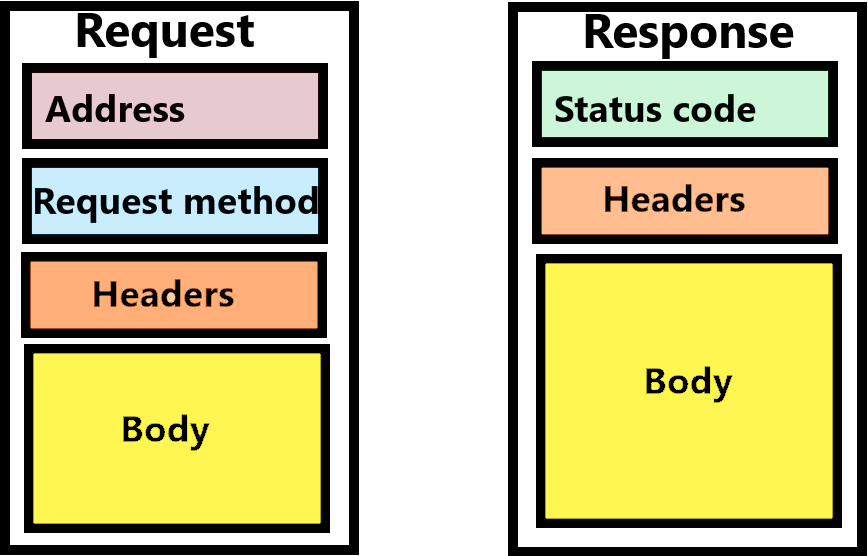
HTTP - к примеру postman отсылает запрос (риквест) на сервер на котором находится запрашиваемая инф., сервер получает, обрабатывает и отсылает ответ (респонс). Для того чтобы такой вид камуникации был доступен используется HTTP. Другими словами **это протокол, (набор правил) которым необходимо следовать и серверу и клиенту** (в нашем случае postman). в каждом риквесте и риспонсе обязательно должны быть указанные на картинке элементы. HTTP протокол (набор правил) в какойто мере можно сравнить с отправкой письма на почте, эта процедура также проводится по определённым правилам (можно сказать по протоколу). Каждое письмо обязательно имеет:

1)- адресс (что соответствует address у HTTP)

2)- почтовые марки (что соответствует Request method)

3)- служебные пометки (Headers)

4)- само письмо (body).

И так же как и с настоящим письмом, если что-то не заполнено письмо не дойдет до получателя, тоже самое и с HTTP.

**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**

Протокол прикладного уровня передачи данных. Используется для передачи данных. Протокол использует клиент-серверную архитектуру. Говоря о клиент сервеной архитектуре, о запросах и ответах сервера, дак вот эти запросы и ответы это HTTP-запросы и HTTP-ответы.

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

**Из чего состоят HTTP-запросы:**

1. Стартовая строка (request line).

а) Методы HTTP протокола .

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…)

б)Цель / путь запроса (как правило URL - Uniform Recorse 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

в) версия HTTP

1. 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. Hedears Бывают:

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

1. 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. И в ответе к стати тоже, я думаю.

**Из чего состоят HTTP- RESPONSE:**

1. Строка статуса (Status line)
2. Версия протокола (обычно HTTP /1.1)
3. Код статуса (Status code)

\*1XX - информационные

\*2ХХ - Успешная операция. In HTTP, a status code of 200 OK indicates that the request was successful and the server was able to fulfill the request. This is the most common status code that is returned in response to a successful request.

\*3ХХ - Перенапрвление

\*4ХХ - Клиентские ошибки

\*5ХХ - Серверные ошибки

с) Пояснения (Status text)

2) Заголовки (Headears)

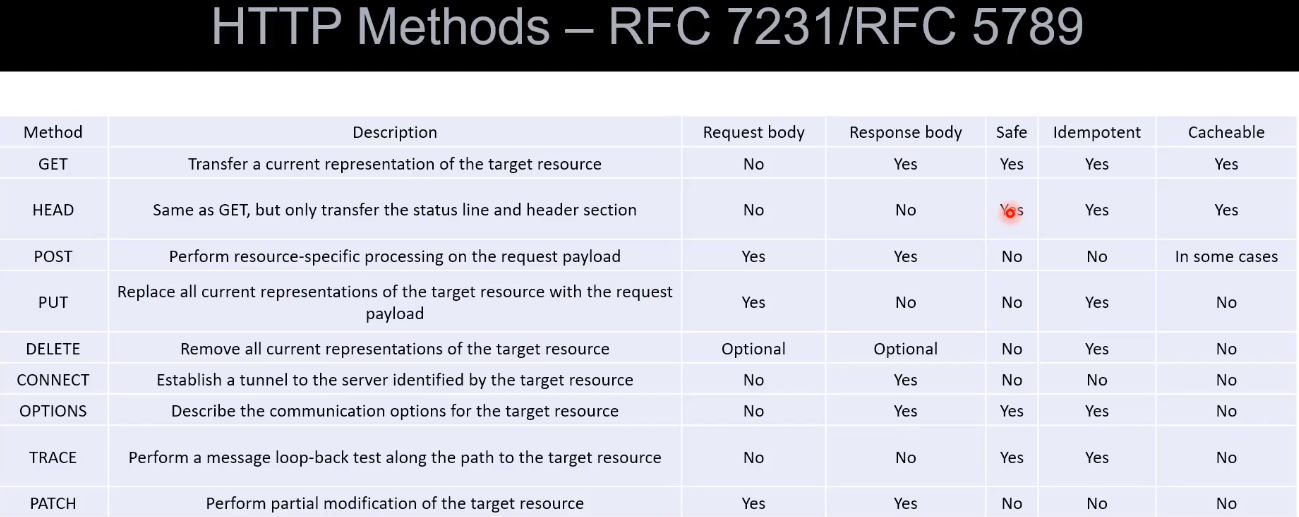
3) Тело (Body)

**ТИПЫ ДАННЫХ:**

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



**HTTP METHODS !**



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.

**delete** - its not difficult, see the table above.

**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.

STATUS CODE !

Status codes are groped 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 - (Succesful): The request was successfully received, understood, and accepted.

3xx - (redirection): further actions needs 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.

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.