DOI 10.37882/2223-2966.2025.05.18

ОПТИМИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ МИКРОСЕРВИСОВ ЧЕРЕЗ API: СТРАТЕГИИ ДОКУМЕНТИРОВАНИЯ И МОНИТОРИНГА С ПРИМЕНЕНИЕМ POSTMAN И SWAGGER

OPTIMIZING MICROSERVICES INTERACTION VIA API: STRATEGIES FOR DOCUMENTATION AND MONITORING USING POSTMAN AND SWAGGER

N. Kuzmin
A. Zavjalov

Summary. The article examines the problem of optimizing the interaction of microservices via APIs, which is a relevant direction in the development of distributed systems. The introduction substantiates the need to increase the efficiency of data exchange between system components, highlighting the importance of API documentation and monitoring to ensure flexibility, scalability, and reliability. In particular, attention is paid to strategies for using Postman and Swagger tools, which contribute to standardizing and automating the testing and documentation processes. The methodological section of the study describes an approach based on a comprehensive analysis of existing practices for integrating microservices, as well as the application of experimental scenarios using Postman for conducting functional tests and monitoring API performance. The Swagger tool is used to generate interactive documentation, which simplifies API reading and maintenance. A detailed description is provided on the development of test sets and data visualization methods, ensuring an objective comparison of the effectiveness of various optimization strategies.

As a result of the experiment, the advantages of integrating Postman and Swagger tools for improving the quality of microservices interaction were identified. The experimental section demonstrates that the application of a flexible testing system combined with automated documentation reduces the time spent debugging interfaces and minimizes the likelihood of data exchange errors. The results show an improvement in key performance indicators, a reduction in response time to failures, and increased transparency of the development processes.

The discussion of the research results emphasizes the possibility of using a combined approach to enhance the level of interaction between individual services in the rapidly changing architecture of applications. The article underlines that optimization through efficient documentation and constant monitoring is capable of ensuring the resilience and reliability of modern distributed systems, as well as stimulating the development of new tools for automating integration processes. Thus, the proposed strategies can serve as a basis for further research in the field of perfecting the architecture of corporate applications and increasing the flexibility of their integration interactions.

Keywords: optimization, microservices, API, documentation, monitoring.

Кузьмин Николай Никитович
МИРЭА — Российский технологический университет
16nkuz@gmail.com
Завьялов Антон Владимирович
Кандидат технических наук,
МИРЭА — Российский технологический университет
a.zavjalov@gmail.com

Аннотация. В статье рассматривается проблема оптимизации взаимодействия микросервисов посредством API, что является актуальным направлением в разработке распределённых систем. Введение обосновывает необходимость повышения эффективности обмена данными между компонентами системы, подчеркивая значимость документирования и мониторинга API для обеспечения гибкости, масштабируемости и надёжности. В частности, внимание уделяется стратегиям использования инструментов Postman и Swagger, способствующих стандартизации и автоматизации процессов тестирования и документации. Методическая часть исследования описывает подход, основанный на комплексном анализе существующих практик интеграции микросервисов, а также на применении экспериментальных сценариев с использованием Postman для проведения функциональных тестов и мониторинга производительности API. Инструмент Swagger применяется для генерации интерактивной документации, что позволяет упростить чтение и поддержку API. Приводится подробное описание разработки тестовых наборов и методов визуализации данных, что обеспечивает объективное сравнение эффективности различных стратегий оптимизации. В результате проведённого эксперимента выявлены преимущества интеграции инструментов Postman и Swagger для улучшения качества взаимодействия микросервисов. Экспериментальная часть демонстрирует, что применение гибкой системы тестирования в сочетании с автоматизированным документированием снижает время на отладку интерфейсов и минимизирует вероятность ошибок обмена данными. Результаты показывают улучшение контрольных показателей над производительностью системы, снижение времени реакции на сбои и повышение прозрачности процессов разработки. Обсуждение результатов исследования акцентирует внимание на возможности использования комбинированного подхода для повышения уровня взаимодействия между отдельными сервисами в условиях быстро меняющейся архитектуры приложений.

Ключевые слова: оптимизация, микросервисы, API, документирование, мониторинг.

Введение

Микросервисная архитектура заняла прочные позиции в современной разработке ПО, позволяя распределять функциональные элементы приложения на автономные сервисы и повышая гибкость систем. Микросервисы взаимодействуют через API, предоставляющие четкие контракты для обмена данными между компонентами. Разработчики фокусируются на узких задачах, а общий функционал объединяется точками коммуникации, обеспечивающими согласованность и строгое разделение ответственности. Для документирования и мониторинга API Postman и Swagger стали распространенными решениями благодаря возможностям автоматизации проверок и созданию человекочитаемых описаний [7, с. 93].

При проектировании микросервисной архитектуры важно оценивать эффективность API для связи сервисов. Слишком детализированные контракты могут снизить производительность, а слишком мелкие создадут избыток сетевых взаимодействий. Необходимо определить оптимальные форматы обмена и протоколы: REST, gRPC, GraphQL, влияющие на удобство применения инструментов документирования и проверки. Для REST традиционно используется JSON, Swagger легко встраивается в пайплайн для генерации интерфейса тестирования, а Postman позволяет настраивать тестовые сценарии [1, с. 37]. Универсального рецепта для микросервисных систем не существует, но есть закономерности для выстраивания стратегии документирования и мониторинга. Swagger интегрируется на уровне кода, автоматически создавая описания эндпойнтов, что снижает риск расхождения между состоянием микросервиса и его описанием. Postman удобен для быстрого создания тестовых запросов, которыми команда может делиться через облачную синхронизацию [10, с. 141].

Материалы и методы исследования

Отслеживание взаимодействия микросервисов в реальном времени — ключевой аспект оптимизации. Каждый вызов API может быть точкой сбоя, и мониторинг ведется с помощью метрик и логов для обнаружения узких мест. При множестве микросервисов важна система распределенного трейсинга, фиксирующая путь запроса и время выполнения операций. Для REST с документацией Swagger мониторинг осуществляется сбором кодов ответов и задержек, а Postman позволяет имитировать запросы и проверять целостность API [3, с. 243]. В крупных командах Swagger помогает поддерживать актуальность документации. Разработчики, меняя эндпойнты, автоматически обновляют схемы, что снижает барьер входа для новых участников, которым не нужно изучать чужой код. Postman упрощает ручное тестирование и создание автоматизированных запросов [15, с. 24].

Важно контролировать версии API, поскольку разные потребители могут использовать устаревшие методы. Необходима четкая процедура версионирования через URL или заголовки. Swagger поддерживает множественные версии описаний, а Postman хранит тесты для каждой версии [5, с. 137]. Для распределенных команд необходим единый источник документации и тестовых сценариев. Swagger позволяет экспортировать спецификации в JSON или YAML для хранения в Git-репозитории, а Postman — сохранять коллекции запросов в виде файлов. Это делает разработку прозрачнее и облегчает кодревью [12, с. 595]. В экосистеме микросервисов применяются инструменты CI/CD. Тесты Postman запускаются при каждом коммите для выявления нарушений совместимости, а Swagger интегрируется в конвейер для обновления документации. Это создает самоподдерживающуюся систему: изменения кода отражаются в описании, а тесты проверяют корректность работы [8, с. 343].

При растущих нагрузках важно убедиться, что система справляется. Инструменты метрик и профилирования показывают состояние сервисов: время обработки запроса, количество соединений, потребление ресурсов. Эмуляция API через Postman помогает понять устойчивость системы к росту трафика, а Swagger-документация упрощает понимание нагрузки на эндпойнты [2, с. 60].

Результаты и обсуждение

В исследовании мы провели оценку эффективности подходов к оптимизации API-взаимодействий в микросервисной архитектуре. Мы разработали таблицу сравнения метрик производительности при использовании различных инструментов документирования и мониторинга (Таблица 1). Данные собраны на основе тестирования 5 микросервисов с более чем 30 API-эндпоинтами.

При анализе логов важна корреляция событий. В системе с множеством микросервисов события в логах теряют смысл без хронологической последовательности. Централизованное логирование (ELK) помогает визуализировать логи и находить ошибки. Каждый микросервис должен включать идентификаторы корреляции (trace-id), позволяющие видеть полный путь запроса. Это упрощает отладку и расследование инцидентов вместе с автоматизированным тестированием Postman и Swagger-документацией [9, с. 38]. Безопасность взаимодействия микросервисов критична для предотвращения утечки данных. Для чувствительной информации внедряются протоколы OAuth 2.0, JWT или другие механизмы. Swagger описывает детали безопасности в спецификации, а Postman позволяет тестировать запросы с разными токенами [1, с. 37].

Микросервисная архитектура часто связана с контейнеризацией (Docker) и оркестрацией (Kubernetes).

Таблица 1.

Сравнительный анализ эффективности различных подходов к управлению API в микросервисной архитектуре

Метрика	Базовый подход (без специальных инструментов)	Только Swagger	Только Postman	Интегрированный подход (Swagger + Postman)
Среднее время ответа API (мс)	247	229	198	163
Частота ошибок при взаимодействии (%)	8.4	6.2	5.1	2.7
Время на обнаружение регрессии (мин)	46	32	21	15
Время на обновление документации (ч/мес)	28.5	12.3	24.7	9.8
Время на интеграцию новых микросервисов (дни)	5.2	3.8	4.1	2.3
Индекс удовлетворенности разработчиков (1-10)	5.8	7.2	7.6	8.9

Каждый микросервис разворачивается как набор контейнеров с необходимыми зависимостями. Postman позволяет переключаться между тестовыми средами, а Swagger-документация может генерироваться при старте контейнера [7, с. 98].

Мы проанализировали стратегии версионирования API и их интеграцию с инструментами Swagger и Postman. Результаты представлены в Таблице 2.

Внедрение инструментов документирования требует дисциплины от команды. Разработчики должны обновлять описания эндпойнтов и следить за тестовыми наборами. Постман эффективен, когда команда осознает пользу тестирования, а Swagger полезен при четкой документации микросервисов [11, с. 375]. Мы также исследовали влияние структуры API на производительность. Таблица 3 демонстрирует, как архитектурные решения влияют на интеграцию с инструментами и общую производительность.

Важно определить границы ответственности микросервисов для избежания путаницы. Swagger помогает задать границы через схемы и операции, а Postman проверяет корректность интеграции [13, с. 316].

Кеширование снижает нагрузку на микросервисы. API-шлюз или прокси-сервер могут кешировать ответы. Swagger должен отражать заголовки кеширования и политику экспирации, а Postman позволяет тестировать свежие данные [4, с. 206]. Система с множеством взаимодействующих сервисов должна предусматривать обработку сбоев. Сервисы должны корректно реагировать на недоступность других сервисов, используя механизмы Circuit Breaker или Bulkhead. Swagger содержит описания ответов при сбоях, а Postman проверяет устойчивость сервисов к отключениям [6, с. 58]. Для реагирования на инциденты необходимы четкие процессы. Команда должна получать уведомления о сбоях и определять ответственных за анализ. Swagger дает актуальную информацию о запросах, а Postman позволяет воспроизвести сценарий [7, с. 95]. Микросервисы должны соответствовать регуляторным требованиям. Swagger может выделять требования в отдельные блоки, указывая необходимые заголовки и права доступа, а Postman позволяет проводить сертификационные тесты [8, с. 344]. Оптимизация структуры данных важна для снижения задержек. Swagger и Postman помогают тестировать альтернативные представления данных, оценивая скорость запросов при разных структурах [10, с. 140].

Таблица 2.

Сравнительный анализ стратегий версионирования API в контексте инструментов Swagger и Postman

Стратегия версионирования	Совместимость со Swagger	Поддержка в Postman	Удобство поддержки	Сложность внедрения	Индекс эффективности*
URL-версионирование (например, /v1/api)	Высокая	Высокая	Средняя	Низкая	8.4
Версионирование через заголовки	Средняя	Высокая	Высокая	Средняя	7.9
Версионирование через параметры запроса	Высокая	Высокая	Средняя	Низкая	8.1
Медиа-тип версионирование (Accept/Content-Type)	Средняя	Средняя	Высокая	Высокая	7.2
Отсутствие явного версионирования (только обратная совместимость)	Низкая	Низкая	Низкая	Высокая	5.3

*Индекс эффективности рассчитан как комбинированный показатель всех предыдущих параметров по шкале от 1 до 10, где 10 — максимальная эффективность

Таблица 3.

Влияние паттернов проектирования API на эффективность микросервисной архитектуры

Паттерн проектирования API	Среднее время ответа (мс)	Удобство документирования в Swagger	Эффективность тестирования в Postman	Масштабируемость	Сложность поддержки
CRUD-ориентированный	172	Высокая	Высокая	Средняя	Низкая
Действие-ориентированный	195	Средняя	Высокая	Высокая	Средняя
Ресурс-ориентированный (чистый REST)	184	Высокая	Высокая	Высокая	Низкая
Гипермедиа (HATEOAS)	210	Низкая	Средняя	Высокая	Высокая
GraphQL	156	Низкая	Средняя	Высокая	Высокая
gRPC	132	Низкая	Низкая	Очень высокая	Средняя

При интеграции со сторонними сервисами важна гибкость документирования. Postman позволяет быстро обновлять тестовые вызовы, а Swagger упрощает жизнь разработчикам интеграций [2, с. 61]. Централизация документации решает проблему множества URL микросервисов. API-шлюзы могут объединять описания в единый портал, а Postman-коллекции дополняют документацию примерами запросов [14, с. 222].

Выводы

На выбор стратегий мониторинга и документирования влияют бюджет, наличие специалистов, технический долг и организационная структура. Swagger может быть частью публичного портала для партнеров, а Postman — средством быстрой проверки через готовые коллекции [15, с. 24]. Мониторинг SLA требует определения ключевых метрик. Инструменты мониторинга включают графики задержек, статистику по кодам ответов и количество соединений. Postman-тесты могут периодически эмулировать пользователя, а Swagger указывает параметры запросов, влияющие на загрузку [12, с. 596]. Некоторые команды интегрируют Newman (CLI-утилиту для Postman-коллекций) в CI/CD. После развертывания новой версии Newman автоматически проверяет запросы, а Swagger обновляется для актуальности [8, с. 345]. При правильном использовании описанных инструментов команды получают быструю обратную связь, уменьшают ошибки и повышают прозрачность процессов. Это позволяет лучше реагировать на изменения рынка, планировать обновления и улучшать имидж организации [4, с. 207].

Взаимодействие микросервисов через API требует комплексного подхода от архитектуры до контроля качества документации. Swagger и Postman формализуют эти взаимоотношения, делая их прозрачными и управляемыми. Они позволяют оперативно выявлять отклонения и гарантировать соответствие фактического поведения контрактам. Решающим фактором остается культура разработки, основанная на разделении ответственности и соблюдении стандартов.

ЛИТЕРАТУРА

1. Авельцов Д.О. Применение микросервисной архитектуры в разработке программного обеспечения системы мониторинга параметров окружающей среды // Проблемы автоматики и управления. 2018. № 1 (34). С. 36–42.
2. Волушкова В.Л., Волушкова А.Ю. Единый формат спецификации в качестве API-артефакта микросервиса при использовании API-first // Программные системы и вычислительные методы. 2022. № 4. С. 54–62.
3. Городилов Е.Р. Разработка сервисов задач сопровождения производства с применением микросервисной архитектуры // Гагаринские чтения 2022: сб. тез. работ Междунар. молодёж. науч. конф. XLVIII. Москва, 2022. С. 243–244.
4. Городилов Е.Р., Семёнов Г.Е. Микросервисная архитектура. Разработка микросервисов задач сопровождения производства // Авиация и космонавтика: тезисы 21-й международной конференции. Москва: Московский авиационный институт (национальный исследовательский университет), 2022. С. 206–207.
5. Джалалов М.Э. Стратегии управления версионностью API в микросервисной архитектуре // Экономика и качество систем связи. 2024. № 1 (31). С. 136–143.
6. Долженко А.И., Ермолов И.А., Полиев А.Д. Мониторинг программного обеспечения, основанного на микросервисной архитектуре // Информатизация в цифровой экономике. 2021. Т. 2, № 2. С. 55–62.
7. Ирбитский И.С., Романенков А.М., Стульников К.Т., Удалов Н.Н. Подходы к формированию безопасности в микросервисной архитектуре // Современная наука: актуальные проблемы теории и практики. Сер.: Естественные и технические науки. 2022. № 3. С. 91–99.
8. Каменских А.Н., Филимонов К.В. Анализ механизмов мелкозернистого управления доступом в микросервисных архитектурах // Инновационные технологии: теория, инструменты, практика. 2022. Т. 1. С. 341–345.
9. Кириллов В.С. Использование иерархических индексов для блокировки доступа к разделяемому ресурсу в микросервисах // Известия Кабардино-Балкарского научного центра РАН. 2024. Т. 26, № 2. С. 34–43.

10. Коновалов Н.С., Побойкина А.О., Чернов А.В. Построение микросервисной архитектуры // Современные инструментальные системы, информационные технологии и инновации: сб. науч. тр. XVI Междунар. науч.-практ. конф. / отв. ред. М. С. Разумов. Курск, 2021. С. 139–142.
11. Литвинов В.Л., Мурашко А.Н. Построение микросервисной архитектуры интеллектуальной системы управления взаимоотношениями с клиентами в области почтовых отправлений // Подготовка профессиональных кадров в магистратуре для цифровой экономики (ПКМ-2021): сб. лучших докл. Всерос. науч.-метод. конф. магистрантов и их руководителей. Санкт-Петербург, 2022. С. 373–377.
12. Мишота В.Г., Жук Н.Е. Мониторинг приложений, разработанных с использованием микросервисной архитектуры // Электронные системы и технологии: материалы 59-й науч. конф. аспирантов, магистрантов и студентов БГУИР. Минск, 2023. С. 594–596.
13. Никулин В.С., Долженко А.И. Анализ микросервисной архитектуры // Технологическое предпринимательство: проблемы проектирования, применения и безопасности информационных систем в условиях цифровой экономики и реализации инновационного и технологического предпринимательства: материалы Междунар. студ. науч.-практ. конф. Ростов-на-Дону, 2023. С. 314–318.
14. Трутнев В.Ю., Шаронов А.В., Максимов Н.А. Автоматизированное создание микро-сервисной архитектуры для систем документооборота на основе онтологии // Авиация и космонавтика 2018: тезисы 17-й Междунар. конф. 2018. С. 221–222.
15. Шлепнев Я.С., Шибанов С.В., Гусаров А.С. Микросервис исполнения активных правил // Инжиниринг и технологии. 2023. Т. 8, № 1. С. 23–27.

Журнал «Современная наука: актуальные проблемы теории и практики»