СТАТЬИ

УДК 004.4’2:641

К ВОПРОСУ О ПЕРСПЕКТИВАХ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ В РАЗРАБОТКЕ ФУДТЕХ-ПРИЛОЖЕНИЙ

Кукитз П.В.

ООО «Рестомания», Москва, e-mail:paulkukitz@gmail.com

Статья посвящена исследованию общих вопросов и перспектив использования микросервисной архитектуры в разработке приложений для фудтех-индустрии. Рассматриваются специфические особенности разработки программных продуктов для сферы заказа еды в онлайн-приложениях, в которых использование микросервисной архитектуры особенно востребовано. Выделяются ключевые характеристики фудтех-приложений: высокая вариативность функций, необходимость быстрой адаптации к изменениям рынка, неравномерная нагрузка на приложения и потребность в интеграции с внешними сервисами. Анализируется, каким образом микросервисная архитектура позволяет эффективно решать задачи, связанные с обозначенными особенностями приложений фудтех индустрии. Рассматриваются способы обеспечения согласованности данных в микросервисной архитектуре, а также приводятся рекомендации для разработчиков, которые помогут избежать ошибок и улучшить процесс разработки приложений с микросервисной архитектурой под нужды и задачи фудтеха. Особое внимание уделяется вопросам минимизации зависимостей, версионирования API, автоматического обнаружения сервисов и обеспечения отказоустойчивости системы как ключевых преимуществ-перспектив перехода на микросервисную архитектуру разработки. По итогам проведенного исследования делаются выводы о том, что сегодня микросервисная архитектура фактически становится стандартом для разработки приложений в фудтех индустрии, поскольку обеспечивает масштабируемость и высокую производительность приложений.

Ключевые слова: микросервисная архитектура, фудтех-приложения, разработка программного обеспечения, вариативность функций, масштабируемость, отказоустойчивость, микросервисы

ON THE PROSPECTS OF MICROSERVICE ARCHITECTURE IN FOODTECH APPLICATION DEVELOPMENT

Kukitz P.V.

Restomania LLC, Moscow, email:paulkukitz@gmail.com

This article explores the general aspects and prospects of utilizing microservice architecture in the development of applications for the foodtech industry. It examines the specific characteristics of software development for online food ordering applications, where the use of microservice architecture is particularly relevant. The article identifies key features of foodtech applications: a wide variety of functions, the need for rapid adaptation to market changes, uneven load on applications, and the requirement for integration with external services. The article analyzes how microservice architecture effectively addresses the challenges associated with these features within the foodtech industry. Furthermore, it explores methods for ensuring data consistency within a microservice architecture and provides recommendations for developers to mitigate errors and enhance the development process of foodtech-specific microservice applications. Particular attention is given to dependency minimization, API versioning, automatic service discovery, and system fault tolerance as key advantages and prospects of transitioning to microservice-based development. The study concludes that microservice architecture is becoming the de facto standard for developing applications in the foodtech industry as it ensures scalability and high application performance.

Keywords: microservice architecture, foodtech applications, software development, functionality variability, scalability, fault tolerance, microservices

Введение

В условиях роста требований к качеству и универсальности разработки программного обеспечения для фудтех индустрии (фудтех-приложения), вопросы рассмотрения различных вариаций развертывания программного обеспечения приобретают основополагающее значение, сопряженное с необходимостью обоснования целесообразности тех или иных решений, которые применяются разработчиками. С точки зрения актуальных трансформаций бизнеса и постоянного расширения потребности в формировании многофункциональных, универсальных и адаптированных программных приложений, особенно перспективный и прикладной характер приобретает тема исследования микросервисной архитектуры в разработке приложений под нужды фудтех индустрии. Состояние конкурентной среды в фудтехе достаточно вариативно и изменчиво, что формирует главную проблему, с которой сталкиваются практически все разработчики данной отрасли – необходимость постоянного внесения изменений и актуализации приложений, с одной стороны, и высокое значение подстраивания под ожидания и целевые запросы потребителей, с другой. Именно

микросервисная архитектура обладает потенциалом решения данных проблем, в связи с чем популярность разработки решений с микросервисной архитектурой, доступных на рынке, постепенно увеличивается.

Цель исследования заключается в определении перспектив микросервисной архитектуры при разработке приложений для фудтех индустрии.

Материалы и методы исследования

Теоретико-методологическим базисом настоящего исследования послужили труды ученых, затрагивающие концептуальные основы, специфические особенности, проблемы и инструменты формирования микросервисной архитектуры. Немаловажную роль в структуре исследования заняли работы, раскрывающие технические тонкости применения микросервисной архитектуры, которые позволили сформировать перечень ключевых факторов целесообразности перехода на микросервисную разработку приложений для индустрии фудтеха.

В основу исследования положен комплекс методов: теоретический анализ научной литературы по теме исследования; синтез; библиографическое описание; формально-логический анализ; абстракция и некоторые другие.

Результаты исследования и их обсуждение

Вопросы целесообразности перехода на применение микросервисной архитектуры в деятельности разработчика в фудтех индустрии во многом связываются со специфическими особенностями разработки программных продуктов под нужды данной индустрии. Анализ научной литературы по теме исследования позволил выделить существенный пробел, выраженный в отсутствии комплексных исследований, которые бы точно резюмировали специфические особенности фудтех-приложений, обосновывающих привлекательность микросервисной архитектуры для них.

Отметим, что микросервисная архитектура представляет, как верно замечает Д.А. Кравченко, определенный подход к разработке приложений или программных продуктов, в рамках которого работоспособность всей системы обеспечивается за счет работы независимых сервисов, которые образуют единую архитектуру приложения и при этом подконтрольны отдельным командам разработчиков, ориентированных на конкретный микросервис [1]. Примечательными в контексте рассмотрения монолитной и микросервисной архитектуры видится исследование Е.А. Пахмуриной, которая рассматривает микросервисную архитектуру в качестве технического преимущества приложения, которое обеспечивает высокое качество работоспособности системы ввиду функциональных возможностей [2]. Однако во многом факторы «качества» диктуются конкретной командой и задачей в разработке; на наш взгляд, целесообразным видится рассмотреть преимущества и перспективы микросервисной архитектуры с привязкой к приложениям фудтех индустрии.

Как показывает практический опыт автора настоящего исследования, особенности разработки фудтех приложений во многом перекликаются с концептуальными преимуществами микросервисной архитектуры, в случае их применения как фундамента развертывания программных продуктов и приложений. В целом, для индустрии фудтеха характерны следующие особенности, связанные с разработкой приложений:

• высокая вариативность доступных функций, начиная от заказа еды, оформления доставки, заканчивая выставлением рейтингов ресторанам, использования программ лояльности и прочих подфункций большой системы заказа еды. В особенности подобное характерно для крупных агрегаторов; например, приложения Деливери или Яндекс.Еда оснащены целой системой функций, которые не ограничиваются только доставкой еды. В частности, можно предположить, что в основу программной части таких приложений положена система микросервисов непосредственно заказа еды с выбором формы оплаты и оформлением доставки (ядро агрегатора) и подключенными к ним подсистем отслеживания статуса заказа, слежения за передвижением курьера, рейтингов, обзоров, микросервисов персонализации, программ лояльности, а также подключения других сервисов для получения специфических функций. Вариативность функций в данном случае обосновывает целесообразность перехода на микросервисную архитектуру, поскольку возникает потребность параллелизировать выполнение функций и оптимизировать все приложение. Разделение на микросервисы позволяет снизить нагрузку на «большую» систему, что положительно сказывается на качестве исполнении функций, скорости работы всего приложения;

• необходимость быстрого внесения изменений в отдельные подсистемы, функции или сервисы без затрагивания всей системы, что позволяет быстро и без значительных проблем адаптировать отдельные части приложения под текущие тренды, без необходимости изменения всей системы фудтех

приложения. На примере все тех же вышеупомянутых популярных агрегаторов, заметим, что каждый сервис отвечает за собственные функции; например, появление нового ресторана в перечне доступных в приложении потребует внесения изменений в базу данных сервиса ресторана и отдельные подсистемы, при этом практически не затронет сервис доставки или сервис заказа. Необходимость частного расширения функций (масштабируемости) – ключевое обоснование перехода на микросервисную архитектуру, которая, впрочем, часто используется в фудтех индустрии;

• неравномерная нагрузка на приложения, поскольку, как правило, пиковый спрос наблюдается в периоды «обеденных» и «вечерних» часов. Пиковая нагрузка в часы потребления формирует вызовы для работоспособности и отказоустойчивости традиционных монолитных систем, которые в индустрии фудтеха демонстрируют, исходя из нецеленаправленных наблюдений автора, частые нарушения работы, большое количество системных ошибок. Переход на микросервисную архитектуру фудтех-приложения позволяет масштабировать отдельные сервисы или их компоненты по мере необходимости, обеспечивать за счет этого стабильную работу приложения даже при большом количестве единовременно взаимодействующих пользователей. К тому же сбои в одном сервисе не всегда приводят к парализации всего приложения, что также является преимуществом для службы технического контроля, у которой появляется возможность с меньшими потерями для бизнеса восстановить работоспособность системы микросервиса без вмешательства во всю структуру приложения;

• наличие объективной потребности потребителей связывать агрегаторы доставок и фудтех-приложения с внешними приложениями, что свидетельствует в пользу необходимости обеспечивать открытость архитектуры. Интеграция к микросервисной архитектуре происходит несколько проще, поскольку может осуществляться в отдельной системе API, без необходимости обеспечивать прямую связь с ключевыми подсистемами приложения. Однако наличие API в виде отдельного микросервиса не освобождает разработчиков от решения задач в области безопасности, проектирования, согласования версий и обновления, управления нагрузками и согласования данных.

Отметим, что обозначенные преимущества микросервисной архитектуры находят подтверждение и в научной литературе, не связанной конкретно с разработкой приложений для индустрии фудтеха. Так, по мнению Д.А. Сабирова, формирование микросервисов для пользователей и для разработчиков связывается с собственными особенностями и преимуществами. Автор отмечает, что для разработчика ориентированность на микросервисную архитектуру обеспечивает четкую реализацию логики функционирования бизнеса при параллелизации работы команд; подобное эффективно для управления разработкой и ускорения работы системы, с возможностью перевода отдельных команд и их участников в другие проекты при более быстром завершении разработки и отладке микросервиса. Кроме того, вся система приложения, как подчеркивает автор, функционирует независимо, что позволяет быстрее, проще и с большей эффективностью вносить изменения, включать новые функции, осуществлять обновления, масштабировать, локализовать ошибки без необходимости «отключения» других подсистем приложения [3]. А.М. Шитько отмечает, что микросервисная архитектура фактически позволяет «отказаться» от традиционных недостатков монолитных систем, однако скрывает за собой собственные нюансы и вопросы разработки. Например, для работы микросервисной архитектуры возникает потребность в привлечении дополнительных специалистов в сфере DevOps, которые ответственны за развертывание приложения. К руководителю проектом разработки (техническому директору) предъявляются дополнительные требования в управлении командой, с фокусом на обеспечение логического, точного и быстрого исполнения функций и задач, за которые ответственна каждая микрокоманда. В конечном счете автор в своем исследовании подчеркивает необходимость переформирования монолитных приложений в систему микросервисов, что обеспечит, по его мнению, более эффективное исполнение функций и общую работоспособность приложения [4]. Примечательным видится и исследование С.Г. Косова и Л.В. Кальянова, которые подчеркивают, что переход на микросервисную архитектуру позволяет упростить проведение процедур тестирования приложений за счет отказа от интеграционного тестирования, в тоже время, связывается с другими сложностями тестирования, что указывает на неоднозначность трансформаций [5]. Исходя из представленных исследований, важно подчеркнуть, что микросервисная архитектура не избавляет от проблем разработки и не становится способом упрощения; она, скорее, должна рассматриваться как способ обеспечить более высокое качество готового продукта.

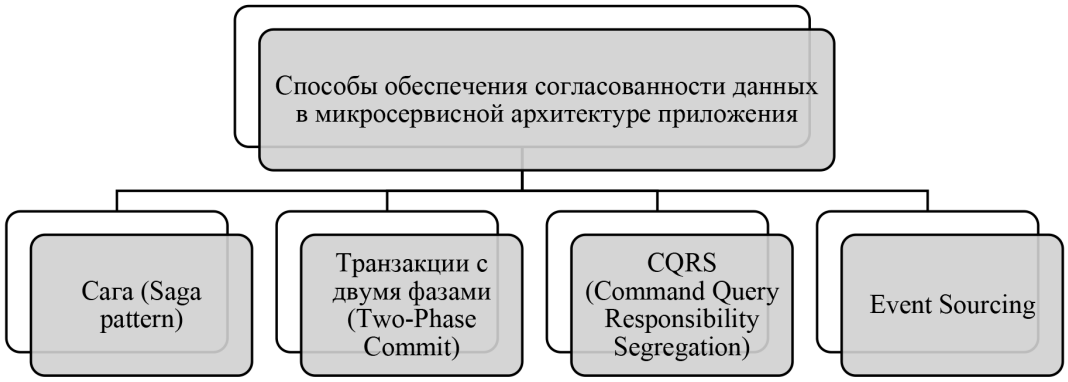

Рис. 1. Способы обеспечения согласованности данных в микросервисной архитектуре приложения
Источник: составлено автором

Рассматривая недостатки микросервисной архитектуры, отметим, что ключевая проблема таковой – обеспечить согласованность работы микросервисов, актуальный обмен данными и снизить уровень задержки, с которой сталкивается пользователь. Аналогичные проблемы встречаются и в научной литературе. Например, Н.Ю. Кучеренко предлагает уделять особое внимание проблеме целостности данных при разработке приложений с микросервисной архитектурой [6]. На практике обеспечить согласованность данных между микросервисами удается за счет системы инструментов (рис. 1).

Д.Б. Осипов в своём исследовании приводит некоторые рекомендации для разработчиков, осуществляющих работу с приложениями с микросервисной архитектурой; во многом данные рекомендации связываются с предупреждением проблем и применением механизмов улучшения процесса разработки, позволяют снизить число ошибок при разработке приложения [7]. Аналогичные рекомендации выделяет В.А. Ворсин, который подчеркивает необходимость предупреждения типовых проблем микросервисной архитектуры [8].

Как замечает М.Э. Джалалов, важно управлять версионностью API, что во многом обеспечивает качество и работоспособность всего приложения. Автор предлагает систему методов управления версионностью API (рис. 2).

Так, резюмируя результаты проведенного исследования, заметим, что среди ключевых рекомендаций для разработчиков микросервисных приложений современные авторы выделяют:

Во-первых, необходимость комплексного проектирования системы сервисов, что предполагает наделение каждого сервиса отдельной функцией (блоком функций) будущего приложения; формирование слабой связанности с независимостью сервисов и качества их работы друг от друга; с определением точного соприкосновения внешних интерфейсов.

Создание отдельных версий для каждого микросервиса	Централизованный подход	Использование API шлюзов
•Гибкость в разработке, паралелизация работы над сервисами •Сложность управления зависимостями, проблемы интеграции и совместимости	•Консистентность в системе, упрощение управления тестированием •Ограниченная гибкость, проблема масштабируемости больших систем	•Централизация управления версиями, изоляция клиентов от изменений •Создание централизованной точки отказа, увеличение сложности системы

Рис. 2. Методы управления версионностью API в микросервисной архитектуре приложения по М.Э. Джалалову [9]

Для разработчиков фудтех-приложений данная рекомендация сводится к созданию отдельных микросервисов для: 1) заказа блюд (обработка запросов пользователей, взаимодействие с меню ресторанов); 2) оплаты (интеграция с платежными системами, обработка транзакций); 3) доставки (отслеживание местоположения курьеров, оптимизация маршрутов); 4) рейтинга (сбор и отображение отзывов, управление рейтингом ресторанов); 5) лояльности (обработка бонусных программ, персональных предложений).

Во-вторых, значимость постоянного управления зависимостями, что предполагает их минимизацию, создание версий API для обратной совместимости сервисов, а также автоматическое обнаружения сервисов, чтобы исключить привязку к конкретным адресам или портам.

С точки зрения фудтех-приложений, важно привести несколько примеров управления зависимостями. Так, например, микросервис оплаты заказа не должен «знать» детали реализации сервиса доставки, поскольку ему достаточно получить подтверждение, что доставка возможна, прежде чем обрабатывать платеж. Аналогично с точки зрения версий; например, при изменении API сервиса меню разработчику целесообразнее создать новую версию API (v2), оставить старую версию (v1) доступной для сервисов, которые еще не адаптированы. С позиции автоматического обнаружения сервисов, вместо жесткой привязки к IP-адресу и порту сервиса лояльности необходимо использовать service discovery. Сервис заказа автоматически найдет сервис лояльности через регистр сервисов.

В-третьих, фокус на отказоустойчивости системы, чтобы обеспечить получение реальных преимуществ от микросервисной архитектуры, что предполагает постоянную работу с ошибками приложения, исключения каскадных сбоев, а также осуществление масштабирования приложений.

Так, для фудтех-приложений можно выделить несколько ключевых рекомендаций по обеспечению отказоустойчивости системы. Например, сервис оплаты должен корректно обрабатывать ситуации, когда платежный шлюз недоступен. Вместо сбоя всей системы пользователь должен получить сообщение о временной невозможности оплаты и предложение повторить попытку (провести оплату) позже. Если сервис меню недоступен, сервис заказа не должен «падать» вместе с ним; пользователю можно демонстрировать ограниченное меню из кэша или предложить выбрать другой ресторан, что изолирует от проблем. В часы пиковой активности нагрузка на сервис заказа резко возрастает. Чтобы система не «захлебнулась», нужно заранее предусмотреть возможность горизонтального масштабирования – запуска дополнительных экземпляров сервиса для обработки пиковой нагрузки, что обеспечит производительность и отказоустойчивость системы.

Заключение

Таким образом, переход на микросервисную архитектуру приложений в фудтех-индустрии является объективной необходимостью, с которой сталкиваются разработчики приложений. В индустрии фудтеха микросервисная архитектура становится прикладным способом обеспечения работоспособности, отказоустойчивости и быстрого масштабирования программной части приложения, что подтверждается как распространенной практикой крупнейших агрегаторов доставки еды, так и проведенным анализом научной литературы. Подобное отличается системой перспектив как с точки зрения технической части (разработка), так и финансово-экономических возможностей, предупреждает убытки от технических сбоев и потерь системы, вызванных традиционными недостатками монолитной разработки. Учитывая высокий динамизм и изменчивость фудтех индустрии, для разработчиков микросервисная архитектура становится единственно верным решением, при учете необходимости ориентироваться на широкую функциональность системы и качество разработки, проактивное реагирование на возможные проблемы.

Список литературы

1. Кравченко Д.А. Микросервисная архитектура // Интерактивная наука. 2022. № 4 (69). С. 43-44.

2. Пахмурина Е.А. Необходимость перехода от монолитной к микросервисной архитектуре при разработке и внедрении bss-решений // Теория и практика современной науки. 2020. № 6 (60). С. 277-279.

3. Сабиров Д.А. Микросервисная архитектура на frontend // Научный журнал. 2021. № 7 (62). С. 15-27.

4. Шитько А.М. Проектирование микросервисной архитектуры программного обеспечения // Труды БГТУ. Серия: Физико-математические науки и информатика. 2017. № 9 (200). С. 122-125.

5. Косов С.Г., Кальянов Л.В. Разработка и тестирование приложений с микросервисной архитектурой // Форум молодых ученых. 2019. № 4 (32). С. 567-571.

6. Кучеренко Н.Ю. Проблема целостности данных в микросервисной архитектуре // Столыпинский вестник. 2022. № 4. С. 1974-1983.

7. Осипов Д.Б. Проектирование программного обеспечения с помощью микросервисной архитектуры // Вестник науки и образования. 2018. № 5 (41). С. 41-46.

8. Ворсин В.А. Микро-сервисная архитектура бизнес-приложений – перспективы и проблемы // Глобус. 2020. № 4 (50). С. 50-52.

9. Джалалов М.Э. Стратегии управления версионностью api в микросервисной архитектуре // Экономика и качество систем связи. 2024. № 1 (31). С. 136-143.