# 1. Введение

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола.

Системы, поддерживающие REST, называются RESTful-системами.   
В сети [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) [вызов удалённой процедуры](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80) может представлять собой обычный [HTTP](https://ru.wikipedia.org/wiki/HTTP)-запрос (обычно «GET» или «POST»; такой запрос называют *«REST-запрос»*), а необходимые данные передаются в качестве [параметров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80) запроса.

Для [веб-служб](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0" \o "Веб-служба), построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «**RESTful**».

В отличие от веб-сервисов (веб-служб) на основе [SOAP](https://ru.wikipedia.org/wiki/SOAP), не существует «официального» стандарта для RESTful веб-API. Дело в том, что REST является **архитектурным стилем**, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют стандарты, такие как [HTTP](https://ru.wikipedia.org/wiki/HTTP), [URL](https://ru.wikipedia.org/wiki/URL), [JSON](https://ru.wikipedia.org/wiki/JSON) и [XML](https://ru.wikipedia.org/wiki/XML).

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат. Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Updtae-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Архитектура REST очень проста в плане использования. По виду пришедшего запроса сразу можно определить, что он делает, не разбираясь в форматах (в отличие от SOAP, XML-RPC). Данные передаются без применения дополнительных слоев, поэтому REST считается менее ресурсоемким, поскольку не надо парсить запрос чтоб понять что он должен сделать и не надо переводить данные из одного формата в другой.

# Свойства Архитектуры REST

Свойства архитектуры, которые зависят от ограничений, наложенных на REST-системы:

1. Производительность — взаимодействие компонентов системы может являться доминирующим фактором производительности и эффективности сети с точки зрения пользователя;

2. Масштабируемость для обеспечения большого числа компонентов и взаимодействий компонентов.

Рой Филдинг — один из главных авторов спецификации протокола HTTP, описывает влияние архитектуры REST на масштабируемость следующим образом:

- Простота унифицированного интерфейса;

- Открытость компонентов к возможным изменениям для удовлетворения изменяющихся потребностей (даже при работающем приложении);

- Прозрачность связей между компонентами системы для сервисных служб;

- Переносимость компонентов системы путем перемещения программного кода вместе с данными;

- Надёжность, выражающаяся в устойчивости к отказам на уровне системы при наличии отказов отдельных компонентов, соединений или данных.

# Правила и ограничения REST-архитектуры

В REST есть шесть основных ограничений:

1) Модель взаимодействия клиент-сервер.

Первым ограничением, применимым к гибридной модели, является приведение архитектуры к модели клиент-сервер. Разграничение потребностей является принципом, лежащим в основе данного накладываемого ограничения. Отделение потребности интерфейса [клиента](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) от потребностей [сервера, хранящего данные](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)), повышает переносимость [кода](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) клиентского [интерфейса](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) на другие платформы, а упрощение [серверной части](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)) улучшает масштабируемость. Наибольшее же влияние на [всемирную паутину](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0), пожалуй, имеет само разграничение, которое позволяет отдельным частям развиваться независимо друг от друга, поддерживая потребности в развитии интернета со стороны различных организаций.

2) Отсутствие состояния.

Протокол взаимодействия между клиентом и сервером требует соблюдения следующего условия: в период между запросами клиента никакая информация о *состоянии* клиента на сервере не хранится ([Stateless protocol](https://en.wikipedia.org/wiki/Stateless_protocol" \o "en:Stateless protocol) (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=Stateless_protocol&action=edit&redlink=1)). Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. *Состояние* сессии при этом сохраняется на стороне клиента[[2]](https://ru.wikipedia.org/wiki/REST#cite_note-Fielding-Ch5-2). Информация о состоянии сессии может быть передана сервером какому-либо другому сервису (например, в службу базы данных) для поддержания устойчивого состояния, например, на период установления аутентификации. Клиент инициирует отправку запросов, когда он готов (возникает необходимость) перейти в новое состояние.

Во время обработки клиентских запросов считается, что клиент находится в *переходном состоянии*. Каждое отдельное *состояние* приложения представлено связями, которые могут быть задействованы при следующем обращении клиента.

3) Кэширование.

Как и во [Всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0), клиенты, а также промежуточные узлы, могут выполнять [кэширование](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) ответов сервера. Ответы сервера, в свою очередь, должны иметь явное или неявное обозначение как кэшируемые или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных в ответ на последующие запросы. Правильное использование кэширования способно полностью или частично устранить некоторые клиент-серверные взаимодействия, ещё больше повышая производительность и расширяемость системы.

4) Единообразие интерфейса.

Наличие унифицированного интерфейса является фундаментальным требованием дизайна REST-сервисов. Унифицированные интерфейсы позволяют каждому из сервисов развиваться независимо.

К унифицированным интерфейсам предъявляются следующие четыре ограничительных условия:

**Идентификация ресурсов**

Все ресурсы идентифицируются в запросах, например, с использованием URI в интернет-системах. Ресурсы концептуально отделены от представлений, которые возвращаются клиентам. Например, [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) может отсылать данные из [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) в виде [HTML](https://ru.wikipedia.org/wiki/HTML), [XML](https://ru.wikipedia.org/wiki/XML) или [JSON](https://ru.wikipedia.org/wiki/JSON), ни один из которых не является типом хранения внутри сервера.

**Манипуляция ресурсами через представление**

Если клиент хранит представление ресурса, включая метаданные — он обладает достаточной информацией для модификации или удаления ресурса.

**«Самоописываемые» сообщения**

Каждое сообщение содержит достаточно информации, чтобы понять, каким образом его обрабатывать. К примеру, обработчик сообщения (parser), необходимый для извлечения данных, может быть указан в [списке MIME-типов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_MIME-%D1%82%D0%B8%D0%BF%D0%BE%D0%B2).

**Гипермедиа как средство изменения состояния приложения (**[**HATEOAS**](https://ru.wikipedia.org/wiki/HATEOAS)**)**

Клиенты изменяют состояние системы только через действия, которые динамически определены в гипермедиа на сервере (к примеру, [гиперссылки](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0) в [гипертексте](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82)). Исключая простые точки входа в приложение, клиент не может предположить, что доступна какая-то операция над каким-то ресурсом, если не получил информацию об этом в предыдущих запросах к серверу. Не существует универсального формата для предоставления ссылок между ресурсами, [RFC 5988](https://tools.ietf.org/html/rfc5988) и [JSON Hypermedia API Language](https://tools.ietf.org/id/draft-kelly-json-hal-03.txt) являются двумя популярными форматами предоставления ссылок в REST HYPERMEDIA сервисах.

5) Слои.

Клиент обычно не способен точно определить, взаимодействует он напрямую с [сервером](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) или же с промежуточным узлом, в связи с иерархической структурой сетей (подразумевая, что такая структура образует [слои](https://ru.wikipedia.org/wiki/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%8F)). Применение промежуточных серверов способно повысить масштабируемость за счёт [балансировки нагрузки](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BD%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B8) и распределённого [кэширования](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Промежуточные узлы также могут подчиняться [политике безопасности](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0_%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D0%B8) с целью обеспечения [конфиденциальности информации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F).

6) Код по требованию

REST может позволить расширить функциональность клиента за счёт загрузки кода с сервера в виде [апплетов](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%BB%D0%B5%D1%82" \o "Апплет) или [сценариев](https://ru.wikipedia.org/wiki/%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA). Филдинг утверждает, что дополнительное ограничение позволяет проектировать архитектуру, поддерживающую желаемую функциональность в общем случае, но, возможно, за исключением некоторых контекстов.

# Передача данных между клиентом и сервером

Формат данных, передаваемых между клиентом и сервисом указывается с помощью заголовка HTTP — параметр типа содержимого Content-Type. Обычно это JSON или XML, но могут быть разнообразными (например, JPG, PNG — для сервиса обработки графики).

# Преимущества

Филдинг указывал, что приложения, не соответствующие приведённым условиям, не могут называться REST-приложениями. Если же все условия соблюдены, то, по его мнению, приложение получит следующие преимущества[[2]](https://ru.wikipedia.org/wiki/REST#cite_note-Fielding-Ch5-2)[[6]](https://ru.wikipedia.org/wiki/REST#cite_note-SOAwithREST-6):

- Надёжность (за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);

- Производительность (за счёт использования кэша);

- Масштабируемость;

- Прозрачность системы взаимодействия (особенно необходимая для приложений обслуживания [сети](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C));

- Простота [интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81);

- Портативность компонентов;

- Лёгкость внесения изменений;

- Способность [эволюционировать](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D0%BE%D0%BB%D1%8E%D1%86%D0%B8%D1%8F), приспосабливаясь к новым требованиям (на примере [Всемирной паутины](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0)).

# Практическое применение.

Самое главное достоинство сервисов в том, что с ними работать может какая угодно система, будь то сайт, flash, программа и др. так как методы парсинга XML и выполнения запросов HTTP присутствуют почти везде.   
  
Архитектура REST позволяет серьезно упростить эту задачу. Конечно в реальности, того что описано не достаточно, ведь нельзя кому угодно давать возможность изменять информацию, то есть нужна еще авторизация и аутентификация.

# Вывод

Как видно, в архитектура REST очень проста в плане использования. По виду пришедшего запроса сразу можно определить, что он делает, не разбираясь в форматах (в отличие от SOAP, XML-RPC). Данные передаются без применения дополнительных слоев, поэтому REST считается менее ресурсоемким, поскольку не надо парсить запрос чтоб понять что он должен сделать и не надо переводить данные из одного формата в другой.