Аббревиатура API расшифровывается как «Application Programming Interface» (интерфейс программирования приложений, программный интерфейс приложения). Большинство крупных компаний на определённом этапе разрабатывают API для клиентов или для внутреннего использования.  
  
Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера. API — это составляющая часть сервера, которая получает запросы и отправляет ответы.  
Чтобы сгенерировать полную веб-страницу, браузер ожидает ответ на языке разметки HTML, в то время как API Google Календаря вернёт просто данные в формате вроде JSON.

REST - подход к построению апи.

REST (RESTful) - это общие принципы организации взаимодействия приложения/сайта с сервером посредством протокола HTTP. Особенность REST в том, что сервер не запоминает состояние пользователя между запросами - в каждом запросе передаётся информация, идентифицирующая пользователя (например, token, полученный через OAuth-авторизацию) и все параметры, необходимые для выполнения операции.

Для каждого типа операции используется свой метод HTTP-запроса:  
1. получение - GET  
2. добавление - POST (содержит тело)  
3. модификация - PUT  
4. удаление - DELETE

GET – используется для получения со стороны севера определенного ресурса. Если вы производите этот запрос, сервер ищет информацию и отправляет ее вам назад. По сути, он производит операцию чтения на сервере. Дефолтный тип запросов.

POST – нужен для создания/добавление определенного ресурса на сервере. Сервер создает в базе данных новую сущность и оповещает вас, был ли процесс создания успешным. По сути, это операция создания.

PUT и PATCH – используются для обновления/изменения определенной информации на сервере. В таком случае сервер просто изменяет информацию существующих сущностей в базе данных и оповещает об успехе выполнения операции.

DELETE – как и следует из названия, удаляет указанную сущность из базы или сигнализирует об ошибке, если такой сущности в базе не было.

PUT =заменить ВЕСЬ РЕСУРС новым предоставленным представлением  
 мы должны отправить оба параметра первым и последним.

Другими словами, необходимо снова отправить все значения, всю полезную нагрузку.

Однако при отправке запроса на исправление(PATCH - замена частей исходного ресурса предоставленными значениями) мы отправляем только те данные, которые хотим обновить. Другими словами, мы отправляем только имя для обновления, нет необходимости отправлять фамилию.

https://overcoder.net/q/8013/rest-api-put-vs-patch-%D1%81-%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D0%B0%D0%BC%D0%B8-%D0%B8%D0%B7-%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B9-%D0%B6%D0%B8%D0%B7%D0%BD%D0%B8#:~:text=PATCH%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D0%B2%20RFC%205789,%D0%B2%D1%80%D0%B5%D0%BC%D1%8F%20%D0%BA%D0%B0%D0%BA%20PATCH%20%D0%BD%D0%B5%20%D1%8F%D0%B2%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F.

- API - программный интерфейс для общения программ  
- REST - это интерфейс общения через HTTP протокол

если смотреть на REST API есть 3 основных подтипа:  
- REST API JSON  
- REST API XML  
- REST API GraphQL  
  
Это все разные способы взаимодействия.

Как писать апи запросы

<https://itvdn.com/ru/blog/article/rest-api-18>

# **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.

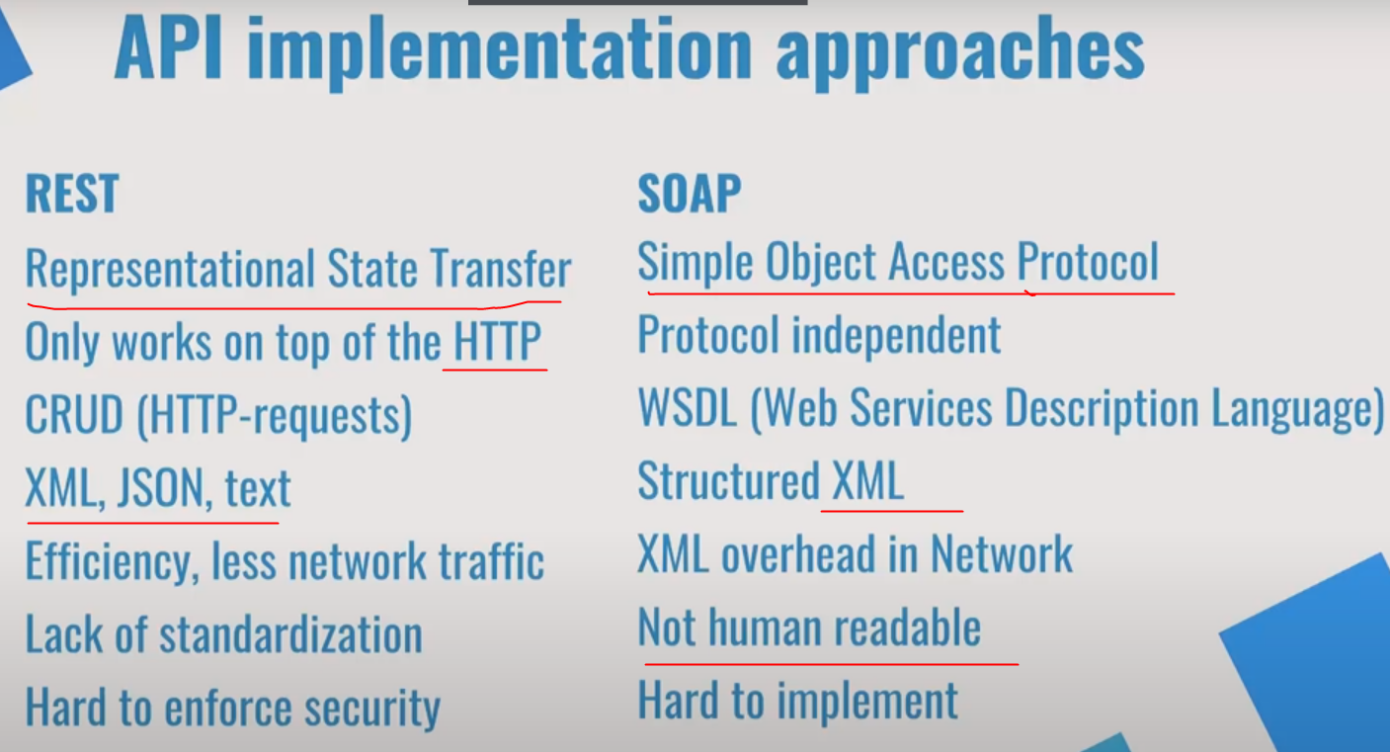
**REST – это просто архитектура, он никак не привязан к каким либо протоколом.**

 REST — это архитектурный стиль: подход к построению апи.

SOAP — это формат обмена сообщениями.  
  
Пример реализации **RESTful**: JSON через HTTP  
Пример реализации **SOAP**: XML поверх SOAP через HTTP

SOAP (протокол) отправляет посредством HTTP-запроса XML-сообщение и получает результат в HTTP-отклике.  
SOAP-сообщение представляет собой XML-документ; сообщение состоит из трех основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body)  
<http://khpi-iip.mipk.kharkiv.edu/library/sotii/lectures/Lecture5.pdf>

* В SOAP вы используете формат SOAP XML для запросов и ответов.
* В REST такого фиксированного формата нет. Вы можете обмениваться сообщениями на основе XML, JSON или любого другого удобного формата. JSON является самым популярным среди используемых форматов.  
    
    
    
    
  [Rest Assured](http://rest-assured.io/" \t "https://medium.com/kotlin-lang-notes/_blank) — это Java-библиотека для тестирования RESTful API. Код, написанный с помощью этой библиотеки, имеет простой и понятный синтаксис. В тесте можно выполнить запрос к API буквально в одну строчку кода и при помощи DSL синтаксиса проверять полученный результат.



Заголовки (headers) содержат мета-информацию, параметры - фактические данные (meta info - это средство классификации, упорядочивания и характеристики данных или содержимого).

Серверная сторона автоматически распишет/декодирует имена / значения параметров. Это не относится к именам/значениям headers.

Headers’ имена / значения должны быть вручную экранированы/закодированы на стороне клиента и вручную не экранированы/декодированы на стороне сервера. Часто используется кодировка Base64 или процентное экранирование.

Параметры могут быть видны конечным пользователям (параметры запроса) на URL-адресе, но headers скрыты для конечных пользователей.  
  
<https://developer.mozilla.org/ru/docs/Web/HTTP/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BA%D0%B8>

Блок **given** описывает предусловия, такие как авторизация и параметры запроса. Обратите внимание: отдельно можно провести базовую http-авторизацию (**auth().basic(«login», «password»)**) и пользовательскую авторизацию, передав нужные куки (**cookies(«sessionId», session)**). Очень удобно использовать опцию **relaxedHTTPSValidation()**, чтобы избежать хлопот с подтверждением сертификатов. Блок **when()** описывает требуемое действие – запрос какого типа и на какой адрес следует отправить. Блок **then()** включает проверки, которые необходимо произвести (их может быть несколько одновременно).

<https://quality-lab.ru/blog/rest-api-testing/>

<https://www.codeflow.site/ru/article/rest-assured-tutorial>

Постмен читать <https://proglib.io/p/api-dlya-qa-uchimsya-testirovaniyu-po-bez-dostupa-k-kodu-2020-12-10>