**API** - is an interface to some data stored somewhere remotely on a different server. Because API- s are designed to be used by programs they dont have a friendly interface like a website or some app on the phone. API-s can only be used if you write some programming code, or use a terminal to write some complex command. When we are working with API we are working with JSON to transfer the data

**A web API** - is a set of rules and protocols that define how two systems can communicate with each other over the internet. It allows one system (such as a server) to expose (выставлять-передовать) certain functionality or data to other systems (such as mobile application). Web APIs are typically accessed over HTTP or HTTPS, and they can use a variety of data formats (such as JSON or XML) to exchange information. They are often used to enable integrations between different systems, or to allow developers to build applications that interact with a web-based service or platform.

**An API endpoint** - is where an API receives requests. For most services, these endpoints are URLs, just like the ones you use to navigate to a website. Here’s an example of an endpoint URL:

https://api.github.com/repos/torvalds/linux

This endpoint belongs to the GitHub REST API and returns information about a repository as a JSON object.

**Authentication** - (proving that it is you). For example, using a **username** and **password** to get into your email account. It's kind of proving who you are.

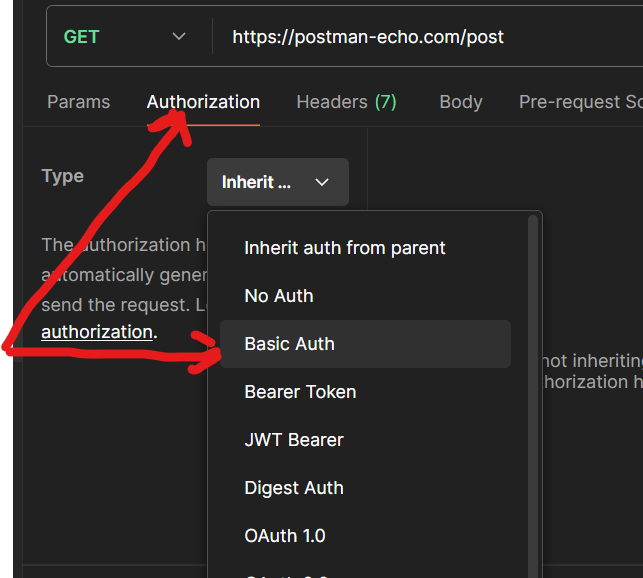
**Authorization** - it means getting limited access to resources. For example theris 2 folders with photos. 1 folder is for public use and 2 folder is for private use only. So other people are only **authorized** to get your photos from public folder #1. They have limited access to resources.

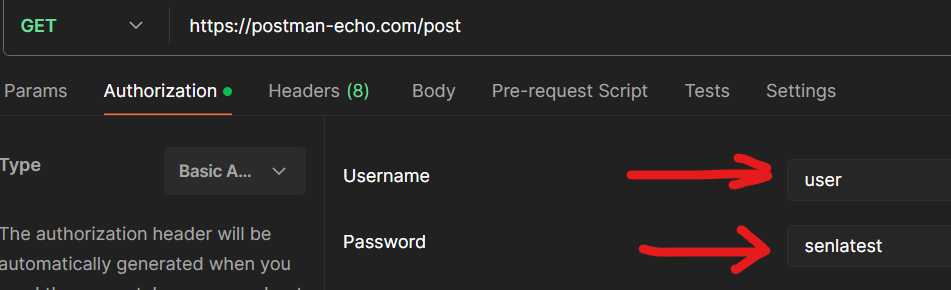
**no Auth (entication)** - this is where **no authentication** and no authorization is involved, example - google search page.In your google search page you call a programm on a google service and it doesn't care who you are.

HTTP authentication schemes

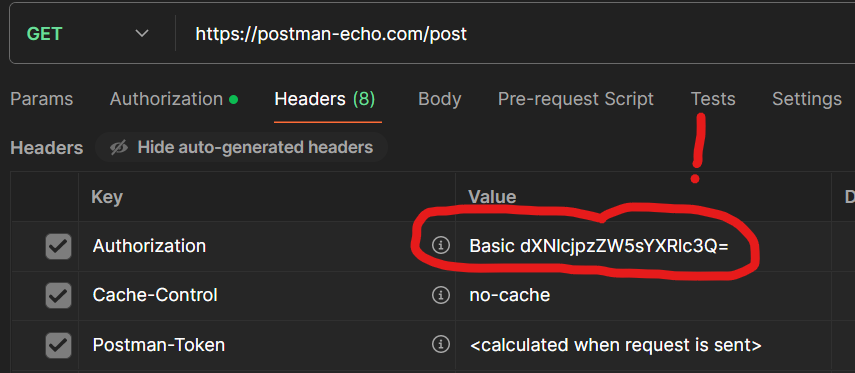
**basic authentication** - this is where you use only credentials to access full resources.So you use **authentication** but no **authorization,** there isno limitation on what you get. The example is email. you get full access to your email. Theris no restrictions. Basic - means simple, authentication - means proving your identity, putting 2 words together = simple way proving your identity. What does this have to do with the API world? For the API they wont to know who you are before they give you access to that API. So now you know what the simplest way of getting the access to API and proving your identity? It is using: **username** and **password**.

The format is: username:password and this is a String that is sent in the header along with the request. The type of String is called Base64 encoding and it is not **safe**. Because the username and password are sent as clear text.

For example we can experiment it in a Postman

← 1) That where it is located in a postman! 

← 2) We need to enter the credentials



← 3) If we open headers we can see the encoded String

that will be sent with the

request

Once the server receives that String it will decode that String in order to authenticate the user and if authentication is successful it will send the response or send the error.

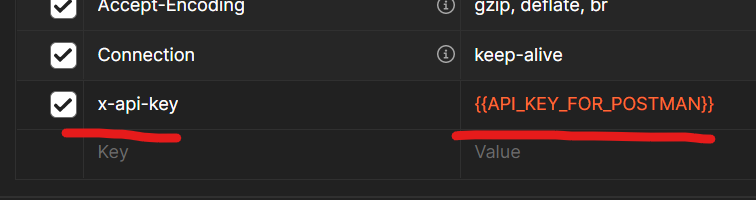
**digest authentication** - digest means something that can be turned into usable format. It has something to do with the secret key that the client knows and API knows, it might use HTTP and not HTTPS, since it has a secret key. I am not about to dig in it since it is not usable for today. The client sends the request to the server without the username and the password. The server then challenges the client by sending certain parameters. The client uses these parameters and encrypts username and password using the hashing algorithm and sends it to the server. The server decrypts the username and password and sends the response if it is successful. The hashing algorithm can be MD5.

The difference between digest and basic authentication is that the username and password is not sent using clear text the username and password sent encrypted using a hashing algorithm. So basically digest authentication is more secure than digest authentication.

**Bearer Token** - this is where somebody use a token (жетон) to get authorized access to resources. The name can be understood as “give access to the bearer of this token.”(«предоставить доступ владельцу этого жетона».) Использование токена не требует от предъявителя доказательства владения. То есть, имея токен, любое приложение может получить доступ к ресурсам.Выдает токен сервер авторизации .Например, какое-либо приложение запрашивает доступ к вашему "Яндекс Диску", запрос направляется в сервис, сервис проверяет статус вашей авторизации, если вы вошли в аккаунт, то показывает страницу с подтверждением на доступ, если нет, отправляет на страницу авторизации (войти), затем возвращает назад, к странице подтверждения прав. Если вы согласны предоставить доступ, нажали "да" или что-то подобное, токен будет отправлен обратно в приложение (callback request). Приложение сохранит токен у себя, и будет его использовать для доступа к вашим ресурсам на "Яндекс Диске". При этом используется только токен, ваши разрешения, имя пользователя и пароль больше не требуются.

In other words…

Bearer is a person or the entity who holds a security token in order to get access to certain resources. Bearer token is a random string that is generated by the server that does not hold any secure data. Whenever client perform login using the username and password, the server sends back bearer token as part of the response, and the client uses bearer token as part of subsequent (последующий) request to the server. The server authenticate the client using bearer token and sends back the response. Most of the time it is part of oAuth 2.0

**Api Key** - it is usually generated during first time login or during signup and it is used as a replacement for username and password. It means that in the HTTP request you don`t have to send username and password, instead you can send API key that can be authenticated by the server and then the server can send back the response. Usually API key can be fetched from the account settings page and often it is possible to delete, regenerate, create multiple API keys. Api key passed as a header (recommended) or query parameter.

← Example from the Postman

**What is OpenID connect** - first of all it is about the authentication (proving your identity) Some of the web applications gives option to sign up with a Google or Facebook account, how is it possible? If we click on these button (for example: join with Google) it will redirect you to google server where you need to provide your google credentials and then google will pass your email, and informs application that user is authenticated (remember that authenticated means - providing your identity using credentials). That approach is called OpenID connect. The application in that case does not need to maintain users accounts.

**Access token** - is used to get the protected resources. The whole point of the OAuth 2.0 is to get an access token to the client so that the client can use it to get authorized access to the resource owner`s resources. It has a String format.

**Refresh token** - is what`s used to get a new access token. Say the access token is good for 1 month, after it expires you use a refresh token to get a new Access token. For that reason your refresh token should last longer than the Access token otherwise there's no point using it. For example, an Access token lasts for 1 month then a refresh token should last for 12 months.It has a String format.

**Two Factor Authentication** - This is where you login not once but twice,

к примеру есть клиент он обращается на сервер за какойто инф., сервер её оддаёт по запросу - это идеальный вариант НО, так на самом деле работать не будет так как , к примеру фронтэнд (клиент) у нас написан на JavaScript а backEnd на Java, данные приложения просто не смогут взаимодействовать. ВОТ для этого и существует API - это технология благодаря которой одно приложение может общаться/взаимодействовать с другим, особенно это актуально когда одно приложение написано на одном стэке технологий а другое на другой технологии. Или ещо пример, мобильная версия приложения обращается на сервер, опять же 2-е разные технологии они напрямую работать не будут, их надо подружить как то и здесь опять же API. или у нас есть приложение и мы хотим использовать это приложение В другом приложении как к примеру какуюто ф-цию (соединить), но при этом оба приложения написаны разными командами и на разных стэках, их можно подружить с помощью API. То есть API - это как внутренний интерфейс приложения для общения но не с конечным пользователем, а в роли пользователя выступает стороннее приложение, и даже возможно наличие у одного приложения несколько API для разных возможных интеграций с разными стэками технологий.