Загидуллин Рустам Мубаризович, бакалавр
Уфимский университет науки и технологий
Zagidullin Rustam Mubarizovich
Ufa University of Science and Technology

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АРХИТЕКТУР REST И GRAPHQL ПРИ РАЗРАБОТКЕ ВЕБ-ПРИЛОЖЕНИЙ
COMPARATIVE ANALYSIS OF REST AND GRAPHQL ARCHITECTURES IN WEB APPLICATION DEVELOPMENT

Аннотация. Рассмотрены архитектурные свойства REST и GraphQL применительно к разработке веб-приложений. Сопоставлены модели контракта, получение данных, кеширование, обработка ошибок, безопасность и эксплуатационная наблюдаемость. Предложены критерии выбора подхода с учетом структуры клиентских сценариев и требований инфраструктуры.

Abstract. The architectural properties of REST and GraphQL in web application development are reviewed. Contract models, data retrieval, caching, error handling, security, and operational observability are compared. Criteria for selecting an approach according to client scenarios and infrastructure requirements are proposed.

Ключевые слова: Веб-приложение, REST, GraphQL, программный интерфейс, HTTP, архитектура.

Keywords: Web application, REST, GraphQL, application programming interface, HTTP, architecture.

Введение

Программный интерфейс определяет границу между клиентом, серверной логикой и данными, поэтому его устройство влияет не только на формат сообщения, но и на развитие всей системы. В веб-разработке REST и GraphQL часто представляют как взаимозаменяемые способы построения API, хотя они описывают разные уровни архитектуры. REST является стилем распределенной гипермедийной системы и задает совокупность ограничений, тогда как GraphQL определяет язык запросов, типовую схему и правила исполнения операций [1, 3].

Цель статьи состоит в сравнении REST и GraphQL по критериям, значимым для современных веб-приложений: выразительность контракта, объем и число сетевых обращений, использование HTTP, кеширование, совместимость изменений, безопасность и эксплуатационный контроль. Сравнение носит архитектурно-аналитический характер и не приписывает одному подходу безусловного преимущества: решение зависит от структуры предметной области, разнообразия клиентов и требований к инфраструктуре.

Архитектурные основы REST и GraphQL

REST выводится из ограничений клиент-серверного взаимодействия, отсутствия состояния сеанса на сервере, кешируемости, единообразного интерфейса и слоистой системы. Их совместное применение направлено на масштабируемость взаимодействий, независимое развитие компонентов и участие посредников [1]. Ресурс идентифицируется URI, а клиент передает или получает его представление. Семантика методов, кодов состояния и полей HTTP описана единообразно и не зависит от внутреннего устройства сервиса [2].

Практический REST API обычно выражает предметную область через набор ресурсов и устойчивые адреса. GET читает представление, POST инициирует обработку или создает подчиненный ресурс, PUT и PATCH изменяют состояние, DELETE удаляет его. Безопасность и идемпотентность методов позволяют клиентам, шлюзам и кешам принимать обоснованные решения о повторе и сохранении ответа [2]. Однако соответствие HTTP-глаголам само по себе не гарантирует REST: важны также отсутствие скрытого состояния и последовательная семантика представлений.

GraphQL строит контракт вокруг типовой схемы. Клиент формирует документ с операцией query, mutation или subscription и перечисляет требуемые поля. Сервер проверяет документ относительно схемы, исполняет выборки и возвращает результат той же формы. Спецификация включает систему типов, фрагменты, переменные, интроспекцию, правила валидации и формат ответа, но не связывает реализацию с конкретным языком программирования или хранилищем [3].

Транспорт не входит в основную спецификацию GraphQL. Наиболее распространенное отображение на HTTP уточняется отдельным проектом спецификации, который описывает GET и POST, типы носителей, параметры запроса и коды состояния [4]. Статус проекта требует учитывать возможность изменений. Следовательно, сравнивать REST и GraphQL как два формата одного запроса недостаточно: первый организует взаимодействие вокруг ресурсов и стандартной семантики HTTP, второй предоставляет клиенту язык композиции данных поверх выбранного транспорта.

Сравнение контрактов, передачи данных и кеширования

В REST форма ответа определяется ресурсом, медиатипом и версией контракта. Если экран объединяет данные нескольких ресурсов, клиент может выполнить несколько запросов либо использовать специально подготовленное агрегирующее представление. GraphQL позволяет одному документу описать связанное дерево полей, что уменьшает расхождение между потребностями конкретного интерфейса и заранее заданным ответом. Но сетевой выигрыш не возникает автоматически: сложная операция может породить множество обращений к внутренним источникам и потребовать пакетной загрузки, ограничения параллелизма и анализа стоимости.

REST естественно использует HTTP-кеши. Ключ ответа как минимум связан с методом и целевым URI, а свежесть, валидаторы и директивы Cache-Control определяют допустимость повторного использования [5]. Публичные неизменяемые представления удобно распространять через браузерные и промежуточные кеши. Персонализированные ответы требуют корректного разделения ключей, заголовка Vary и запрета общего хранения там, где оно способно раскрыть данные.

У GraphQL многие операции передаются POST-запросом к одному адресу, поэтому стандартный HTTP-кеш не различает их по структуре документа. Для безопасных query-операций проект GraphQL over HTTP допускает GET, а прикладные системы применяют сохраненные запросы и клиентские нормализованные кеши [4]. Такой кеш требует устойчивых идентификаторов объектов и правил слияния. Они не задаются основной спецификацией, поэтому корректность инвалидирования становится частью архитектуры приложения, а не свойством протокола.

Эволюция REST-контракта обычно строится на добавлении полей, новых представлений и сохранении семантики существующих адресов. Несовместимое изменение может потребовать новой версии. В GraphQL единая схема явно показывает доступные и устаревающие поля, а статическая проверка документа обнаруживает нарушение контракта до исполнения [3]. При этом удаление поля остается несовместимым, а длительное накопление устаревших элементов усложняет схему. В обоих подходах необходимы каталог потребителей, период миграции и автоматическая проверка совместимости.

Безопасность, эксплуатация и критерии выбора

Авторизация должна выполняться на сервере независимо от гибкости клиента. В REST проверяется доступ к ресурсу и операции, в GraphQL – к типам, объектам и полям, включая вложенные резолверы. Произвольная глубина, большое число псевдонимов, пакетирование и дорогие поля могут увеличить вычислительную стоимость GraphQL-запроса. OWASP рекомендует валидацию входа, ограничения глубины и сложности, тайм-ауты, постраничную выдачу, контроль скорости и минимизацию диагностической информации [6].

Наблюдаемость REST удобно связывать с методом, шаблоном маршрута и кодом HTTP. Для GraphQL одного адреса недостаточно: в журнале и метриках фиксируются имя операции, результат валидации, стоимость, длительность резолверов и наличие частичных ошибок. Ответ GraphQL способен одновременно содержать data и errors, поэтому успешный транспортный статус не означает полного успеха бизнес-операции [3, 4]. Мониторинг должен анализировать как HTTP, так и прикладной результат.

REST рационален для ресурсных интерфейсов с устойчивыми представлениями, активным использованием CDN и стандартных средств HTTP, а также для публичных интеграций, где простота вызова важнее гибкости выборки. GraphQL полезен при большом числе неодинаковых клиентских экранов, быстро меняющейся композиции данных и необходимости единой типизированной точки доступа. Для малой предметной области его схема, резолверы, анализ стоимости и специальный кеш могут оказаться избыточными.

Выбор следует подтверждать на представительном наборе операций. Измеряются число сетевых обращений, объем полезной и избыточной передачи, время ответа, нагрузка внутренних источников, доля попаданий в кеш и диагностируемость отказов. Допустима гибридная архитектура: ресурсные операции и загрузка файлов сохраняют HTTP-ориентированный интерфейс, а агрегированные пользовательские экраны получают GraphQL-слой. Граница должна отражать ответственность сервисов, а не стремление использовать одну технологию повсеместно.

Заключение

REST и GraphQL решают пересекающиеся, но не идентичные задачи. REST формирует масштабируемое взаимодействие ресурсов через единообразную семантику HTTP и хорошо сочетается с посредниками и стандартным кешированием. GraphQL предоставляет типизированный контракт и управляемую клиентом форму результата, сокращая необходимость создавать отдельное представление для каждого интерфейса.

Архитектурное решение определяется характером данных и потребителей. При выборе учитываются не только удобство написания запроса, но и стоимость серверного исполнения, стратегия кеширования, совместимость схемы, авторизация и наблюдаемость. Наиболее устойчивым является подход, при котором контракт проверяется реальными сценариями, а REST и GraphQL применяются в тех границах, где их свойства дают измеримое преимущество

Список литературы:

1. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation. University of California, Irvine, 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (дата обращения: 16.08.2026).
2. Fielding R., Nottingham M., Reschke J. HTTP Semantics. RFC 9110. IETF, 2022. DOI: 10.17487/RFC9110.
3. GraphQL Foundation. GraphQL Specification. September 2025 Edition. URL: https://spec.graphql.org/September2025/ (дата обращения: 16.08.2026).
4. GraphQL Working Group. GraphQL over HTTP. Stage 2 Draft, 17.07.2026. URL: https://graphql.github.io/graphql-over-http/draft/ (дата обращения: 16.08.2026).
5. Fielding R., Nottingham M., Reschke J. HTTP Caching. RFC 9111. IETF, 2022. DOI: 10.17487/RFC9111.
6. OWASP Foundation. GraphQL Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html (дата обращения: 16.08.2026)