ЧТО ТАКОЕ API

**API («Application Programming Interface» или «Программный интерфейс приложения») –** это такой набор способов (а вернее – функций), с помощью которых одна программа может обращаться к другой программе минуя пользовательский интерфейс.

Например, если нам нужно добавить на сайт актуальную информацию о количестве подписчиков в группе ВК, то это можно сделать, обратясь по специальному запросу к API Вконтакте.

Годная статья по основам REST API: <https://wp-kama.ru/handbook/rest/basic>

REST И SOAP – ЧТО ЭТО?

Многие сравнивают REST и SOAP, хотя это не совсем корректно. REST и SOAP на самом деле не сопоставимы. REST — это архитектурный стиль, а SOAP — это формат обмена сообщениями.

**SOAP (от англ. Simple Object Access Protocol — простой протокол доступа к объектам)** – протокол обмена данными, представленными в формате XML. А XML — это язык для описания данных, причём он очень похож на HTML, но предназначен для хранения и транспортировки данных, в то время как HTML предназначен, в первую очередь, для правильности отображения данных.

**REST (от англ. Representational State Transfer — «передача состояния представления») —** это набор правил, по которым программист организовывает написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать.

*- В* ***REST*** *уже может использоваться обмен данными в форматах и* ***XML****, и* ***JSON****, и какие-нибудь другие. Наиболее популярным является JSON.*

*-* ***REST*** *работает только по протоколам HTTP/HTTPS*

**RESTful веб-сервис/система** – такой сервис/система, которые полностью отвечают требованиям REST

Кратко об отличиях:

- SOAP это протокол, REST – это архитектурный стиль, SOAP имеет веб-сервис WSDL с прописанными методами, которые можно удаленно вызывать.

- REST использует Json и XML, SOAP только XML.

- REST работает только по HTTP/HTTPS, SOAP с любым протоколом прикладного уровня: SMPT, FTP, HTTP, HTTPS, POP3

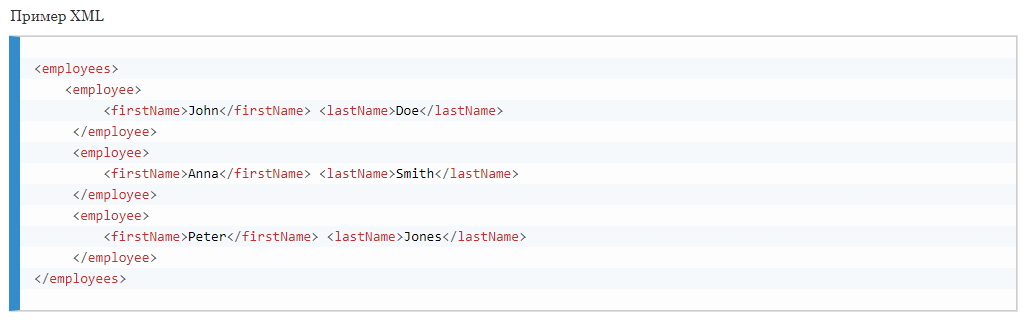
- REST более простой, гибкий и быстрый, SOAP типизированный, но в некоторых случаях лучше визуализируется за счет применения им синтаксиса похожего на HTML разметку.

XML И JSON ФОРМАТЫ

И JSON, и XML могут использоваться для получения данных с веб-сервера.

В следующих примерах JSON и XML определяется один и тот же объект employees с массивом из 3 элементов:





### Судя по внешнему виду этих файлов, данные в JSON передаются как пары ключ: значение, а в XML они передаются в открывающих и закрывающих тегах

### JSON похож на XML тем, что

* И JSON, и XML имеют "самоописательную" структуру (понятную для человека)
* И JSON, и XML имеют иерархическую структуру (значения внутри значений)
* И JSON, и XML могут парситься и использоваться любым языком программирования
* И JSON, и XML могут извлекаться при помощи XMLHttpRequest

### JSON не похож на XML тем, что

* JSON не использует конечный тег
* JSON короче
* JSON быстрее читать и писать
* JSON может использовать массивы

Самое значительное отличие JSON от XML заключается в том, что XML приходится парсить при помощи специального XML парсера, а JSON может парситься стандартной функцией JavaScript.

### Почему JSON лучше XML

Парсить XML гораздо сложнее, чем JSON. При этом JSON парсится в уже готовый для использования объект JavaScript.

Для AJAX приложений JSON быстрее и проще XML

КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

HTTP-ПРОТОКОЛ

*HTTP-протокол* – это протокол, который позволяет обмениваться данными между клиентом (обычно в роли клиента выступает браузер) и сервером. Он также определяет то, КАК ИМЕННО будет происходить обмен данными.

У этого протокола есть широко используемое расширение – *HTTPS (HyperText Transfer Protocol Secure)*. Данные, передаваемые через этот протокол, зашифровываются в криптографический вид (в протоколы SSL или TLS).

**СТРУКТУРА HTTP ЗАПРОСА**

HTTP запрос состоит из трех основных частей, которые идут в нем именно в том порядке, который указан ниже. Между заголовками и телом сообщения находится пустая строка (в качестве разделителя), она представляет собой символ перевода строки.



1. строка запроса (Request Line)

2. заголовки (Message Headers)

Пустая строка (разделитель)

3. тело сообщения (Entity Body) – необязательный параметр. Также зовётся как Payload или полезная нагрузка.

*Строка запроса* – указывает метод передачи, URL-адрес, к которому нужно обратиться и версию протокола HTTP.

***URI (Uniform Resource Identifier)*** – это идентификатор ресурса на который отправляется запрос.

*Заголовки* – описывают тело сообщений, передают различные параметры и др. сведения и информацию.

*тело сообщения (Payload)* - это сами данные, которые передаются в запросе. Тело сообщения – это необязательный параметр и может отсутствовать.

Когда мы получаем ответный запрос от сервера, тело сообщения, чаще всего представляет собой содержимое веб-страницы. Но, при запросах к серверу, оно тоже может иногда присутствовать, например, когда мы передаем данные, которые заполнили в форме обратной связи на сервер.

**МЕТОДЫ**

HTTP-методы GET и POST — самые распространённые способы отправить или получить данные с сервера. Также есть ещё два известных метода – PUT и DELETE. Кроме них есть и другие.

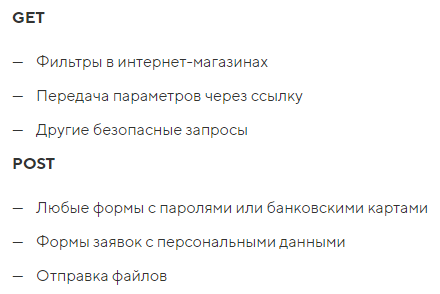
**GET** — метод, который передаёт параметры через ссылку, т.е. URL. Его можно использовать, например, для доступа к указанной странице. Он говорит серверу, что клиент хочет прочитать указанный документ. На практике этот метод используется чаще всего, например, в интернет-магазинах на странице каталога. Фильтры, которые выбирает пользователь, передаются через метод **GET**.

**GET-запрос ограничен по символам.**

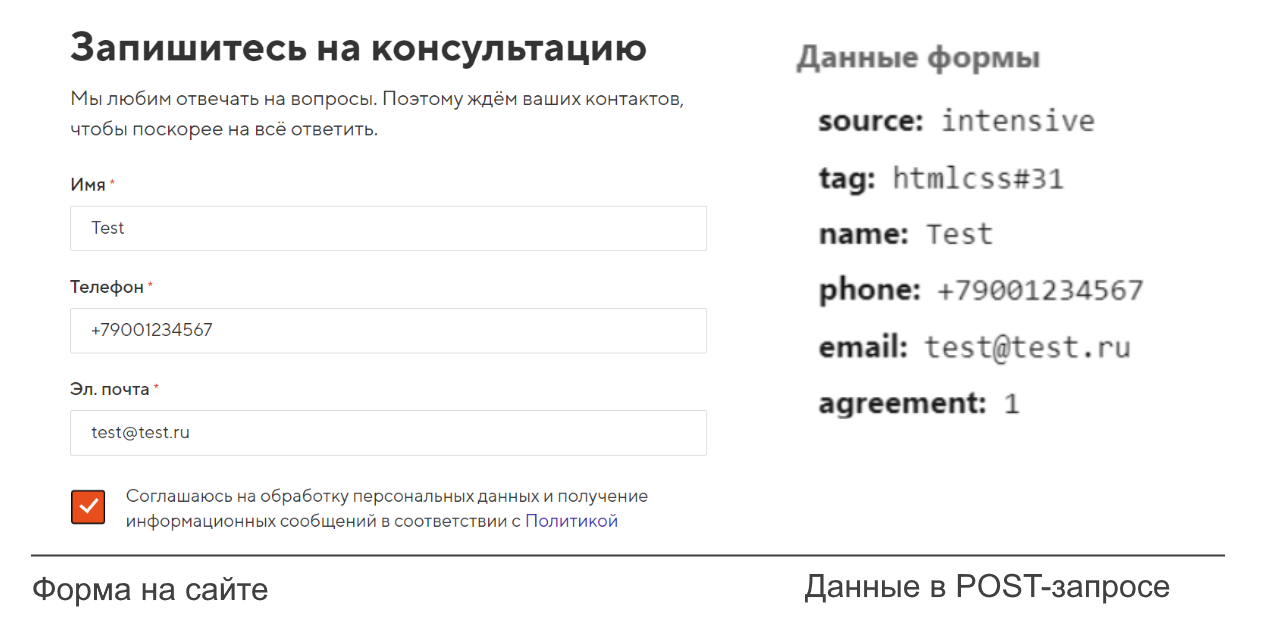
**GET-запросы кешируются на стороне клиента.**

**GET-запросы можно добавлять в закладки.**

**POST** — метод, который передаёт данные уже через тело запроса, и таким образом данные скрываются от посторонних глаз, что делает его более безопасным. Чаще всего с помощью метода **POST** передаются данные из каких-либо инпутов.



<https://guruweba.com/html/metody-get-i-post-ispolzovanie-i-otlichiya/>



**PUT** очень похож на POST, но он используется для замены или обновления каких-то существующих объектов.

*В теле так же передаются ключ: значение*

**DELETE** удаляет указанный ресурс.

*В теле так же передаются ключ: значение*

Также есть:

**HEAD** - запрашивает ресурс так же, как и метод GET, но без тела ответа.

**CONNECT** устанавливает "туннель" к серверу, определённому по ресурсу.

**OPTIONS** используется для описания параметров соединения с ресурсом.

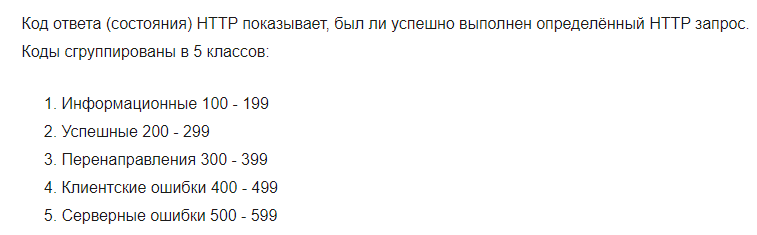
*Не имеет тела*

**TRACE** выполняет вызов возвращаемого тестового сообщения с ресурса.

**PATCH** используется для частичного изменения ресурса. *PATCH отличается от PUT тем, что PATCH частично изменяет существующий ресурс, а PUT – создаёт новый*

**Всего 9.**

**Коды ответа HTTP**

****

200 ОК - означает, что запрос выполнен успешно

301 Moved Permanently - означает, что запрошенный ресурс был окончательно перемещён в URL, указанный в заголовке Location (en-US)

302 Found - означает, что запрошенный ресурс был временно перемещён по адресу, указанному в заголовке Location (en-US)

404 Not Found - указывает, что сервер не может найти запрошенный ресурс

500 Internal Server Error - возникает, когда серверу не удается обработать запрос к сайту

503 Service Temporarily Unavailable - Означает, что по техническим причинам сервер временно не имеет возможности обрабатывать запросы

ЗАГОЛОВКИ ЗАПРОСОВ/ОТВЕТОВ

Заголовок CONTENT-TYPE. Популярные типы

**Content-Type** используется для того, чтобы сообщить клиенту, какой тип данных отдаёт ему сервер или наоборот - сообщить серверу, какой тип данных ему отсылает клиент (в POST или PUT запросах)

Помимо заголовка «**content-type**» существует и заголовок «**accept**». ***accept*** *нужен для того, чтобы сказать СЕРВЕРУ, какой тип данных может принять клиент.*

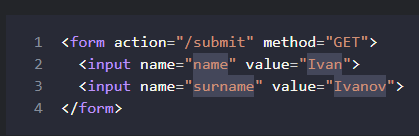
content-type: application/json - говорит серверу, что тело запроса представлено в формате json.

В теле данные передаются данные в json-овском формате ***ключ: значение***

content-type: multipart/form-data — говорит серверу, что это составной тип содержимого, используемый для отправки HTML-форм (например, инпутов) при помощи POST-запроса.

В теле основном через него передают ***картинки, аудио, видео и прочий медиаконтент.***

content-type: application/x-www-form-urlencoded - это стандартное кодирование URL. В основном эта кодировка используется для передачи параметров запроса через строку запроса методом GET и, опять же, в основном, текстовую информацию, в отличие от multipart/formdata, который используется чаще для отправки различных файлов. И параметры передаются в виде пар ***ключ: значение,*** разделённых амперсандом (&) т.е. если у нас есть форма:

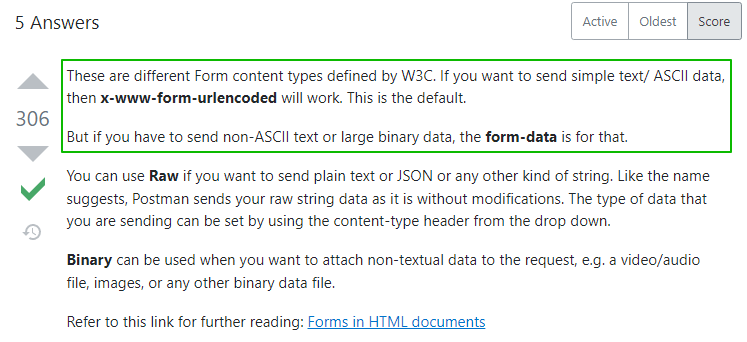


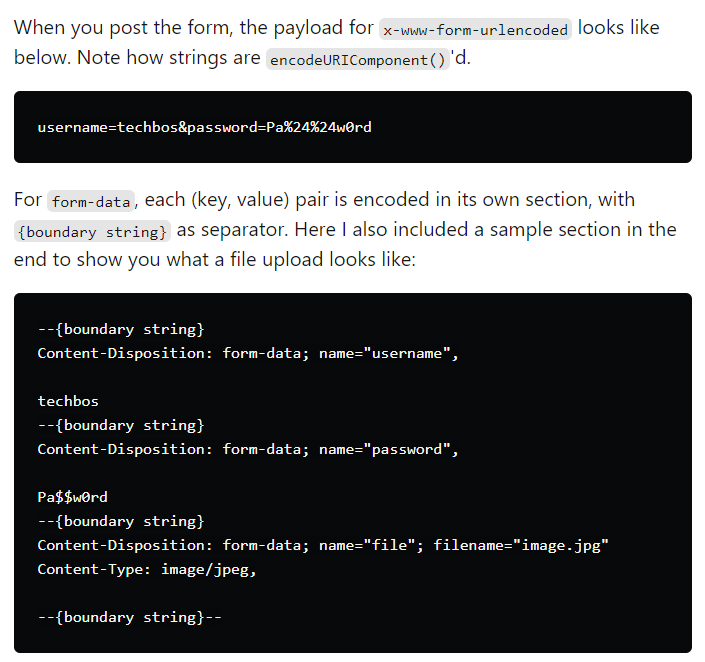
Здесь есть два поля: name=Ivan и surname=Ivanov.

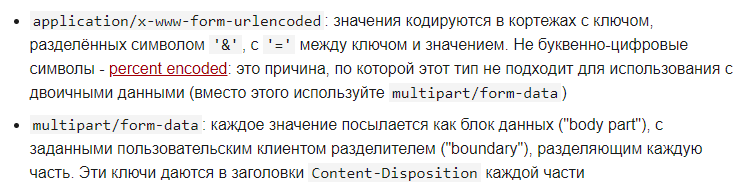
Браузер перечисляет такие пары «имя=значение» через символ амперсанда & и, так как метод GET, итоговый запрос выглядит как ***/submit?name=Ivan&surname=Ivanov***

***В чём отличие*** multipart/form-data от application/x-www-form-urlencoded и когда какой тип используется?

multipart/form-data используется тогда, когда нужно передать бинарные, т.е. двоичные данные или полезные данные значительного размера (всякие медиафайлы типа картинок, аудио и пр.), а application/x-www-form-urlencoded - для передачи параметров в виде пар ключей и значений. В multipart/form-data тоже можно передавать пары ключ-значение, но для каждой из них будет создаваться отдельная секция с заголовком Content-Disposition







**Полная статья (годная): <https://dev.to/getd/x-www-form-urlencoded-or-form-data-explained-in-2-mins-5hk6>**

Годные статьи:

<https://learn.javascript.ru/xhr-forms>

[https://javarevisited.blogspot.com/2017/06/difference-between-applicationx-www-form-urlencoded-vs-multipart-form-data.html#axzz7J9PLuRO6](https://javarevisited.blogspot.com/2017/06/difference-between-applicationx-www-form-urlencoded-vs-multipart-form-data.html" \l "axzz7J9PLuRO6)

<https://medium.com/@rajajawahar77/content-type-x-www-form-urlencoded-form-data-and-json-e17c15926c69>

Заголовок AUTHORIZATION. Популярные типы

***Basic Auth***

Авторизация типа Basic (базовая) осуществляется через ввод данных в форматах:

1. Логин:Пароль

Например:

httpbin.org/basic-auth/Surikat228/228

где Surikat228 - логин, 228 - пароль

Но это применимо к авторизации через API

1. Результирующая строка, закодированная в base64 (YWxhZGRpbjpvcGVuc2VzYW1l). Например:

Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l

***Bearer Auth***

Используется для авторизации по токену, который выдёт пользователю сервер

Пример авторизации: 