# **Работа с REST API.**

# **План**

**Источники**

# [https://ru.stackoverflow.com/questions/519007/rest-api-rest-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D1%87%D1%82%D0%BE-%D1%8D%D1%82%D0%BE-%D0%B8-%D1%81-%D1%87%D0%B5%D0%BC-%D0%B5%D0%B3%D0%BE-%D0%B5%D0%B4%D1%8F%D1%82-full](https://medium.com/@andr.ivas12/rest-простым-языком-90a0bca0bc78)

# [https://medium.com/@andr.ivas12/rest-%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%BC-%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%BC-90a0bca0bc78](https://medium.com/@andr.ivas12/rest-простым-языком-90a0bca0bc78)

* https://habr.com/ru/post/464261/

# [**REST**](https://ru.wikipedia.org/wiki/REST)

(REpresentational State Transfer) — это архитектура, т.е. принципы построения распределенных гипермедиа систем, того что другими словами называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP

Автор идеи и термина [Рой Филдинг](https://en.wikipedia.org/wiki/Roy_Fielding) 2000г.

REST на сегодняшний день практически вытеснил все остальные подходы

# **Что нам дает REST подход**

* Масштабируемости взаимодействия компонентов системы (приложения)
* Общность интерфейсов
* Независимое внедрение компонентов
* Промежуточные компоненты, снижающие задержку, усиливающие безопасность

# **Когда использовать REST?**

• Когда есть ограничение пропускной способности соединения

• Если необходимо кэшировать запросы

• Если система предполагает значительное масштабирование

• В сервисах, использующих AJAX

# **Преимущества REST:**

* Надёжность (за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
* Производительность (за счёт использования кэша);
* Масштабируемость;
* Прозрачность системы взаимодействия;
* Простота [интерфейсов](https://ru.wikipedia.org/wiki/Интерфейс);
* Портативность компонентов;
* Лёгкость внесения изменений;
* Способность [эволюционировать](https://ru.wikipedia.org/wiki/Эволюция), приспосабливаясь к новым требованиям.
* Отсутствие дополнительных внутренних прослоек, что означает передачу данных в том же виде, что и сами данные. Просто отдаются сами данные.
* Каждая единица информации (**ресурс**) однозначно определяется URL — это значит, что URL по сути является первичным ключом для единицы данных. Причем совершенно не имеет значения, в каком формате находятся данные по адресу.
* Как происходит управление информацией ресурса — это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же [HTTP](https://ru.wikipedia.org/wiki/HTTP). Для HTTP действие над данными задается с помощью методов : GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия [CRUD](https://ru.wikipedia.org/wiki/CRUD) (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

# **Что такое RESTful:**

Чтобы распределенная система считалась сконструированной по REST архитектуре (Restful), необходимо, чтобы она удовлетворяла следующим критериям:

1. **Client-Server.** Система должна быть разделена на клиентов и на серверов. Разделение интерфейсов означает, что, например, клиенты не связаны с хранением данных, которое остается внутри каждого сервера, так что мобильность кода клиента улучшается. Серверы не связаны с интерфейсом пользователя или состоянием, так что серверы могут быть проще и масштабируемы. Серверы и клиенты могут быть заменяемы и разрабатываться независимо, пока интерфейс не изменяется.
2. **Stateless.** Сервер не должен хранить какой-либо информации о клиентах. В запросе должна храниться вся необходимая информация для обработки запроса и если необходимо, идентификации клиента.
3. **Cache․** Каждый ответ должен быть отмечен является ли он кэшируемым или нет, для предотвращения повторного использования клиентами устаревших или некорректных данных в ответ на дальнейшие запросы.
4. **Uniform Interface.** Единый интерфейс определяет интерфейс между клиентами и серверами. Это упрощает и отделяет архитектуру, которая позволяет каждой части развиваться самостоятельно.

*Четыре принципа единого интерфейса:*

* *Identification of resources (основан на ресурсах).***В REST ресурсом является все то, чему можно дать имя**. Например,пользователь, изображение, предмет (майка, голодная собака, текущая погода) и т.д. Каждый ресурс в REST должен быть идентифицирован посредством стабильного идентификатора, который не меняется при изменении состояния ресурса. **Идентификатором в REST является URI.**
* *Manipulation of resources through representations. (Манипуляции над ресурсами через представления).* Представление в REST используется для выполнения действий над ресурсами. Представление ресурса представляет собой текущее или желаемое состояние ресурса. Например, если ресурсом является пользователь, то представлением может являться XML или HTML описание этого пользователя.
* *Self-descriptive messages (само-документируемые сообщения)*. Под само-описательностью имеется ввиду, что запрос и ответ должны хранить в себе всю необходимую информацию для их обработки. Не должны быть дополнительные сообщения или кэши для обработки одного запроса.
* *HATEOAS (hypermedia as the engine of application state).* Статус ресурса передается через содержимое body, параметры строки запроса, заголовки запросов и запрашиваемый URI (имя ресурса). Это называется гипермедиа (или гиперссылки с гипертекстом). HATEOAS также означает, что, в случае необходимости ссылки могут содержатся в теле ответа (или заголовках) для поддержки URI , извлечения самого объекта или запрошенных объектов.

5 .**Layered System**. В REST допускается разделить систему на иерархию слоев но с условием, что каждый компонент может видеть компоненты только непосредственно следующего слоя. Например, если вы вызывайте службу PayPal а он в свою очередь вызывает службу Visa, вы о вызове службы Visa ничего не должны знать.

6. **Code-On-Demand (опционально)**. В REST позволяется загрузка и выполнение кода или программы на стороне клиента. Серверы могут временно расширять или кастомизировать функционал клиента, передавая ему логику, которую он может исполнять. Например, это могут быть клиентские скрипты на Javascript

**Важно !** Сама архитектура REST не привязана к конкретным технологиям и протоколам, но в реалиях современного Веб, построение RESTful API почти всегда подразумевает использование HTTP и каких-либо распространенных форматов представления ресурсов, например JSON, или, менее популярного сегодня, XML.

# **Идемпотентность**

С точки зрения RESTful-сервиса, операция (или вызов сервиса) идемпотентна тогда, когда клиенты могут делать один и тот же вызов неоднократно при одном и том же результате на сервере. Другими словами, создание большого количества идентичных запросов имеет такой же эффект, как и один запрос. Заметьте, что в то время, как идемпотентные операции производят один и тот же результат на сервере, ответ сам по себе может не быть тем же самым (например, состояние ресурса может измениться между запросами).

Методы **PUT** и **DELETE** по определению идемпотентны. Тем не менее, есть один нюанс с методом DELETE. Проблема в том, что успешный DELETE-запрос возвращает статус 200 (OK) или 204 (No Content), но для последующих запросов будет все время возвращать 404 (Not Found), Состояние на сервере после каждого вызова DELETE то же самое, но ответы разные.

Методы **GET**, **HEAD**, **OPTIONS** и **TRACE** определены как безопасные. Это означает, что они предназначены только для получения информации и не должны изменять состояние сервера. Они не должны иметь побочных эффектов, за исключением безобидных эффектов, таких как: логирование, кеширование, показ баннерной рекламы или увеличение веб-счетчика.

По определению, **безопасные операции идемпотентны**, так как они приводят к одному и тому же результату на сервере. Безопасные методы реализованы как операции только для чтения. Однако безопасность не означает, что сервер должен возвращать тот же самый результат каждый раз.

Create →POST

Read →GET, HEAD

Update → PUT, PATCH

Delete → DELETE

# **HTTP методы для создания RESTful сервисов**

**HTTP метод GET** используется для получения (или чтения) представления ресурса. В случае “удачного” (или не содержащего ошибок) адреса, GET возвращается представление ресурса в формате XML или JSON в сочетании с кодом состояния HTTP 200 (OK). В случае наличия ошибок обычно возвращается код 404 (NOT FOUND) или 400 (BAD REQUEST).

* ***GET***[***http://www.example.com/api/v1.0/users***](http://www.example.com/api/v1.0/users) *(вернуть список пользователей)*
* ***GET***[***http://www.example.com/api/v1.0/users/12345***](http://www.example.com/api/v1.0/users/12345) *(вернуть данные о пользователе с id 12345)*
* ***GET***[***http://www.example.com/api/v1.0/users/12345/orders***](http://www.example.com/api/v1.0/users/12345/orders)

**HTTP метод PUT** обычно используется для предоставления возможности обновления ресурса. Тело запроса при отправке PUT-запроса к существующему ресурсу URI должно содержать обновленные данные оригинального ресурса (полностью, или только обновляемую часть).

Для создания новых экземпляров ресурса предпочтительнее использование POST запроса. В данном случае, при создании экземпляра будет предоставлен корректный идентификатор экземпляра ресурса в возвращенных данных об экземпляре.

При успешном обновлении посредством выполнения PUT запроса возвращается код 200 (или 204 если не был передан какой либо контент в теле ответа). PUT не безопасная операция, так как вследствии ее выполнения происходит модификация (или создание) экземпляров ресурса на стороне сервера, но этот метод идемпотентен. Другими словами, создание или обновление ресурса посредством отправки PUT запроса — ресурс не исчезнет, будет располагаться там же, где и был при первом обращении, а также, многократное выполнение одного и того же PUT запроса не изменит общего состояния системы

* ***PUT***[***http://www.example.com/api/v1.0/users/12345***](http://www.example.com/api/v1.0/users/12345) *(обновить данные пользователя с id 12345)*
* ***PUT***[***http://www.example.com/api/v1.0/users/12345/orders/98765***](http://www.example.com/api/v1.0/users/12345/orders/98765) *(обновить данные заказа с id 98765 для пользователя с id 12345)*

**HTTP метод POST**запрос наиболее часто используется для создания новых ресурсов. На практике он используется для создания вложенных ресурсов. Другими словами, при создании нового ресурса, POST запрос отправляется к родительскому ресурсу и, таким образом, сервис берет на себя ответственность на установление связи создаваемого ресурса с родительским ресурсом, назначение новому ресурсу ID и т.п.

При успешном создании ресурса возвращается HTTP код 201, а также в заголовке `Location` передается адрес созданного ресурса.

POST не является безопасным или идемпотентным запросом. Потому рекомендуется его использование для не идемпотентных запросов. В результате выполнения идентичных POST запросов предоставляются сильно похожие, но не идентичные данные.

* ***POST***[***http://www.example.com/api/v1.0/customers***](http://www.example.com/api/v1.0/customers) *(создать новый ресурс в разделе customers)*
* ***POST***[***http://www.example.com/api/v1.0/customers/12345/orders***](http://www.example.com/api/v1.0/customers/12345/orders) *(создать заказ для ресурса с id 12345)*

**HTTP метод DELETE**используется для удаления ресурса, идентифицированного конкретным URI (ID).

При успешном удалении возвращается 200 (OK) код HTTP, совместно с телом ответа, содержащим данные удаленного ресурса. Также возможно использование HTTP кода 204 (NO CONTENT) без тела ответа. Согласно спецификации HTTP, DELETE запрос идемпотентен. Если вы выполняете DELETE запрос к ресурсу, он удаляется. Повторный DELETE запрос к ресурсу закончится также: ресурс удален. Если DELETE запрос используется для декремента счетчика, DELETE запрос не является идемпотентным. Используйте POST для не идемпотентных операций.

Тем не менее, существует предостережение об идемпотентности DELETE. Повторный DELETE запрос к ресурсу часто сопровождается 404 (NOT FOUND) кодом HTTP по причине того, что ресурс уже удален (например из базы данных) и более не доступен. Это делает DELETE операцию не идемпотентной, но это общепринятый компромисс на тот случай, если ресурс был удален из базы данных, а не помечен, как удаленный.

* ***DELETE***[***http://www.example.com/api/v1.0/customers/12345***](http://www.example.com/api/v1.0/customers/12345) *(удалить из customers ресурс с id 12345)*
* ***DELETE***[***http://www.example.com/api/v1.0/customers/12345/orders***](http://www.example.com/api/v1.0/customers/12345/orders)***/21****(удалить у ресурса с id 12345 заказ с id 21)*

# **Коды состояний HTTP**

Основные группы кодов состояний: ([link](https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP))

**1xx: Information**

**2xx: Success**

200: OK

201: Created

**3xx: Redirect**

**4xx: Client Error**

400: Bad Request

401: Unauthorized

403: Forbidden

404: Not Found

**5xx: Server Error**

500: Internal Server Error