API, интеграции и распределённые данные

Программная инженерия
Лекция 4

Михаил Пожидаев

Институт прикладной математики и компьютерных наук ТГУ

16 ноября 2026 г.

Проектирование REST API

Ресурсы, HTTP-методы и предсказуемые интерфейсы

REST — архитектурный стиль взаимодействия компонентов через унифицированный интерфейс.

В REST API основными сущностями выступают ресурсы: заказы, пользователи, товары или платежи. Каждый ресурс должен иметь устойчивый адрес, например `/orders/12345`. URL описывает ресурс, а HTTP-метод описывает действие над ним.

Типичное использование методов HTTP

`GET /orders/12345` получает представление заказа. `POST /orders` создаёт новый заказ. `PATCH /orders/12345` изменяет отдельные поля заказа. `DELETE /orders/12345` удаляет заказ или переводит его в архив.

- Имена ресурсов обычно используют во множественном числе.
- В URL не следует включать глаголы, если действие выражается HTTP-методом.
- Вложенные пути стоит применять только для действительно зависимых ресурсов.
- Например, путь `/orders/12345/items` показывает элементы конкретного заказа.

Идемпотентность HTTP-методов

Повторный запрос не всегда должен менять результат

Идемпотентным называют запрос, повторное выполнение которого приводит к тому же наблюдаемому состоянию системы. Свойство идемпотентности особенно важно при сетевых сбоях и автоматических повторах запросов. Клиент может не получить ответ, даже если сервер уже успешно обработал операцию.

- `GET`, `PUT` и `DELETE` по смыслу должны быть идемпотентными.
- `POST` обычно не является идемпотентным, потому что каждый вызов может создать новый ресурс.
- Для критичных операций создания применяют ключ идемпотентности в заголовке `Idempotency-Key`.
- Сервер сохраняет результат операции, связанный с ключом, и возвращает его при повторном запросе.

Пример

Повторная отправка запроса на списание средств с одним ключом идемпотентности не должна создавать второе списание.

Интерфейс должен возвращать согласованный формат данных во всех методах. Для REST API чаще всего используют JSON. Поля ответа должны иметь стабильные имена и предсказуемые типы.

Структура ошибки

Ошибка может содержать машинно-читаемый код, сообщение для разработчика и идентификатор запроса. Идентификатор запроса помогает сопоставить ответ клиента с записями логов. Пример полей: `code`, `message`, `details` и `traceId`.

Постраничная выдача

Большие коллекции нельзя без ограничений возвращать одним ответом. API может использовать параметры `limit` и `cursor`. Cursor pagination лучше подходит для часто изменяемых списков, чем нумерация страниц через `offset`.

CC BY-NC 4.0

GraphQL

Запрос только нужных данных

GraphQL — язык запросов и среда выполнения API со строго описанной схемой типов. Клиент сам указывает поля, которые хочет получить в ответе. Обычно GraphQL API предоставляет одну HTTP-точку входа.
Например, экран заказа может запросить заказ, его позиции и данные покупателя одним запросом. Клиент не получает лишние поля, которые не нужны для отображения интерфейса.

- GraphQL уменьшает проблему over-fetching, когда сервер возвращает слишком много данных.
- Он также снижает число запросов при получении связанных сущностей.
- Схема GraphQL становится контрактом между клиентом и сервером.
- Для чтения обычно используют `query`, а для изменения данных — `mutation`.
- Для потоковых обновлений могут применяться `subscription`.

Ограничение

GraphQL требует контроля сложности запросов, глубины вложенности и стоимости выполнения резолверов.

gRPC — фреймворк удалённого вызова процедур, обычно использующий HTTP/2 и Protocol Buffers. Контракт сервиса описывается в файле `proto`. На основе контракта инструменты генерируют клиентский и серверный код для разных языков.

Возможности

- Unary RPC соответствует обычному запросу с одним ответом.
- Server streaming позволяет серверу передавать поток сообщений клиенту.
- Client streaming позволяет клиенту последовательно отправлять сообщения серверу.
- Bidirectional streaming поддерживает двусторонний поток сообщений.

- Protocol Buffers компактнее JSON и эффективнее при высоких нагрузках.
- gRPC хорошо подходит для внутренних взаимодействий микросервисов.
- Браузерная поддержка требует дополнительных решений, например gRPC-Web.
- Отладка gRPC вручную обычно сложнее, чем отладка JSON over HTTP.

CC BY-NC 4.0 | Институт прикладной математики и компьютерных наук ТГУ

Выбор протокола

REST, GraphQL и gRPC решают разные задачи

REST

- Подходит для публичных HTTP API и ресурсной модели предметной области.
- Хорошо поддерживается браузерами, прокси, инструментами мониторинга и кэширования.
- Прост в освоении для внешних интеграторов.

GraphQL

- Подходит для клиентских приложений со сложными экранами и множеством связанных данных.
- Даёт клиенту гибкость при выборе полей.
- Требует строгого контроля запросов и наблюдаемости на уровне резолверов.

gRPC

- Подходит для низколатентных внутренних вызовов и полиглотных микросервисов.
- Удобен при потоковой передаче данных и строгих контрактах.
- Требует поддержки инструментов для внешних клиентов.

API-first и contract-first

Интерфейс проектируется до реализации

- **API-first** означает, что интерфейс рассматривается как самостоятельный продукт и проектируется до серверной реализации.
- **Contract-first** означает, что разработка начинается с формального контракта: схемы, методов, сообщений и правил совместимости. Контракт обсуждают разработчики, аналитики, тестировщики и команды-потребители API.

1. Сначала определяют ресурсы, операции, модели данных и ошибки.
2. Затем контракт проходит ревью и публикуется в репозитории.
3. После этого можно параллельно разрабатывать клиент, сервер и автоматические тесты.
4. Заглушки сервера позволяют проверять интеграцию до готовности бизнес-логики.
5. Изменения контракта должны проходить те же проверки, что и исходный код.

Практический результат

Contract-first снижает риск ситуации, когда две команды по-разному понимают формат одного и того же запроса.

OpenAPI Specification

Машиночитаемое описание HTTP API

OpenAPI Specification — стандарт описания HTTP API в форматах YAML или JSON. Документ OpenAPI фиксирует пути, методы, параметры, тела запросов, ответы, схемы данных и механизмы безопасности. OpenAPI не реализует API, но создаёт единый контракт для всех участников интеграции.

Что можно получить из OpenAPI

Из спецификации можно сгенерировать интерактивную документацию. Можно создать типизированный SDK для клиента. Можно проверить корректность запросов и ответов в автоматических тестах. Можно поднять mock-сервер для ранней интеграции.

- Раздел `paths` описывает доступные HTTP-маршруты.
- Раздел `components/schemas` содержит переиспользуемые модели данных.
- Раздел `securitySchemes` описывает аутентификацию и авторизацию.
- Версия спецификации должна храниться вместе с кодом API.

Документация интерфейсов

API должно быть понятно без чтения исходного кода

Документация API нужна не только внешним разработчикам, но и внутренним командам. Хорошая спецификация отвечает на вопрос, как вызвать метод, какие данные передать и как обработать результат.

Минимальный состав документации

- Для каждого метода нужно указать назначение, HTTP-метод и путь.
- Нужно описать параметры, обязательность полей и допустимые значения.
- Следует привести примеры успешных ответов и типичных ошибок.
- Необходимо зафиксировать правила аутентификации, лимиты запросов и условия повторных вызовов.

Критическое правило

1. Документация не должна расходиться с работающим интерфейсом.
2. Проверка соответствия контракта и реализации должна быть частью CI/CD-конвейера.

Версионирование API

Изменения без поломки существующих клиентов

API развивается вместе с продуктом, но старые клиенты могут обновляться медленно. Обратная совместимость означает, что ранее корректный клиент продолжает работать после изменения API. Совместимые изменения желательно вносить без создания новой версии.

- Добавление необязательного поля в JSON-ответ обычно считается совместимым изменением.
- Добавление нового необязательного параметра запроса обычно считается совместимым изменением.
- Удаление поля, переименование поля или изменение его типа может сломать клиентов.
- Изменение смысла уже существующего поля тоже является потенциально несовместимым изменением.

Способы явного версионирования

- Версию можно включить в путь: `/v1/orders`.
- Версию можно передавать через заголовок `Accept` или специальный media type.
- Независимо от способа необходимо определить срок поддержки и правила вывода версии из эксплуатации.

Синхронные и асинхронные интеграции

Ответ сейчас или результат позже

- При синхронной интеграции вызывающая система ожидает результат операции в рамках текущего запроса.
- При асинхронной интеграции отправитель передаёт сообщение или задачу и не ждёт немедленного завершения обработки.

Синхронное взаимодействие

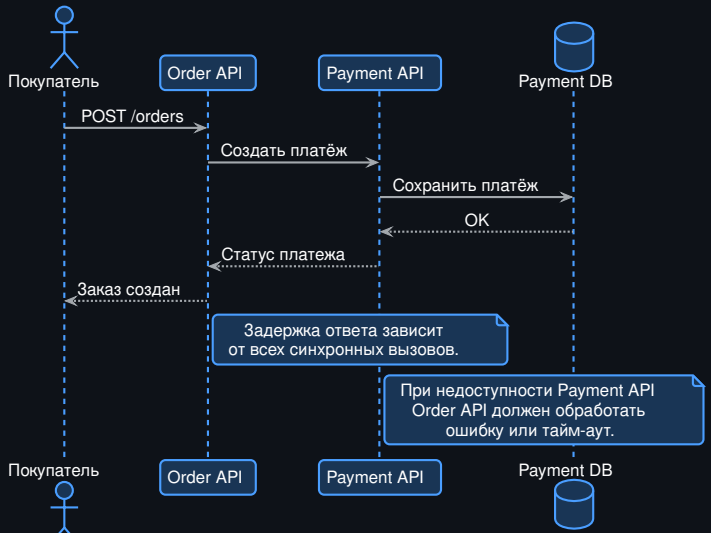
Подходит, когда пользователю нужен немедленный ответ. Примером является проверка цены товара при оформлении заказа. Основные риски — рост задержки и каскадные отказы зависимых сервисов.

Асинхронное взаимодействие

Подходит для длительных задач и слабосвязанных процессов. Примером является отправка письма после регистрации пользователя. Основные особенности — eventual consistency и необходимость отслеживать статус обработки.

Синхронный вызов сервиса

Клиент ожидает завершения цепочки запросов



Очереди сообщений и брокеры

Асинхронная передача событий и команд

Очередь сообщений разделяет отправителя и получателя во времени. Отправитель публикует сообщение в брокер, а обработчик получает его, когда готов к обработке. Брокер отвечает за хранение, маршрутизацию и доставку сообщений в соответствии с настройками.

Распространённые брокеры

- RabbitMQ поддерживает очереди, маршрутизацию и разные паттерны доставки.
- Apache Kafka ориентирован на журналы событий, партиции и высокую пропускную способность.
- NATS используется для лёгкого обмена сообщениями и событийных взаимодействий.
- Облачные платформы предоставляют управляемые очереди и топики.

- Сообщение должно содержать идентификатор, тип события, время создания и полезные данные.
- Формат схемы сообщения нужно версионировать так же внимательно, как HTTP API.
- Обработчик не должен зависеть от порядка доставки, если такой порядок не гарантирован явно.

Publish-subscribe

Одно событие для нескольких независимых подписчиков

В паттерне **publish-subscribe** издатель публикует событие в тему или топик. Издатель не знает, какие сервисы будут обрабатывать это событие. Каждый подписчик получает собственную копию события и выполняет свою работу независимо.

Событие `OrderCreated`

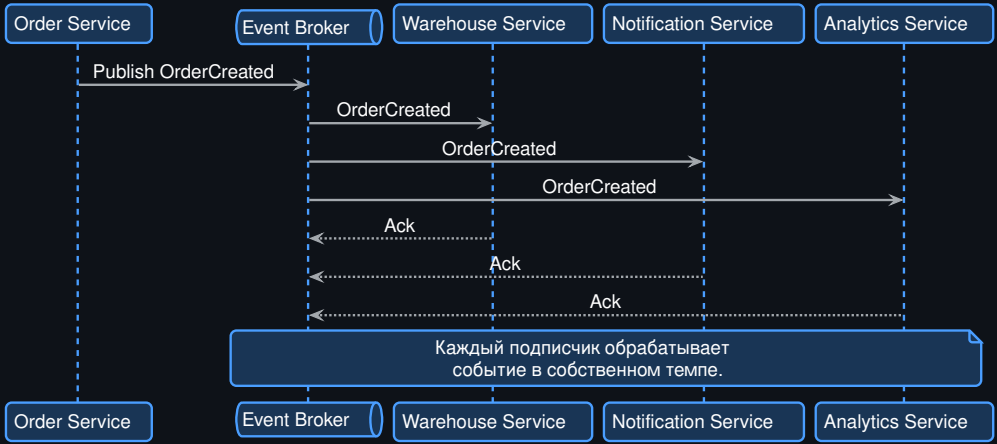
- Сервис уведомлений отправляет покупателю подтверждение.
- Сервис склада резервирует товары.
- Сервис аналитики обновляет показатели продаж.
- Сервис лояльности начисляет бонусные баллы.

Ограничение

Событие должно сообщать о факте, который уже произошёл, а не быть скрытой командой конкретному сервису. Например, `OrderCreated` выражает факт, а `ReserveItems` является командой для определённого получателя.

Publish-subscribe в обработке заказа

Подписчики реагируют на событие независимо



Retry и дедупликация

Доставка сообщений обычно имеет повторы

Распределённая система не может надёжно отличить потерянный ответ от медленного ответа. Поэтому брокеры и клиенты часто повторяют доставку или выполнение операции. На практике многие системы предоставляют семантику доставки at-least-once. Это означает, что одно и то же сообщение может быть обработано более одного раза.

- Обработчик должен проверять уникальный идентификатор сообщения или бизнес-операции.
- Результат успешно обработанного сообщения нужно сохранять в хранилище дедупликации.
- Повторные попытки стоит выполнять с экспоненциальной задержкой и случайным разбросом.
- Ограничение числа попыток защищает систему от бесконечной обработки неисправного сообщения.

Пример

Если событие о начислении бонусов пришло дважды, сервис лояльности должен начислить баллы только один раз.

Dead-letter queue

Изоляция сообщений, которые не удалось обработать

Dead-letter queue — отдельная очередь для сообщений, которые не были обработаны после заданного числа попыток. Причиной может быть ошибка схемы, нарушение бизнес-правила или временно недоступная зависимость. DLQ предотвращает бесконечное повторение одного проблемного сообщения в основной очереди.

Процесс обработки

Обработчик получает сообщение и пытается выполнить операцию. При временной ошибке он возвращает сообщение на повторную обработку. После исчерпания лимита попыток брокер переносит сообщение в DLQ. Команда анализирует причину, исправляет проблему и при необходимости запускает сообщение повторно.

DLQ не является заменой мониторингу

Для DLQ нужны метрики размера очереди, возраста сообщений и причин ошибок. Накопление сообщений в DLQ должно создавать предупреждение для команды эксплуатации.

Согласованность распределённых данных

Одна бизнес-операция часто затрагивает несколько хранилищ

В монолитном приложении несколько изменений можно объединить в одну транзакцию базы данных. В микросервисной архитектуре данные принадлежат разным сервисам и хранилищам. Общая распределённая транзакция часто делает систему медленнее, сложнее и менее доступной. Поэтому сервисы нередко принимают eventual consistency.

Eventual consistency означает, что сразу после операции разные части системы могут временно видеть разные состояния.

При отсутствии новых изменений система должна сойтись к согласованному состоянию. Пользовательский интерфейс должен корректно отображать промежуточные статусы, например «платёж обрабатывается».

Пример

После создания заказа склад может зарезервировать товар не мгновенно, а через несколько секунд после обработки события.

CAP-теорема

Компромисс при сетовом разделении

CAP-теорема рассматривает распределённые системы, в которых возможны сетевые разделения. При наличии разделения система не может одновременно гарантировать строгую согласованность и доступность каждого запроса.

- Под согласованностью в CAP понимается получение актуальных данных после успешной записи.
- Под доступностью понимается получение ответа на каждый запрос к доступному узлу.
- Tolerance to partition означает способность продолжать работу при потере связи между частями системы.

Практическая интерпретация

В реальных распределённых системах сетевые разделения необходимо учитывать. Поэтому во время разделения команда выбирает между отказом части запросов ради согласованности и ответом с потенциально устаревшими данными. CAP не означает, что систему можно навсегда отнести только к одному из трёх классов.

Saga

Согласование распределённой бизнес-операции

Saga — последовательность локальных транзакций нескольких сервисов. Каждая локальная транзакция фиксирует изменения в собственной базе данных. Если один из шагов завершился ошибкой, система запускает компенсирующие действия для уже выполненных шагов.

Оформление заказа

1. Сервис заказов создаёт заказ в статусе `PENDING`.
2. Сервис платежей резервирует средства.
3. Сервис склада резервирует товары.
4. После успешных шагов заказ получает статус `CONFIRMED`.
5. Если резервирование товара не удалось, сервис платежей отменяет резервирование средств.

- Оркестрация предполагает центральный компонент, управляющий шагами Saga.
- Хореография предполагает, что сервисы реагируют на события друг друга.
- Компенсация не всегда является точной отменой исходной операции.

Transactional outbox

Надёжная публикация событий после изменения данных

Проблема dual write возникает, когда сервис отдельно сохраняет данные в базу и публикует событие в брокер. Если база успешно обновилась, а публикация не произошла, другие сервисы не узнают об изменении. Если сначала опубликовано событие, а запись в базе не сохранилась, потребители увидят несуществующий факт.

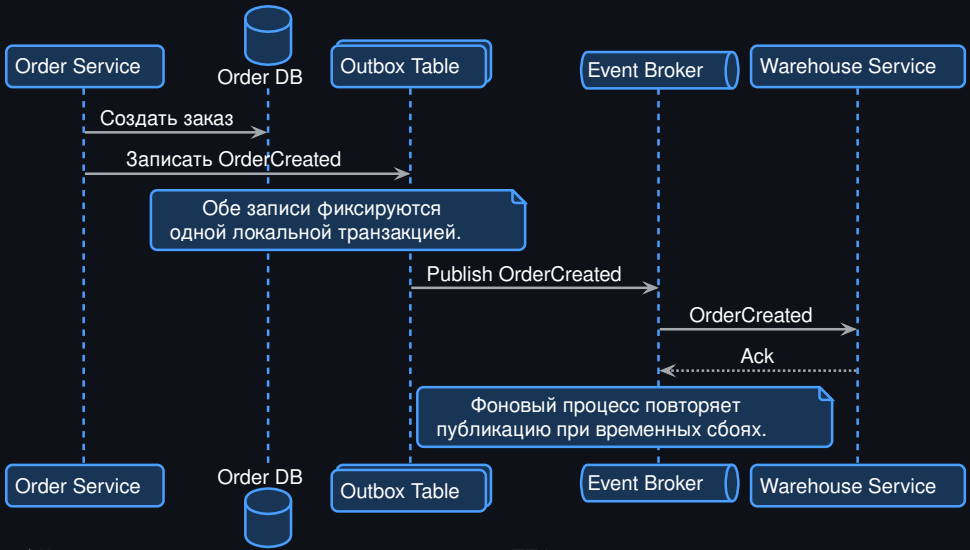
Transactional outbox решает проблему через таблицу исходящих сообщений в той же базе данных. Бизнес-данные и запись outbox сохраняются в одной локальной транзакции. Отдельный процесс читает outbox и публикует события в брокер. После подтверждения публикации запись помечается как отправленная.

Ограничение

Публикатор outbox также может отправить событие повторно после сбоя. Поэтому потребители всё равно должны быть идемпотентными.

Transactional outbox

Изменение данных и событие фиксируются вместе



Спасибо за внимание!

Всё о курсе:
https://marigostra.ru/materials/engineering-adv.html

Электронная почта:
msp@luwrain.org

Telegram:
@Marigostra



Материал можно распространять с соблюдением требований Creative Commons Attribution-NonCommercial 4.0 International License. При обнаружении любых ошибок и неточностей в материале просьба сообщить о них автору.