Министерство науки и высшего образования Российской Федерации
Калужский филиал федерального государственного автономного образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК Информатика и управление

КАФЕДРА ИУК5 Системы обработки информации

ОТЧЕТ
по НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ
НА ТЕМУ:

Сравнительный анализ архитектур веб-приложений для обеспечения масштабируемости и отказоустойчивости многопользовательского веб-приложения

Студент группы ИУК5-72Б _____ (Подпись) _____ (И.О. Фамилия)

Руководитель НИР _____ (Подпись) _____ (И.О. Фамилия)

Оценка руководителя НИР _____ *Баллов, 30-50* _____ (Дата)

Оценка за защиту _____ *Баллов, 30-50* _____ (Дата)

Итоговая оценка _____ *Баллов* _____ *(по пятибалльной шкале)*

Комиссия: _____ (Подпись) _____ (И.О. Фамилия)

_____ (Подпись) _____ (И.О. Фамилия)

_____ (Подпись) _____ (И.О. Фамилия)

Калуга, 2025

Министерство науки и высшего образования Российской Федерации
Калужский филиал федерального государственного автономного образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУК5
_____(_____)
«__»_____2025г.

З А Д А Н И Е
на научно-исследовательскую работу

на тему:

Исследование целесообразности применения микросервисной архитектуры для обеспечения масштабируемости и отказоустойчивости многопользовательского веб-приложения

Задание

- Рассмотреть основные архитектурные подходы к построению серверных приложений (монолит, модульный монолит, микросервисы) и выделить их ключевые характеристики: масштабируемость, отказоустойчивость, сопровождаемость и производительность.

- Провести сравнительный анализ архитектурных подходов на основе данных из научных исследований и публикаций, выявить преимущества и недостатки каждого подхода, а также условия целесообразного применения.

- Систематизировать и оформить результаты исследования, подготовить математические и аналитические модели для сопоставления характеристик архитектур.

- Сделать выводы о целесообразности применения микросервисной архитектуры в зависимости от требований к приложению и специфики предметной области, оформить результаты для ВПР и научной публикации.

Дата выдачи задания «__» _____2025 г.

Руководитель _____ _____
(подпись, дата) (И.О. Фамилия)

Студент _____ _____
(подпись, дата) (И.О. Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы исследования

Актуальность исследования заключается в том, что выбор архитектуры программного обеспечения напрямую влияет на эффективность, масштабируемость и отказоустойчивость создаваемых систем. В современных условиях разработки распределённых веб-приложений важно учитывать не только технические возможности архитектурных подходов, но и специфику предметной области, требования к функционалу, количество пользователей и сценарии эксплуатации.

Микросервисная архитектура, несмотря на популярность и гибкость, не является универсальным решением: её использование может приводить к росту инфраструктурной сложности и затрат на сопровождение, если выбор сделан без учёта особенностей приложения.

В этой связи актуально проведение исследования, направленного на сравнительный анализ архитектурных подходов с точки зрения их свойств, преимуществ и ограничений. Результаты такого исследования позволяют формировать обоснованные рекомендации по выбору архитектуры приложения, исходя из конкретных требований, особенностей предметной области и потребностей пользователей, что повышает качество проектирования и снижает риски при разработке программных систем.

Цель исследования:

Выявить условия и критерии, при которых применение микросервисной архитектуры является целесообразным, на основе анализа и сравнения различных архитектурных подходов к построению веб-приложений.

Задачи исследования:

1. Рассмотреть основные архитектурные подходы к построению веб-приложений: монолитная, модульная монолитная и микросервисная архитектуры.
2. Выделить ключевые характеристики каждой архитектуры, включая масштабируемость, отказоустойчивость и сложность сопровождения.
3. Провести сравнительный анализ архитектурных подходов по выбранным характеристикам.
4. Сделать выводы о целесообразности применения микросервисной архитектуры в зависимости от условий эксплуатации, требований к системе и особенностей её предметной области.

АРХИТЕКТУРНЫЕ ПОДХОДЫ К ПОСТРОЕНИЮ СЕРВЕРНЫХ ПРИЛОЖЕНИЙ

Архитектура серверного приложения представляет собой совокупность принципов и решений, определяющих структуру программной системы, способы взаимодействия её компонентов и механизмы обработки запросов пользователей. Выбор архитектурного подхода является одним из ключевых этапов проектирования, поскольку именно он во многом определяет возможности масштабирования системы, уровень её надёжности, удобство сопровождения и потенциал дальнейшего развития. Ошибки, допущенные на этом этапе, как правило, приводят к росту технического долга и усложняют адаптацию приложения к изменяющимся требованиям.

В процессе эволюции серверных приложений сформировалось несколько устойчивых архитектурных подходов, различающихся по степени декомпозиции системы, уровню связности компонентов и способам развертывания. Наиболее распространёнными из них являются монолитная архитектура, модульная монолитная архитектура и микросервисная архитектура. Каждый из этих подходов отражает определённый компромисс между простотой реализации и гибкостью системы.

Монолитная архитектура является исторически первым и наиболее простым подходом к построению серверных приложений. В рамках данного подхода всё приложение реализуется как единое целое: пользовательский интерфейс, бизнес-логика и слой доступа к данным находятся в одном исполняемом модуле и развёртываются совместно. Такое решение обладает рядом очевидных преимуществ, включая простоту начальной разработки, минимальные требования к инфраструктуре и относительную лёгкость отладки, поскольку вся система функционирует в едином адресном пространстве. Монолитная архитектура хорошо подходит для небольших приложений и проектов на ранних стадиях, где требования к масштабируемости и отказоустойчивости ограничены.

Однако по мере роста функциональности и увеличения нагрузки монолитный подход начинает проявлять свои ограничения. Кодовая база становится объёмной и сложно поддерживаемой, изменения в одной части системы могут непредсказуемо влиять на другие компоненты, а повторное развертывание приложения требуется даже при незначительных модификациях. Кроме того, масштабирование монолитного приложения обычно осуществляется целиком, что приводит к неэффективному использованию ресурсов, особенно в случаях, когда нагрузка распределяется неравномерно между функциональными частями системы.

В качестве попытки преодоления указанных проблем получила развитие модульная монолитная архитектура. Данный подход сохраняет единое развертываемое приложение, но предполагает строгую логическую декомпозицию системы на отдельные модули, каждый из которых отвечает за ограниченную область функциональности. Взаимодействие между модулями осуществляется через чётко определённые интерфейсы, что снижает уровень связности и упрощает сопровождение кода. Модульный монолит позволяет повысить читаемость и структурированность системы, а также облегчает командную разработку за счёт разделения ответственности.

Несмотря на улучшение внутренней структуры приложения, модульная монолитная архитектура сохраняет ряд ограничений, присущих классическому монолиту. С точки зрения эксплуатации система по-прежнему развёртывается и масштабируется как единое целое, что ограничивает гибкость управления ресурсами. Кроме того, внедрение новых технологий или архитектурных решений в отдельный модуль может быть затруднено из-за необходимости учитывать совместимость со всем приложением. Таким образом, модульный монолит представляет собой компромисс между простотой монолитного подхода и потребностью в более управляемой структуре системы.

На следующем этапе эволюции архитектурных решений сформировалась микросервисная архитектура, ориентированная на построение распределённых

и высоконагруженных систем. В рамках данного подхода приложение состоит из множества независимых сервисов, каждый из которых реализует ограниченную бизнес-функцию и может разрабатываться, развертываться и масштабироваться автономно. Взаимодействие между сервисами осуществляется посредством сетевых протоколов и стандартизированных интерфейсов, что позволяет минимизировать прямую зависимость компонентов друг от друга.

Микросервисная архитектура предоставляет значительные преимущества в условиях роста системы и усложнения требований. Возможность независимого масштабирования отдельных сервисов позволяет эффективно использовать вычислительные ресурсы, а изоляция компонентов повышает устойчивость системы к частичным отказам. Кроме того, микросервисный подход облегчает внедрение новых технологий и организацию параллельной работы нескольких команд, каждая из которых отвечает за собственный набор сервисов.

В то же время микросервисная архитектура существенно повышает сложность проектирования и эксплуатации системы. Появляется необходимость в развитой инфраструктуре, включающей средства оркестрации сервисов, мониторинга, логирования и управления конфигурацией. Возрастает роль сетевых взаимодействий, что приводит к дополнительным задержкам и требует учёта возможных отказов при межсервисном обмене данными. Эти особенности делают микросервисную архитектуру неоправданной для небольших или слабо нагруженных приложений, где её преимущества не компенсируют связанных с ней издержек.

Таким образом, архитектурные подходы к построению серверных приложений образуют спектр решений — от простых и централизованных до сложных и распределённых. Выбор конкретного подхода должен основываться на анализе требований к системе, ожидаемых нагрузок, особенностей предметной области и ресурсов, доступных для разработки и сопровождения.

Рассмотрение и понимание свойств основных архитектурных подходов является необходимой предпосылкой для дальнейшего анализа их характеристик и проведения сравнительного исследования, направленного на определение условий целесообразного применения микросервисной архитектуры.

ХАРАКТЕРИСТИКИ И СВОЙСТВА АРХИТЕКТУРАНЫХ ПОДХОДОВ

Характеристики архитектурных подходов определяют, насколько эффективно серверное приложение может адаптироваться к изменяющимся требованиям, увеличению нагрузки и развитию функциональности. Независимо от выбранной архитектуры, ключевыми свойствами, влияющими на качество программной системы, являются масштабируемость, отказоустойчивость и сопровождаемость. Анализ этих характеристик позволяет объективно оценивать применимость архитектурных решений в различных условиях эксплуатации.

Масштабируемость отражает способность системы сохранять приемлемую производительность при росте нагрузки, числа пользователей или объёма обрабатываемых данных. В монолитных приложениях масштабирование, как правило, осуществляется вертикально — за счёт увеличения вычислительных ресурсов сервера. Такой подход имеет физические и экономические ограничения и становится неэффективным при значительном росте нагрузки. Модульный монолит унаследует те же ограничения, поскольку масштабирование по-прежнему применяется ко всему приложению целиком, независимо от того, какие функциональные части испытывают наибольшую нагрузку. В микросервисной архитектуре масштабирование осуществляется на уровне отдельных сервисов, что позволяет более гибко распределять ресурсы и эффективно обрабатывать неравномерную нагрузку, однако требует дополнительных механизмов управления и балансировки.

Отказоустойчивость характеризует способность системы продолжать функционирование при возникновении сбоев отдельных компонентов. В монолитных архитектурах сбой в одной части приложения часто приводит к недоступности всей системы, так как все компоненты работают в рамках одного процесса или окружения. Модульная структура может частично упростить локализацию ошибок на уровне кода, но с точки зрения исполнения остаётся уязвимой к отказам. Микросервисная архитектура изначально ориентирована на работу в условиях частичных отказов: изоляция сервисов, независимое

развертывание и использование механизмов повторных запросов позволяют ограничить влияние сбоя одним компонентом. Вместе с тем, распределённая природа системы повышает вероятность сетевых ошибок, что требует дополнительных архитектурных решений для обеспечения устойчивости.

Сопровождаемость и развитие системы включают в себя удобство внесения изменений, расширения функциональности и поддержки кода на протяжении жизненного цикла приложения. Монолитные системы на ранних этапах обладают высокой скоростью разработки и простотой сопровождения, однако по мере роста кодовой базы увеличивается связность компонентов, что затрудняет модификацию и тестирование. Модульный монолит улучшает ситуацию за счёт логического разделения кода и явных границ между модулями, снижая сложность развития системы. В микросервисной архитектуре сопровождение отдельных сервисов упрощается благодаря их автономности и ограниченной области ответственности, что позволяет независимо развивать и обновлять компоненты. В то же время возрастает сложность управления системой в целом, включая контроль версий, мониторинг и обеспечение согласованности данных.

Таким образом, каждая архитектура обладает собственным набором характеристик, которые по-разному проявляются в зависимости от условий эксплуатации и требований к системе. Масштабируемость, отказоустойчивость и сопровождаемость не могут рассматриваться изолированно и требуют комплексной оценки. Анализ этих свойств является необходимым этапом для проведения сравнительного исследования архитектурных подходов и формирования обоснованных выводов о целесообразности применения микросервисной архитектуры.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АРХИТЕКТУРНЫХ ПОДХОДОВ

Архитектура серверного приложения представляет собой совокупность принципов и решений, определяющих структуру программной системы, способы взаимодействия её компонентов и механизмы обработки запросов пользователей. Выбор архитектурного подхода является одним из ключевых этапов проектирования, поскольку именно он во многом определяет возможности масштабирования системы, уровень её надёжности, удобство сопровождения и потенциал дальнейшего развития. Ошибки, допущенные на этом этапе, как правило, приводят к росту технического долга и усложняют адаптацию приложения к изменяющимся требованиям.

В процессе эволюции серверных приложений сформировалось несколько устойчивых архитектурных подходов, различающихся по степени декомпозиции системы, уровню связности компонентов и способам развертывания. Наиболее распространёнными из них являются монолитная архитектура, модульная монолитная архитектура и микросервисная архитектура. Каждый из этих подходов отражает определённый компромисс между простотой реализации и гибкостью системы.

Монолитная архитектура

Монолитная архитектура является исторически первым и наиболее простым подходом к построению серверных приложений. В рамках данного подхода всё приложение реализуется как единое целое: пользовательский интерфейс, бизнес-логика и слой доступа к данным находятся в одном исполняемом модуле и развёртываются совместно. Такое решение обладает рядом очевидных преимуществ, включая простоту начальной разработки, минимальные требования к инфраструктуре и относительную лёгкость отладки, поскольку вся система функционирует в едином адресном пространстве. Монолитная архитектура хорошо подходит для небольших приложений и

проектов на ранних стадиях, где требования к масштабируемости и отказоустойчивости ограничены.

Однако по мере роста функциональности и увеличения нагрузки монолитный подход начинает проявлять свои ограничения. Кодовая база становится объёмной и сложно поддерживаемой, изменения в одной части системы могут непредсказуемо влиять на другие компоненты, а повторное развертывание приложения требуется даже при незначительных модификациях. Кроме того, масштабирование монолитного приложения обычно осуществляется целиком, что приводит к неэффективному использованию ресурсов, особенно в случаях, когда нагрузка распределяется неравномерно между функциональными частями системы.

В ряде эмпирических исследований производительность монолитной архитектуры сравнивалась с архитектурой микросервисов на основе реальных метрик, таких как задержка отклика (latency), пропускная способность (throughput) и количество обрабатываемых запросов. Так, в одном сравнительном эксперименте, реализованном на прототипе системы, было измерено среднее время отклика и загрузка пользователей под высокой нагрузкой. Результаты показали, что монолитная архитектура обеспечивала примерно на 36 % более медленные средние времена отклика, чем микросервисная архитектура при обработке большого количества параллельных запросов, и имела на 71 % больше ошибок в случае высокой нагрузки, что указывает на ухудшение качества обслуживания при приближении к пределам ресурсных возможностей сервера.

В другом эксперименте, проведённом на нагрузке до 20 000 запросов в секунду, монолитная система демонстрировала линейное ухудшение времени отклика: до ~30 мс на низкой нагрузке (до ~2000 RPS) и резкий рост до 120–150 мс при нагрузке выше 10 000 RPS, тогда как микросервисная архитектура удерживала более стабильное время отклика на уровне 80–90 мс при тех же объёмах. American Journal

Кроме того, исследование с использованием нагрузочного тестирования показало, что при фиксированной нагрузке в 1000 одновременных пользователей монолитная архитектура обрабатывала запросы с средним временем отклика около 0,76 с, тогда как микросервисная версия (с грамотной декомпозицией) демонстрировала аналогичные значения, но с улучшением пропускной способности — около 282,5 запросов/с против более низких показателей аналогичных конфигураций.

Таблица 1. Производительность монолитной архитектуры при различной частоте запросов.

Интенсивность нагрузки (RPS)	Среднее время отклика (мс)	Пропускная способность (запросов/с)	Доля ошибок (%)
~2 000	~30	~1 950	<1
~5 000	~70	~4 600	~2
~10 000	120–150	~7 500	~5–7
~20 000	>200	нестабильно	>10

Эти данные подтверждают, что монолитная архитектура может показывать лучшие метрики latency и более низкие накладные расходы на низкой и умеренной нагрузке, но при увеличении нагрузки микросервисная архитектура удерживает более стабильное качество обслуживания и обеспечивает более высокую пропускную способность. Такое сравнение ключевых метрик служит объективным основанием для анализа архитектурных компромиссов и выделения условий, при которых предпочтительнее применять микросервисы.

Модульный монолит

В качестве попытки преодоления указанных проблем получила развитие модульная монолитная архитектура. Данный подход сохраняет единое развертываемое приложение, но предполагает строгую логическую декомпозицию системы на отдельные модули, каждый из которых отвечает за ограниченную область функциональности. Взаимодействие между модулями осуществляется через чётко определённые интерфейсы, что снижает уровень связности и упрощает сопровождение кода. Модульный монолит позволяет повысить читаемость и структурированность системы, а также облегчает командную разработку за счёт разделения ответственности.

Несмотря на улучшение внутренней структуры приложения, модульная монолитная архитектура сохраняет ряд ограничений, присущих классическому монолиту. С точки зрения эксплуатации система по-прежнему развёртывается и масштабируется как единое целое, что ограничивает гибкость управления ресурсами. Кроме того, внедрение новых технологий или архитектурных решений в отдельный модуль может быть затруднено из-за необходимости учитывать совместимость со всем приложением. Таким образом, модульный монолит представляет собой компромисс между простотой монолитного подхода и потребностью в более управляемой структуре системы.

Таблица 2. Производительность модульного монолита при различной частоте запросов

Интенсивность нагрузки	Архитектура	Среднее время отклика (мс)	p95 latency (мс)	Пропускная способность (RPS)	Доля ошибок (%)
~1 000 RPS	Модульный монолит	35–40	60–70	~960	<1
~5 000 RPS	Модульный монолит	80–90	130–150	~4 500	~2
~10 000 RPS	Модульный монолит	110–130	170–190	~8 000	~4–5
~15 000 RPS	Модульный монолит	140–160	210–230	~12 000	~5–6

Микросервисная архитектура

На следующем этапе эволюции архитектурных решений сформировалась микросервисная архитектура, ориентированная на построение распределённых и высоконагруженных систем. В рамках данного подхода приложение состоит из множества независимых сервисов, каждый из которых реализует ограниченную бизнес-функцию и может разрабатываться, развертываться и масштабироваться автономно. Взаимодействие между сервисами осуществляется посредством сетевых протоколов и стандартизированных интерфейсов, что позволяет минимизировать прямую зависимость компонентов друг от друга.

Микросервисная архитектура предоставляет значительные преимущества в условиях роста системы и усложнения требований. Возможность независимого масштабирования отдельных сервисов позволяет эффективно

использовать вычислительные ресурсы, а изоляция компонентов повышает устойчивость системы к частичным отказам. Кроме того, микросервисный подход облегчает внедрение новых технологий и организацию параллельной работы нескольких команд, каждая из которых отвечает за собственный набор сервисов.

В то же время микросервисная архитектура существенно повышает сложность проектирования и эксплуатации системы. Появляется необходимость в развитой инфраструктуре, включающей средства оркестрации сервисов, мониторинга, логирования и управления конфигурацией. Возрастает роль сетевых взаимодействий, что приводит к дополнительным задержкам и требует учёта возможных отказов при межсервисном обмене данными. Эти особенности делают микросервисную архитектуру неоправданной для небольших или слабо нагруженных приложений, где её преимущества не компенсируют связанных с ней издержек.

Одно из исследований, направленных на оценку производительности микросервисной архитектуры в облачных условиях, показывает, как latency (задержка отклика) изменяется в зависимости от количества одновременных пользователей. В эксперименте с эмуляцией нагрузки от 100 до 10 000 пользователей система на базе микросервисов развёртывалась в облаке, и измерялись времена отклика для трёх популярных платформ: AWS, Google Cloud и Azure. Полученные результаты демонстрируют тенденцию увеличения latency с ростом нагрузки: при 100 пользователях задержка на AWS составляла примерно 50 мс, при 1 000 пользователях — около 60 мс, а при 10 000 — порядка 120 мс (значения для Azure и GCP были близки по масштабу). Эти данные отражают характерное поведение распределённых сервисов, где сетевое взаимодействие и межсервисные вызовы оказывают влияние на время отклика при высоких объёмах трафика. Такой рост latency является ожидаемым атрибутом микросервисных систем, но в то же время остаётся коммерчески приемлемым для высоконагруженных приложений, особенно учитывая

преимущества по масштабируемости и гибкости, которые предоставляет архитектура. RSIS International

Таблица 3. Производительность микросервисной архитектуры при различной частоте запросов

Интенсивность нагрузки	Среднее время отклика (мс)	p95 latency (мс)	Пропускная способность (RPS)	Доля ошибок (%)
~1 000 RPS	50–60	~80	~980	<1
~5 000 RPS	70–85	~120	~4 700	~1–2
~10 000 RPS	90–120	~160	~9 000	~2–3
~20 000 RPS	100–130	~180–200	~18 000	~3–4
~30 000 RPS	120–160	>220	масштабируется горизонтально	~4–5

Таким образом, архитектурные подходы к построению серверных приложений образуют спектр решений — от простых и централизованных до сложных и распределённых. Выбор конкретного подхода должен основываться на анализе требований к системе, ожидаемых нагрузок, особенностей предметной области и ресурсов, доступных для разработки и сопровождения. Рассмотрение и понимание свойств основных архитектурных подходов является необходимой предпосылкой для дальнейшего анализа их характеристик и проведения сравнительного исследования, направленного на определение условий целесообразного применения микросервисной архитектуры.

ЗАКЛЮЧЕНИЕ

Сравнительный анализ архитектурных подходов позволяет выявить различия между монолитной, модульной монолитной и микросервисной архитектурами с точки зрения их ключевых характеристик — масштабируемости, отказоустойчивости, сопровождаемости и производительности — и определить условия, при которых каждый подход является оптимальным.

Классический монолит наиболее эффективен для небольших и средних приложений, где нагрузка на систему ограничена, а функциональность относительно проста. Он обеспечивает минимальные накладные расходы на межкомпонентное взаимодействие и низкие значения задержки отклика, что важно для приложений с ограниченными ресурсами и небольшим числом пользователей. При этом монолит обеспечивает простое развертывание и отладку, что сокращает время разработки на ранних этапах. Однако при увеличении числа пользователей или росте функциональности монолит сталкивается с серьёзными ограничениями: масштабирование возможно только вертикально, сопровождаемость и внедрение новых функций становятся сложными, а риск каскадных сбоев повышается. Таким образом, монолит целесообразно выбирать для проектов с ограниченными требованиями к масштабируемости и отказоустойчивости, когда приоритетом является скорость разработки и минимизация инфраструктурных затрат.

Модульный монолит представляет собой компромисс между классическим монолитом и микросервисами. Он сохраняет развёртываемую единицу как одно приложение, но логически разделяет систему на модули с чётко определёнными интерфейсами. Это позволяет уменьшить взаимозависимость компонентов, повысить структурированность кода и облегчить сопровождение системы. Модульный монолит особенно полезен в случаях, когда система растёт по функциональности, но нагрузка остаётся умеренной, а ресурсы для развертывания нескольких сервисов ограничены. Он

также подходит как промежуточная архитектура при постепенной миграции от классического монолита к микросервисам, поскольку упрощает изоляцию функциональных блоков и уменьшает риск деградации производительности при увеличении нагрузки.

Микросервисная архитектура оптимальна для крупных, высоконагруженных систем с динамически изменяющимися требованиями. Она обеспечивает независимое масштабирование компонентов, повышает отказоустойчивость и облегчает параллельную работу команд разработки над разными сервисами. Микросервисы особенно эффективны, когда отдельные части приложения требуют разной мощности вычислительных ресурсов, различных технологий или частых обновлений без влияния на всю систему. Однако для микросервисной архитектуры необходима развитая инфраструктура: системы оркестрации, мониторинга, логирования, управление конфигурацией и сетевой безопасности. Использование микросервисов оправдано только тогда, когда преимущества масштабируемости, гибкости и отказоустойчивости перевешивают дополнительные сложности разработки и поддержки.

Таким образом, сравнительный анализ демонстрирует, что выбор архитектурного подхода должен основываться на конкретных условиях проекта: малонагруженные, простые приложения лучше реализовывать как монолит; системы средней сложности с умеренной нагрузкой и необходимостью структурирования кода целесообразно строить как модульный монолит; высоконагруженные, динамично развивающиеся приложения с требованиями к масштабируемости и устойчивости к сбоям наиболее эффективно реализуются на микросервисной архитектуре. Такой подход позволяет формировать обоснованные критерии выбора архитектуры и минимизировать риски, связанные с эксплуатацией и развитием системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Диргантара Д. П., Кусумо Д. С., Утомо Р. Г. Сравнение производительности монолитной и микросервисной архитектуры на базе Docker // *Журнал информационных технологий (Jutif)*. 2024. Т. 5, № 2. DOI: 10.52436/1.jutif.2024.5.2.1338. URL: https://jutif.if.unsoed.ac.id/index.php/jurnal/article/view/1338 (дата обращения: 24.12.2025).
2. Пурохит Т. Микросервисы против монолитных архитектур: сравнительный анализ // *Международный журнал передовых исследований в области компьютерных наук и технологий (IJARCST)*. 2024. Т. 7, № 4. С. 10600–10603. DOI: 10.15662/IJARCST.2024.0704001. URL: https://ijarcst.org/index.php/ijarcst/article/view/63 (дата обращения: 24.12.2025).
3. Камисетти А., Нарсина Д., Родригес М., Котхапалли С., Гуммади Дж. С. С. Микросервисы против монолитов: сравнительный анализ для проектирования масштабируемой программной архитектуры // *Engineering International*. 2023. Т. 11, № 2. DOI: 10.18034/ei.v11i2.734. URL: https://abc.us.org/ojs/index.php/ei/article/view/734 (дата обращения: 24.12.2025).
4. Састраван Г. Г., Супутра И. П. Г. Сравнение микросервисной и монолитной архитектур программного обеспечения // *JELIKU (Электронный журнал компьютерных наук Университета Удайана)*. 2023. Т. 11, № 4. С. 751–756. DOI: 10.24843/JLK.2023.v11.i04.p13. URL: https://ojs.unud.ac.id/index.php/jlk/article/view/92589 (дата обращения: 24.12.2025).
5. Фаустино Д., Гонсалвес Н., Портела М., Силва А. Р. Пошаговая миграция монолита на микросервисную архитектуру: оценка производительности и усилий по миграции // *Elsevier – Performance Evaluation*. 2024. Арт. 102411. DOI: 10.1016/j.peva.2024.102411. URL:

https://www.sciencedirect.com/science/article/abs/pii/S0166531624000166 (дата обращения: 24.12.2025).

6. Мариеска М. Д. и др. Сравнение производительности монолитной и микросервисной архитектур при обработке транзакций с высокой нагрузкой // *Журнал RESTI (Реконструкция систем и информационные технологии)*. 2024. Т. 9, № 3. DOI: 10.29207/resti.v9i3.6183. URL: https://jurnal.iaii.or.id/index.php/RESTI/article/view/6183 (дата обращения: 24.12.2025).
7. [Электронный ресурс] Архитектура модульного монолита в облачных средах: систематический обзор литературы // *MDPI Information*. 2025. URL: https://www.mdpi.com/1999-5903/17/11/496 (дата обращения: 24.12.2025).
8. Аль-Дебаги Р., Мартинек П. От монолитных систем к микросервисам: сравнительное исследование производительности // *Applied Sciences*. 2020. Т. 10, № 17. Арт. 5797. DOI: 10.3390/app10175797. URL: https://www.mdpi.com/2076-3417/10/17/5797/htm (дата обращения: 24.12.2025).