электронной коммерции.

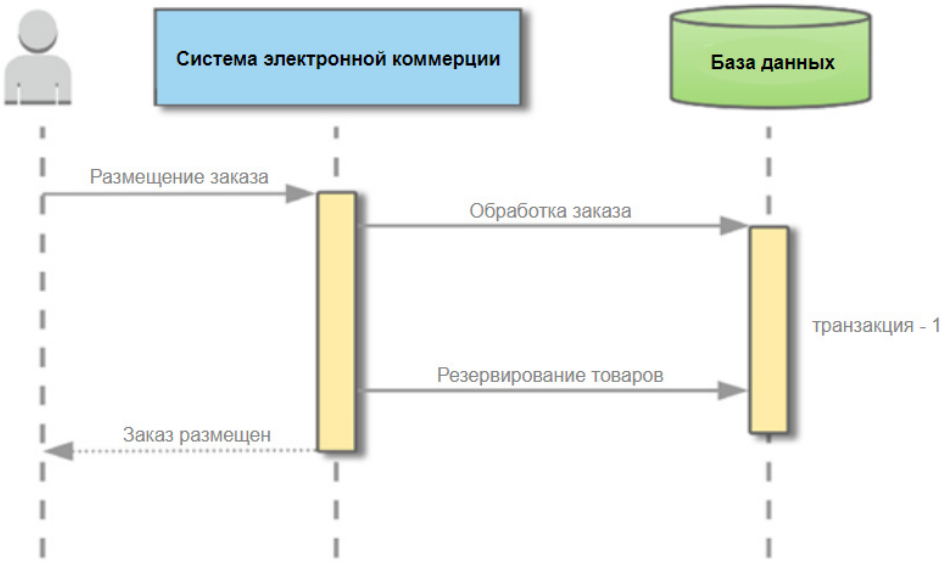

Рис. 1. Пример использования транзакции
в монолитной системе электронной коммерции

В данном примере пользователь отправляет запрос на оформление заказа в систему, система создает транзакцию локальной базы данных, которая работает с несколькими таблицами базы данных, чтобы обработать заказ и зарезервировать элементы из инвентаря. Если какой-либо шаг не удается, транзакция, заказ и зарезервированные позиции могут вернуться в первоначальное состояние. Все данные бизнес-операции выполняются в рамках одной транзакции.

Транзакции позволяют выполнить набор операций в базе данных, исполняясь как атомарная операция. Это говорит о том, что если одна команда из набора в транзакции выполнится с ошибкой, то вся транзакция будет полностью отклонена [2]. Принцип атомарности соответствует требованиям *ACID* (*Atomicity, Consistency, Isolation, Durability* – Атомарность, Последовательность, Изоляция, Долговечность), которые гарантируются системой управления базой данных [3].

При использовании «чистой» микросервисной архитектуры у каждого сервиса, который работает с бизнес-объектами, должна быть своя база данных для их обработки и хранения [4] (рис. 2).

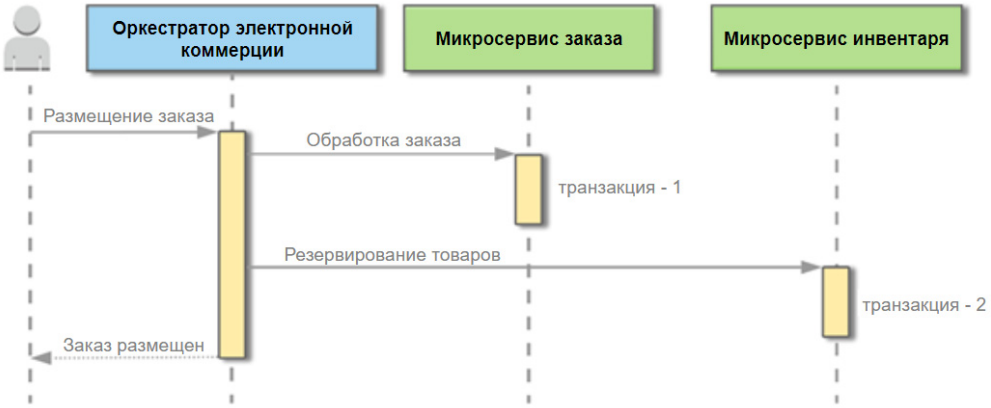

Рис. 2. Пример системы электронной коммерции
с использованием микросервисной архитектуры

При перестроении системы на микросервисную архитектуру выделяются все бизнес-процессы и их объекты, связанные с заказом в отдельный сервис «Микросервис заказа», а объекты и бизнес-процессы, связанные с резервирование и работой с элементами инвентаря, – в «Микросервис инвентаря» [5]. Теперь, когда от пользователя приходит запрос на оформление заказа, сначала данные передаются в «Микросервис заказа», где они обрабатываются и сохраняются, а затем последовательно или параллельно данные отправляются в «Микросервис инвентаря», где также выполняются свои бизнес-процессы в своей базе данных.

Поскольку теперь один бизнес-процесс «Размещение заказа» выполняется в разных сервисах и, как следствие, в разных базах данных, то нет возможности выполнить *ACID*-требования и возникают следующие проблемы:

1. Сохранение атомарности транзакции. Как говорилось выше, атомарность транзакции означает, что в транзакции либо завершены все операции, либо ни одна из операций не завершена и транзакция полностью откатывается. В приведенном выше примере, если бизнес-процессу, который занимается «резервированием товаров в инвентаре» в сервисе «Микросервис инвентаря», не удается провести резервирование или возникнут ошибки, необходимо откатить изменения бизнес-процесса «создания и обработки заказа», которые были применены в сервисе «Микросервис заказа» [6].

2. Обработка параллельных запросов, связанных с объектами, которые участвуют в исполняющемся бизнес-процессе. Так как теперь используется микросервисная архитектура, то бизнес-процесс по оформлению заказа

состоит из двух этапов (или из двух транзакций): выполнение всех бизнес-процессов по созданию и обработке заказа в сервисе «Микросервис заказа» и выполнение всех бизнес-процессов по резервированию элементов в «Микросервисе инвентаря». При исполнении бизнес-процесса оформления заказа, когда «Микросервис заказа» находится в стадии завершения исполнения своих бизнес-процессов, а сервис «Микросервис инвентаря» находится в стадии выполнения, при поступлении параллельного запроса, связанного с объектами «заказов», необходимо включать в набор данных заказ, даже если основной бизнес-процесс еще не завершен [6].

Две вышеупомянутые проблемы очень важны при проектировании и создании приложений или систем на основе микросервисной архитектуры. Для решения данных проблем существует несколько основных методов:

1) метод двухфазной фиксации;

2) паттерн «Сага» [7].

1. Метод двухфазной фиксации

Как следует из названия, этот способ обработки транзакций состоит из двух этапов: фазы подготовки и фазы фиксации. Одним из важных участников является координатор транзакции, который поддерживает жизненный цикл транзакции [7].

При исполнении основного бизнес-процесса по оформлению заказа все сервисы создают у себя в локальных базах данных транзакции и отправляют уведомление координатору о готовности их фиксации. Затем по готовности всех сервисов, участвующих в основном бизнес-процессе, координатор транзакций отправляет команду всем сервисам о применении фиксации в случае успеха или команду отката в случае ошибок или иных обстоятельств [8].

Пример системы электронной коммерции с использованием микросервисной архитектуры и координатора транзакций изображен на рис. 3.

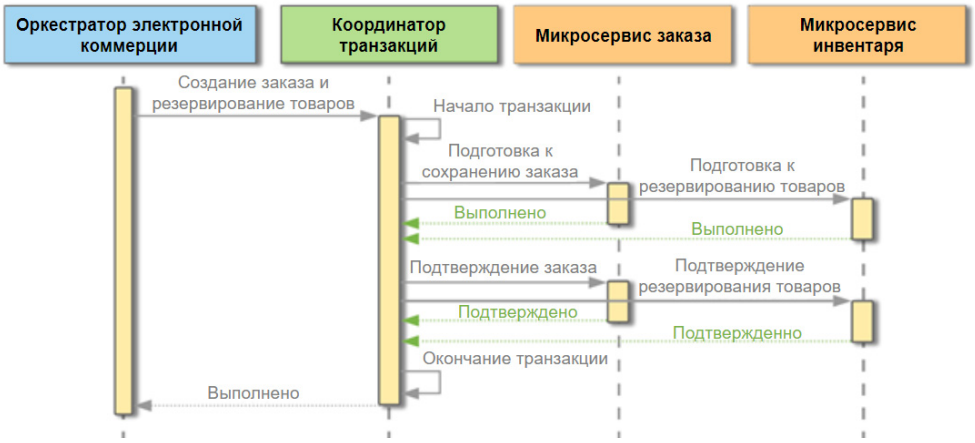

Рис. 3. Пример системы электронной коммерции с использованием микросервисной архитектуры и координатора транзакций

В приведенном выше примере, когда пользователь отправляет запрос на оформление заказа, «Координатор транзакций» создает виртуальную глобальную транзакцию со всей контекстной информацией. Затем данные от-

правляются в «Микросервис заказа» для исполнения бизнес-процессов по созданию и обработке заказа и подготовке локальной транзакции. После этого данные отправляются в «Микросервис инвентаря» для исполнения бизнес-процессов по резервированию элементов инвентаря и подготовке локальной транзакции. Когда все сервисы готовы, они уведомляют координатора транзакций о готовности фиксации и блокируют объекты от дальнейших изменений. Как только все сервисы уведомляют «Координатор транзакций» об исполнении всех своих бизнес-процессов и готовности фиксации транзакции, он отправляет команду этим сервисам о применении их фиксации. Затем данные сервисы применяют фиксацию транзакции и далее все объекты будут разблокированы [8].

Также необходимо рассмотреть процесс в случае возникновения ошибки и невозможности одним из сервисов применения фиксации транзакции. Данный процесс изображен на рис. 4.

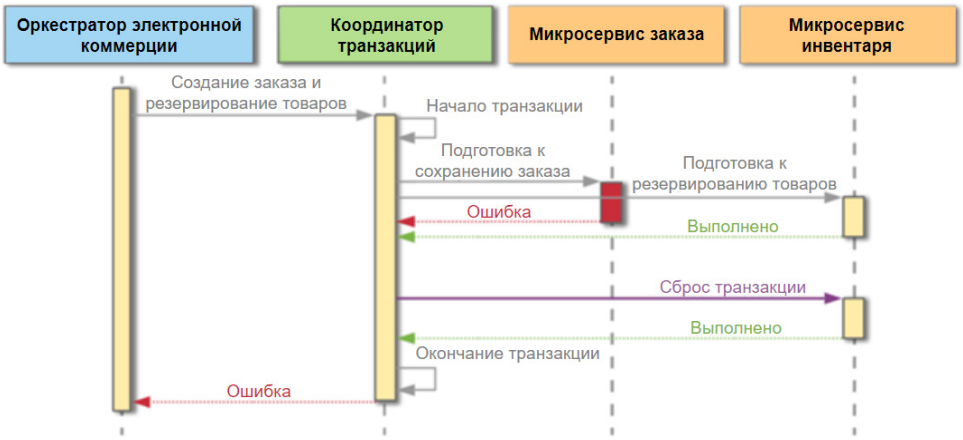

Рис. 4. Пример системы электронной коммерции с использованием микросервисной архитектуры и координатора транзакций в случае сбоя бизнес-процесса

В сценарии сбоя основного бизнес-процесса по оформлению заказа, если в какой-то момент один из сервисов не может выполнить все свои бизнес-процессы и подготовить локальную транзакцию к фиксации, то «Координатор транзакций» прервет свою виртуальную глобальную транзакцию и начнет процесс отката. Как показано на рис. 4, сервис «Микросервис заказа» не смог выполнить свои бизнес-процессы по созданию и обработке заказа, но сервис «Микросервис инвентаря» уведомил, что выполнил все свои бизнес-процессы по резервированию элементов инвентаря и перевел локальную транзакцию к стадии фиксации. В этом случае «Координатор транзакций» отправит команду отката транзакции сервису «Микросервис инвентаря», тем самым весь бизнес-процесс будет отменен, объекты будут разблокированы и никакие изменения не попадут в локальные базы данных сервисов [9].

Рассмотренный метод имеет следующие преимущества:

1. Подход гарантирует атомарность транзакции. Транзакция завершится либо при успешном завершении всех бизнес-процессов во всех участвующих сервисах, либо все базы данных останутся в начальном состоянии.

2. Во-вторых, метод позволяет изолировать чтение и запись, изменения объектов не видны, пока координатор транзакций не зафиксирует изменения.

3. Подход может представлять собой синхронный вызов, при котором клиент будет уведомлен об успехе или неудаче.

Также нельзя исключить следующие недостатки указанного метода:

1. Двухфазная фиксация выполняется довольно медленно по сравнению со временем работы одного сервиса. Они сильно зависят от координатора транзакций, что действительно может замедлить работу системы при высокой нагрузке.

2. Другой главный недостаток – блокировка строк базы данных. Блокировка может стать узким местом производительности, и возможна тупиковая ситуация, когда две транзакции взаимно блокируют друг друга.

2. Паттерн «Сага»

Паттерн «Сага» представляет собой набор локальных транзакций [10]. Каждая локальная транзакция обновляет базу данных и публикует сообщение или событие, инициируя следующую локальную транзакцию в саге. Если транзакция завершилась неудачей, например из-за нарушения бизнес-правил, тогда сага запускает компенсирующие транзакции, которые откатывают изменения, сделанные предшествующими локальными транзакциями. В этом подходе распределенная транзакция выполняется асинхронными локальными транзакциями на связанных микросервисах. Микросервисы связываются друг с другом через шину событий [4].

Существует два способа координации саг:

• Хореография (*Choreography*) – каждая транзакция публикует события, которые запускают транзакции в других сервисах.

• Оркестровка (*Orchestration*) – оркестратор говорит участникам, какие транзакции должны быть запущены.

В качестве примера можно рассмотреть ранее описанную систему электронной коммерции с использованием микросервисной архитектуры и паттерна «Сага» (рис. 5).

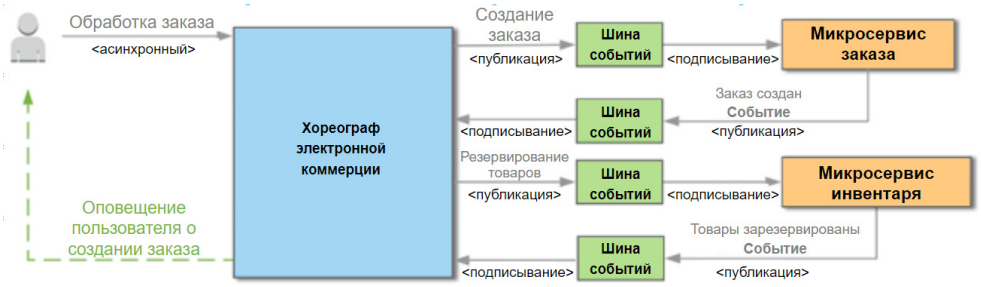

Рис. 5. Пример системы электронной коммерции
с использованием микросервисной архитектуры и паттерна «Сага»

В приведенном выше примере, когда клиент отправляет запрос на оформление заказа, «Хореограф» генерирует событие «Создание заказа», означающее начало транзакции. В свою очередь «Микросервис заказа» прослушивает это событие и запускает бизнес-процессы, связанные с созданием и обработкой заказа, и, если он был успешным, генерирует событие «Заказ создан». После этого «Хореограф», получая событие от «Микросервиса заказа» об успешном создании и обработке заказа переходит к резервированию предметов, генерируя событие «Резервирование товаров». «Микросервис ин-

вентаря» также отслеживает это событие и запускает бизнес-процессы, связанные с резервированием элементов, и в случае успеха генерирует событие «Товары зарезервированы». Данное событие сигнализирует «Хореографу» об успешном выполнении всего основного бизнес-процесса по оформлению заказа и конце саги (транзакций) [11].

Вся связь на основе событий между микросервисами происходит через шину событий и оркестрируется другой системой для решения проблемы сложности [12].

На рис. 6 изображен сценарий сбоя основного бизнес-процесса по оформлению заказа. Например, если по какой-либо причине «Микросервису инвентаря» не удалось выполнить все свои бизнес-процессы по резервированию элементов, он генерирует событие «Ошибка резервирования товаров». «Хореограф», получая данное событие, запускает «компенсационную транзакцию», генерируя событие «Удаление заказа». «Микросервис заказа» прослушивает это событие и запускает бизнес-процесс и соответствующие транзакции по удалению данного заказа [13].

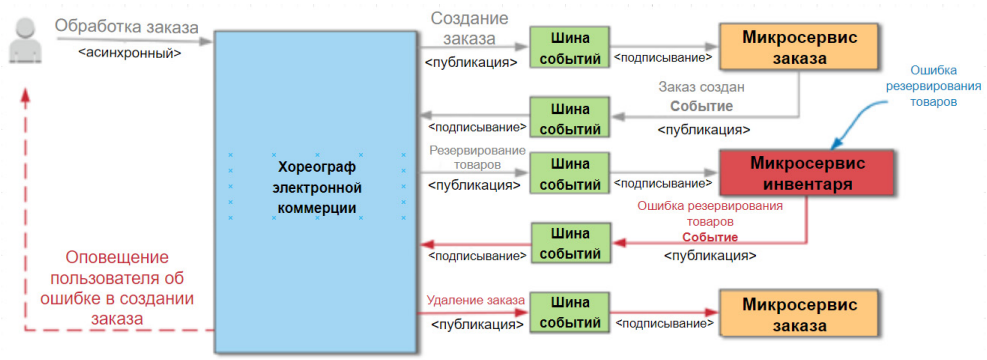

Рис. 6. Пример системы электронной коммерции с использованием микросервисной архитектуры и паттерна «Сага» в случае сбоя бизнес-процесса

Одним из больших преимуществ этого подхода является то, что каждый микросервис фокусируется только на своей атомарной транзакции. Микросервисы не блокируются, если другой сервис занимает больше времени. Это также означает, что блокировка базы данных не требуется. Использование этого подхода делает систему хорошо масштабируемой при большой нагрузке благодаря решению, основанному на асинхронных событиях [11, 14].

Основным недостатком этого подхода является отсутствие изоляции чтения. Это означает, что в приведенном выше примере клиент мог видеть, что заказ был создан, но в следующую секунду заказ удаляется из-за компенсирующей транзакции. Кроме того, когда количество микросервисов увеличивается, их становится труднее отлаживать и поддерживать [11, 15].

Заключение

Рассмотренные подходы по обработке распределенных транзакции достаточно актуальны в нынешнее время, так как популярность микросервисной архитектуры из года в год становится все выше. Рассмотренные в данной статье проблемы, подходы для их решения с преимуществами и недостатками дают разработчикам целостную картину по работе с данными и их источ-

никами в рамках микросервисной архитектуры. Эти подходы как правило предполагают введение дополнительного сервиса («Координатора транзакций» или «Оркестратора Саги»), которые управляют жизненным циклом транзакций, что несколько увеличивает затраты на разработку и ее сложность. Применяя данные методы решения, система становится более отказоустойчивой, масштабируемой и, наверное, без неопределенного поведения при выполнении серьезных бизнес-процессов, связанных с различными источниками данных.

Список литературы

1. Фаулер М. Шаблоны корпоративных приложений. М. : Диалектика-Вильямс, 2019. 546 с.
2. Лейн К., Черети М. Базы данных. Инжиниринг надежности. СПб. : Питер, 2020. 304 с.
3. Баженова И. Ю. Разработка распределенных приложений баз данных : курс лекций. М. : МГУ им. М. В. Ломоносова, 2006. 203 с.
4. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. СПб. : Питер, 2018. 352 с.
5. Ильин В. В. Моделирование бизнес-процессов. Практическое использование ARIS. М. : Диалектика-Вильямс, 2006. 176 с.
6. Goetsch K. Microservices for Modern Commerce: Report. CA : O'Reilly Media, 2017. P. 25–47.
7. Проблематика распределенных транзакций в контексте микросервисной архитектуры. URL: https://habr.com/ru/company/otus/blog/516720/ (дата обращения: 08.03.2021).
8. Nadareishvili I., Mitra R., McLarty M., Amundsen M. Microservice architecture. CA : O'Reilly Media, 2016. P. 25–38.
9. Committing a Transaction in Single-Phase and Multi-Phase. URL: https://docs.microsoft.com/ru-ru/dotnet/framework/data/transactions/committing-a-transaction-in-single-phase-and-multi-phase (дата обращения: 07.03.2021).
10. Saga distributed transactions. URL: https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga (дата обращения: 08.03.2021).
11. Saga pattern and distributed transactions. URL: https://bool.dev/blog/detail/saga-pattern-i-raspredelennye-tranzaktsii (дата обращения: 08.03.2021).
12. Применение паттерна сага. URL: https://medium.com/@evgenzt/%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%D0%BF%D0%B0%D1%82%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%81%D0%B0%D0%B3%D0%B0-7eead841d90f (дата обращения: 08.03.2021).
13. Sagas. URL: https://microservices.io/patterns/data/saga.html (дата обращения: 08.03.2021).
14. Паттерн: Сага. URL: https://itnan.ru/post.php?c=1&p=427705 (дата обращения: 09.03.2021).
15. Паттерн: Сага. URL: https://habr.com/ru/company/otus/blog/519636/ (дата обращения: 09.03.2021).

References

1. Fauler M. *Shablony korporativnykh prilozheniy = Enterprise application templates*. Moscow: Dialektika-Vil'yams, 2019:546. (In Russ.)
2. Leyn K., Chereti M. *Bazy dannykh. Inzhiniring nadezhnosti = Database. Reliability engineering*. Saint-Petersburg: Piter, 2020:304. (In Russ.)
3. Bazhenova I.Yu. *Razrabotka raspredelennykh prilozheniy baz dannykh: kurs lektsiy = Developing distributed database applications: a course of lectures*. Moscow: MGU im. M. V. Lomonosova, 2006:203. (In Russ.)

4. Martin R. *Chistaya arkhitektura. Iskusstvo razrabotki programmnogo obespecheniya = Clean architecture. The art of software engineering*. Saint-Petersburg: Piter, 2018:352. (In Russ.)
5. Il'in V.V. *Modelirovanie biznes-protsessov. Prakticheskoe ispol'zovanie ARIS = Business process modeling. Practical use of ARIS*. Moscow: Dialektika-Vil'yams, 2006:176. (In Russ.)
6. Goetsch K. *Microservices for Modern Commerce: Report*. CA: O'Reilly Media, 2017:25–47.
7. *Problematika raspredelennykh tranzaktsiy v kontekste mikroservisnoy arkhitektury = Issues of distributed transactions in the context of a microservice architecture*. Available at: https://habr.com/ru/company/otus/blog/516720/ (accessed 08.03.2021). (In Russ.)
8. Nadareishvili I., Mitra R., McLarty M., Amundsen M. *Microservice architecture*. CA: O'Reilly Media, 2016:25–38.
9. *Committing a Transaction in Single-Phase and Multi-Phase*. Available at: https://docs.microsoft.com/ru-ru/dotnet/framework/data/transactions/committing-a-transaction-in-single-phase-and-multi-phase (accessed 07.03.2021).
10. *Saga distributed transactions*. Available at: https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga (accessed 08.03.2021).
11. *Saga pattern and distributed transactions*. Available at: https://bool.dev/blog/detail/saga-pattern-i-raspredelennye-tranzaktsii (accessed 08.03.2021).
12. *Applying the saga pattern*. Available at: https://medium.com/@evgenzt/%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%D0%BF%D0%B0%D1%82%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%81%D0%B0%D0%B3%D0%B0-7eead841d90f (accessed 08.03.2021).
13. *Sagas*. Available at: https://microservices.io/patterns/data/saga.html (accessed 08.03.2021).
14. *Pattern: Saga*. Available at: https://itnan.ru/post.php?c=1&p=427705 (accessed 09.03.2021).
15. *Pattern: Saga*. Available at: https://habr.com/ru/company/otus/blog/519636/ (accessed 09.03.2021).

Информация об авторах / Information about the authors

Дмитрий Сергеевич Фомин
аспирант, Национальный исследовательский Мордовский государственный университет имени Н. П. Огарева (Россия, г. Саранск, ул. Большевистская, 68)

E-mail: nkfominsa@gmail.com

Dmitriy S. Fomin
Postgraduate student, Ogarev Mordovia State University (68 Bolshevistskaya street, Saransk, Russia)

Александр Витальевич Бальзамов
аспирант, Национальный исследовательский Мордовский государственный университет имени Н. П. Огарева (Россия, г. Саранск, ул. Большевистская, 68)

E-mail: veron13rus@yandex.ru

Aleksandr V. Bal'zamov
Postgraduate student, Ogarev Mordovia State University (68 Bolshevistskaya street, Saransk, Russia)

Поступила в редакцию / Received 22.03.2021

Поступила после рецензирования и доработки / Revised 13.04.2021

Принята к публикации / Accepted 21.05.2021