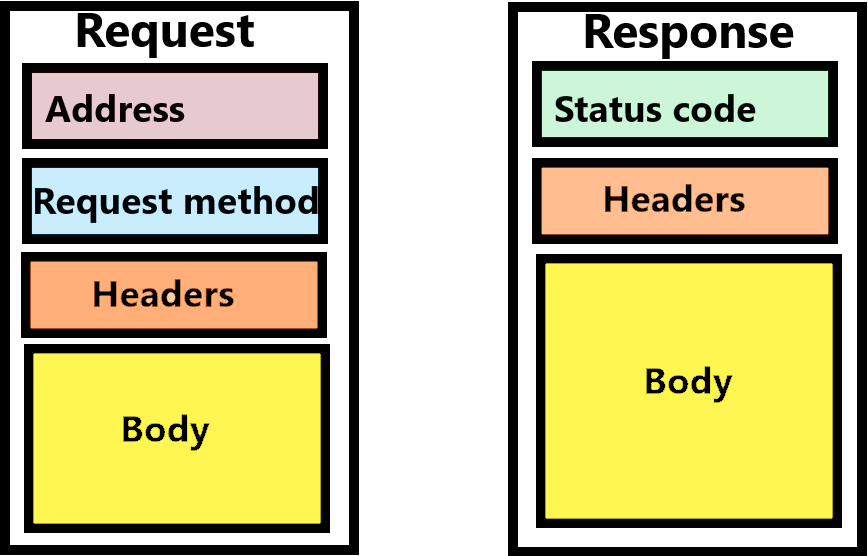
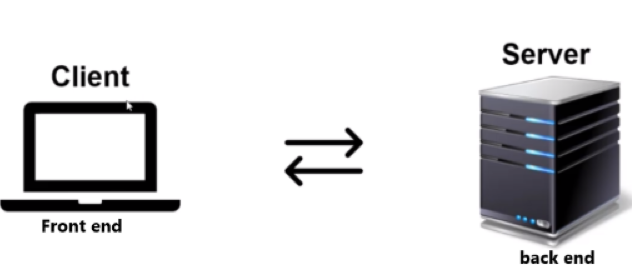
**HTTP -** Протокол прикладного уровня (OSI) передачи данных. 

Используется для передачи данных.

Протокол использует клиент-серверную архитектуру. Говоря о клиент серверной архитектуре, о запросах и ответах сервера, дак вот эти запросы и ответы это HTTP-запросы и HTTP-ответы.



С помощью HTTP - клиент (WEB приложение) посылает запрос (риквест) на сервер на котором находится запрашиваемая инф., сервер получает, обрабатывает и отсылает ответ (респонс).

Другими словами **это протокол, (набор правил) которым необходимо следовать и серверу и клиентам**. в каждом риквесте и риспонсе обязательно должны быть указанные на картинке элементы.

Например: Благодаря протоколу HTTP браузер, получая данные с сервера, знает, как их требуется обработать, и успешно обрабатывает их, показывая Вам запрашиваемую страницу.

**From the perspective of how we interact with APIs in Postman, there is no difference between HTTP and HTTPS. HTTP and HTTPS (HTTP Secure) are both protocols for transferring data over the internet. The main difference between the two is that HTTPS uses an encrypted connection to secure the transmission of data, while HTTP does not. When you make a request to an API in Postman, URL specifies the protocol to use (such as HTTP or HTTPS). Regardless of whether you use HTTP or HTTPS in the URL, the way you use Postman to make the request and inspect the response will be essentially the same.**

**HTTP**

When we call an API (находится на стороне сервера) over the internet we use something that is called **HTTP**. So we need to know what HTTP is, to understand what Web Services are.

Hyper Text Transfer Protocol.

What is regular text? For example “www.google.com”. What does regular text do? nothing. Hyper text meaning that this text can go somewhere else. (hyper means somewhere else). <https://www.google.com>, this thing enables this to go to the google computer (server).

**Из чего состоят HTTP-запросы:**

Само по себе сообщение представляет собой информацию в текстовом виде, записанную в несколько строчек.

What parts in HTTP request? : a) start line (that starts the program, tells it what to do.(**mandatory**).

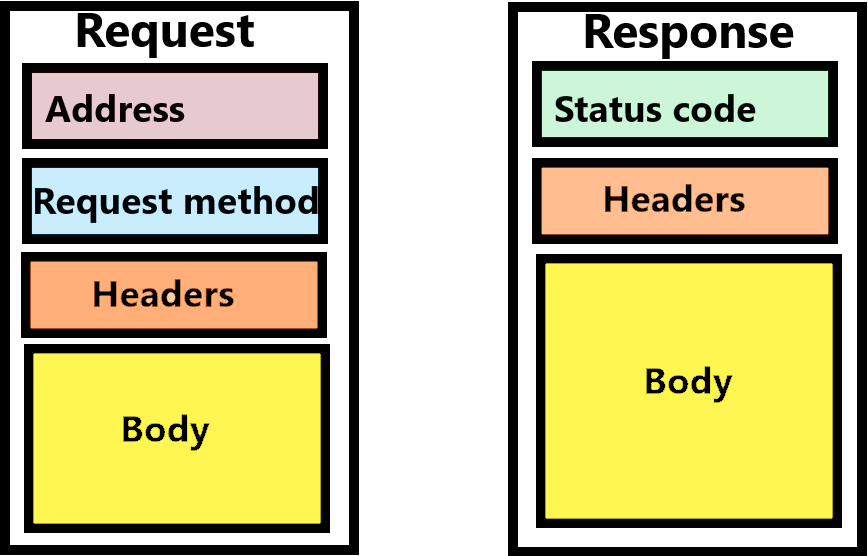
b) headers (that provides additional information other than the start line.

c) blank line

d) body (where we can send information)

What parts in HTTP response? the same 4 parts. What is in this parts:

| HTTP | REQUEST | RESPONSE |
| --- | --- | --- |
| start line | (also called request line)  version,  method(put, get, post, delete),  parameters, folders | (also called response line or status line)  version,  status code |
| headers  (additional info) | For example: 1)-host: [www.google.com](http://www.google.com) <-host is a header. 2) Content-type is also a header that describes the type of content in the body.  2)-token (for example security token) | cookie |
| blank line (used for separating the content) |  |  |
| body | depends of the type of the request | What we requested.  HTML |



**START LINE**

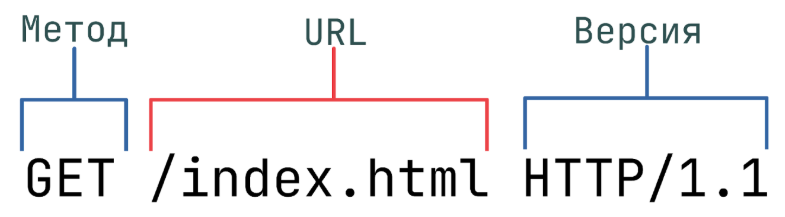
используется для описания версии используемого протокола и другой информации — вроде запрашиваемого ресурса или кода ответа. Как можно понять из названия, ее содержимое занимает ровно одну строчку.

Стартовая строка HTTP-запроса состоит из трех элементов:

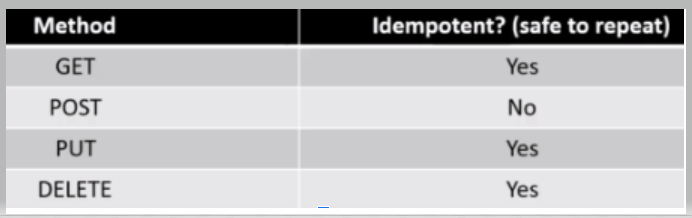
1 - Метод HTTP-запроса

2 - Цель запроса Представлена указателем ресурса URL, который состоит из протокола, доменного имени (или IP-адреса), пути к конкретному ресурсу на сервере. Дополнительно может содержать указание порта, несколько параметров HTTP-запроса и еще ряд опциональных элементов.

* 3 - Версия используемого протокола (либо HTTP/1.1, либо HTTP/2), которая определяет структуру следующих за стартовой строкой данных.



What is idempotence - it means you can repeat something as many times as you want and the results will stay the same, in other words “safe” to repeat.



**Методы HTTP протокола .** (их много тут самые самые)

*Метод HTTP-запроса*. Обычно это короткое слово на английском, которое указывает, что конкретно нужно сделать с запрашиваемым ресурсом.

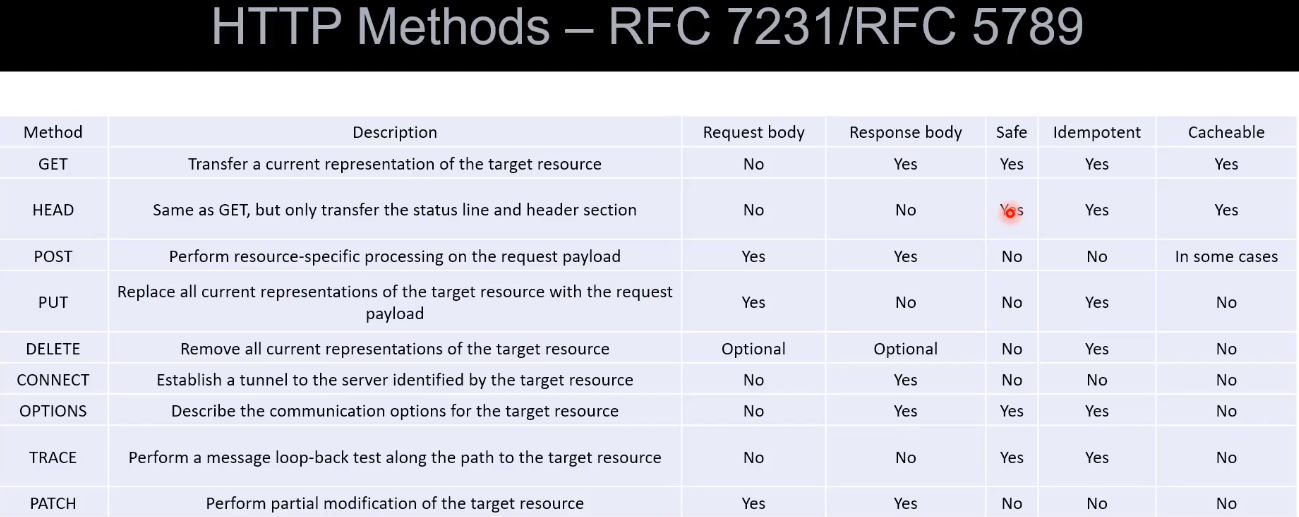
get- получить данные из сервера

you can call get method multiple time with no side effect, (не несёт никаких изменений в API) doesn't have request body parameters in headers and URL

put- внесение изменений в существующую сущность,

post- создание какой то новой сущности, each time you call post new data will be created (это влияет на API) has a request body, parameters in body, headers, url

delete- удаление существующей сущности



Пояснения к таблице с верху:

**Колонка ‘Safe’ - озночает безопасен ли метод. Например если метод не делает никаких изменений на сервере то**

**такой метод называется safe, если же метод вносит какие то изменения то он not safe. К примеру мы можем выполнять какое то обновления на сервере и оно пройдёт успешно но в виду разных причин (slow connection…) ответ от сервера может не прийти и пользователь опять может выполнять тот же запрос хоть по факту изменения внесены.**

**Колонка ‘idempotant’ - no matter how many times the HTTP method is executed, it will not affect the intended impact (не влияет на предполагаемое воздействие) on the server, even though there might be different response. Другими словами сколько раз его не выполняй результат работы этого метода на сервер будет одним и тем же несмотря на то, что ответ от сервера может быть разным.**

**Колонка Cacheable - сервер кэширует запросы и не обращается к конечному запрашиваемому приложению (на стороне сервера) за ответом, а отдаёт уже имеющийся у него ответ тем самым экономит время и ресурсы.**

**get** - fetch the data from the server, data will be about the resource. This method should not have request body and it should have response body

**head** - the same as get but it will not give back the body response (the normal response as get) but only status line and header section, and it will be the same header as the get will bring. It's just the short cut. It might be useful if we use expensive get method, for example if we downloading some beeg file, in that case we might want first ro execute head - it will return only: headers and status code.

**post** - create the resource on the server or update existing resource. Example we create request with post method in order to create, lets say, workspace (in Postman), once we executed it, the postman API created the workspace for us.

But if we execute it again with no change, the postman again will create workspace, iven with the same name and other tegs, but the ID - will be different. So its not safe and not idemportant.

**put** - makes the update to the server. This method is idemportant because it doesn't matter how many times we update the resource, the server will respond the same it will do its work, the output might be different, but the server work in a nutshell will be the same it will update the resource… So as long as there is no side effect on the server as long as the intended effect not changed it is idemportant. And it can not be cashable since it update the resource and always cari the new information in the body.

**options** - usually it is used as pre-flight request to get the communication options for the target resource, and than use this information to make the actual request. So the main difference between options and head is that the options is giving the information about the API, about the request, while the head is giving information about the resource.

**patch** - it is useful if you just want to apdate particular field of the resource. The difference between put and patch, put - will replace the entire resource, even though you want to update just a part of it, but patch will update the part of the resource that you intend to update. We dont need to send the inire resource only the part you want to update.

**URL**

Получение доступа к ресурсам по HTTP-протоколу осуществляется с помощью указателя URL (Uniform Resource Locator). URL представляет собой строку, которая позволяет указать запрашиваемый ресурс и еще ряд параметров.

Использование URL неразрывно связано с другими элементами протокола, поэтому далее мы рассмотрим его основные компоненты и строение:

**Scheme** используется для указания используемого протокола, всегда сопровождается двоеточием и двумя косыми чертами (://).



**Host** указывает местоположение ресурса, в нем может быть как доменное имя, так и IP-адрес. На рисунке ниже это base url.



**Port**, как можно догадаться, позволяет указать номер порта, по которому следует обратиться к серверу. Оно начинается с двоеточия (:), за которым следует номер порта. При отсутствии данного элемента номер порта будет выбран по умолчанию в соответствии с указанным значением Scheme (например, для http:// это будет порт 80).

**Path** - указывает на ресурс, к которому производится обращение. Если данное поле не указано, то сервер в большинстве случаев вернет указатель по умолчанию (например index.html).

**Query String** начинается со знака вопроса (?), за которым следует пара «параметр-значение», между которыми расположен символ равно (=). В поле Query String могут быть переданы несколько параметров с помощью символа амперсанд (&) в качестве разделителя. В примере ниже это query parameters

Не все компоненты необходимы для доступа к ресурсу. Обязательно следует указать только поля Scheme и Host.

URL - Uniform Resource Locator, пример пути запроса:



**Resource** - это например товары в интернет магазине или видео на YouTube.

**Path parameter** - это детализация нашего resource

**Query Parameters** - задают необходимую фильтрацию, указываются (перечисляются) после вопросительного знака

**HTTP HEADERS (in other words headers line)**

Заголовки (Headers) умное определение - это метаданные.

Другимим словами это информация о информации т.е. для корректного ответа нам надо сказать в каком формате мы ожидаем информацию(content type) + идёт доп инф. про клиента, т.е. те же кукис и пр.. т.е. чисто техническая инф между клиентом и сервером.

| a)- General Headers | b)- Request headers, | c)- Response headers. |
| --- | --- | --- |
|  |  |  |

HTTP headers contain metadata in key-value pairs that are sent along with HTTP requests and responses. They can be used to define caching behavior (поведение кэширования), facilitate authentication, and manage session state. HTTP headers help the API client and server communicate more effectively

For example: Content-Type: text/html; charset=UTF-8 ←Context-Type and charset are headers!!!

So basically the headers consist of the metadata that is sent along with the request. For example, for authentication of the account, we may include some sort of the token, that server is expecting from us, and that server will know who we are.

Headers gives some additional information to both: the server, and the client. So that they can communicate in a better way.

The headers can also be used to determine the kind of functionality, the kind of response the server should send back to the client. And there a lot of headers that can be used along with the request or response.

Request headers are sent by the client to the server and contain information and instructions related to the requested resource, while response headers are sent by the server to the client and provide metadata, instructions, and additional information about the response itself.

**Some of the most commonly used request headers are:**

Headers запроса Бывают:

Общего назначения, которые применяются ко всему сообщению целиком.

Заголовки запроса уточняют некоторую информацию о запросе, сообщая дополнительный контекст или ограничивая его некоторыми логическими условиями.

Заголовки представления, которые описывают формат данных сообщения и используемую кодировку. Добавляются к запросу только в тех случаях, когда с ним передается некоторое тело

**Accept**

The Accept header defines the media types that the client is able to accept from the server. For instance, Accept: application/json, text/html indicates that the client prefers JSON or HTML responses. This information allows the server to send a resource representation that meets the client’s needs.

**User-Agent**

The User-Agent header identifies the web browser or client application that is making the request, which enables the server to tailor its response to the client. For instance, if the User-Agent header indicates that the request is coming from the Chrome browser, the server may include CSS prefixes for CSS properties that are compatible with Chrome.

**Authorization**

The Authorization header is used to send the client’s credentials to the server when the client is attempting to access a protected resource. For instance, the client might include a JSON Web Token (JWT) as the value of the header, which the server will then verify before returning the requested resource.

**Content-Type**

The Content-Type header identifies the media type of the content in the request body. For instance, Content-Type: application/json indicates that the request body contains JSON data. This information helps the server successfully interpret and process the payload (полезная нагрузка).

Cookie

The client can use the Cookie header to send previously stored cookies back to the server. The server then uses these cookies to associate the request with a specific user or session. This header plays an important role in delivering personalized experiences, as it enables the server to remember a user’s login state or language preference.

**The most common response headers are:**

Response Headers, или заголовки ответа, используются для того, чтобы уточнить ответ, и никак не влияют на содержимое тела. Они существуют в том же формате, что и остальные заголовки, а именно «Имя-Значение» с двоеточием (:) в качестве разделителя.



**Content-Type**

The Content-Type response header is the counterpart of the Content-Type request header, as it indicates the type of data that the server is sending to the client. The header value typically includes the media type (such as text/html, application/json, image/jpeg, and audio/mp3), as well as any optional parameters.

**Cache-Control**

The Cache-Control header controls caching behavior in the client’s browser or intermediate caches. It defines how the response can be cached, when it expires, and how it should be revalidated. For example, Cache-Control: max-age=3600, public instructs the client to cache the response for a maximum of 3600 seconds (1 hour) and allows caching by public caches.

**Server**

The Server header includes the name and version of the server software that generated the response, as well as information about the server’s technology stack. For instance, Server: Apache/2.4.10 (Unix) indicates that the response was generated by the Apache web server version 2.4.10. It’s important to note that the Server header is informational and doesn’t affect the API’s functionality.

**Set-Cookie**

The Set-Cookie header instructs the client to store a cookie with the specified name, value, and additional attributes, such as expiration, domain, path, and security flags. The client will then include the cookie in subsequent requests in order to facilitate stateful communication and personalized experiences.

Content-Length

The Content-Length header, which specifies the size of the response body in bytes, can help the client anticipate how much data it is going to receive. This improves performance by allowing the client to plan in advance for more efficient memory allocation and data processing.

**BLANK LINE**

Without a blank line, you wouldn't be able to recognize where the header ends, and where the body begins.

**BODY**

Завершающая часть HTTP-запроса — это его тело. Не у каждого HTTP-метода предполагается наличие тела. Так, например, методам вроде GET, HEAD, DELETE, OPTIONS обычно не требуется тело. Некоторые виды запросов могут отправлять данные на сервер в теле запроса: самый распространенный из таких методов — POST.

Тело (Body), обязательно присутствует когда мы используем методы REQUST-а put и post потому, что так как данные методы Request-а что-то создают или меняют(обновляют) нам же надо где то эту информацию “принести” на сервер чтоб он понял не только что мы от него хотим чтоб он сделал но и дали ему материал для этой работы. Нельзя же сказать создай нового пользователя и на этом точка. Надо дать ему имя пользователя, пол, возраст и т. д., т.е. данные. Для этого и существует body в request-е..

**The body is what contains content.**

It could be an image, html web page, video or data. It just contains content that you want to send to the API if you are doing the request, or the content that comes from API if its response from the server.

Theris header is called **Content-type** that indicates the type of data that we are sending or receiving and there's 2 types of **data (**now we are talking about data not video or image**)**that is used more often: **JSON**, **XML** <- these typical ways to send data to the API or receive it back. What I mean by data is **text information**. Very important to know what JSON and XML are.

What is stateless/stateful

Network Protocols for web browser and servers are categorized into two types: Stateless Protocol, and Stateful protocol.

**Stateless Protocols** are the type of network protocols in which Client send request to the server and server response back according to the current state. It does not require the server to retain (сохранить) session information or a status about each communicating partner for multiple requests.

**HTTP** (Hypertext Transfer Protocol), UDP (User Datagram Protocol), DNS (Domain Name System) are examples of Stateless Protocol.

Salient features (отличительные черты) of Stateless Protocols:

1)-Stateless Protocol simplifies the design of the Server.

2)-The stateless protocol requires less resources because system do not need to keep track of the multiple link communications and the session details.

3)-In Stateless Protocol each information packet travel on it’s own without reference to any other packet.

4)-Each communication in Stateless Protocol is discrete and unrelated to those that precedes or follow.

2 ways of doing HTTP: 1) **REST, 2)SOAP**, they are stateless by default, because they use HTTP and HTTP is statless by default it means it does not remember what happened between one http call and the other.

For example: if someone call you on the phone in 1999 you dont know who, you would answer every call in the same manner (Hello…).Same here - server response on every request in the same manner.

**ТИПЫ ДАННЫХ:**

Имеется ввиду типы данных которые могут передаваться по HTTP: XML; JSON (javascript object notation)



**Stateful Protocols**, Its not HTTP !!!.

In Stateful Protocol If client send a request to the server then it expects some kind of response, if it does not get any response then it resend the request. FTP (File Transfer Protocol), TCP, and Telnet are examples of Stateful Protocol.

Salient features of Stateful Protocol:

1)-Stateful Protocols provide better performance to the client by keeping track of the connection information.

2)-Stateful Application require Backing storage.

3)-Stateful request are always dependent on the server-side state.

4)-TCP session follow stateful protocol because both systems maintain information about the session itself during its life.

Comparisons between Stateless and Stateful Protocol:

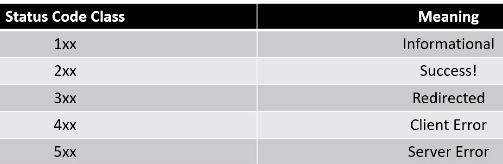
| Stateless | Stateful |
| --- | --- |
| does not require the server to retain the server information or session details. | require the server to save the status and session information. |
| there is no tight dependency between server and client. | there is tight dependency between server and client |
| simplify the server design. | makes the design of server very complex and heavy |
| works better at the time of crash because there is no state that must be restored, a failed server can simply restart after a crash. | does not work better at the time of crash because stateful server have to keep the information of the status and session details of the internal states. |
| handle the transaction very fastly. | handle the transaction very slowly. |
| are easy to implement in Internet. | are logically heavy to implement in Internet. |
| Scaling architecture is relatively easier. | It is difficult and complex to scale architecture. |
| The requests are not dependent on the server side and are self contained. | The requests are always dependent on the server side. |
| To process different information at a time , different servers can be used. | To process every request , the same server must be utilized. |
| Example of Stateless are UDP , DNS , HTTP , etc. | Example of Stateful are FTP , Telnet , etc. |

So then how it is possible that Amazon seems to remember over time the items you put in the cart?

The answer is:

**STATUS CODE MEANING**

If there is no status code it means the request did not even get to the server.



Status codes are grouped in a different categories:

1xx - are termed as informational (for example: the request was received, continuing process). It can be used to tell the client to wait till the server processes the request.

2xx - (Successful): The request was successfully received, understood, and accepted.

3xx - (redirection): further actions need to be taken in order to complete the request.

4xx - client errors! The request contains bad syntax or cannot be fulfilled.

5xx - server error! The server failed to fulfill an apparently valid request.