**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** - means providing your identity using credentials (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.

**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**.

**digest authentication** - digest means something that can be turned into usable format. It has something to do with the secret key that 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.

**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.

**OAuth 2.0** - (open authorization)

This is where you give another entity, say an app on your cell phone, access to you resources. For example google Map want to have access to you location, and it ask your permission. and after you granted, it will use only your location(and not the others like photos, calls etc.). So theris authentication (you have to unlock your phone) and authorization(you gave access to use location to google Map app).

**One more example:** theris a web page “photosW '' that is a container for photos, again theris folder with public photos the one that you use for public social networks(instagram, VK, so on) and private folder for you only (may be theris photos that are to private for you and so far you dont want to share it). For you to access your photos (both public and private) you use a username and password. Here are resources and you are the resource owner. Now let's say that you want to give access to somebody else only to your public folder. So you have somebody else that you may call **client** or you may consider it as an **app,** so what you are doing is you are giving this client authorization. Authorization meaning limited access to your recorcess. This client or APP is authorized to use your photos, not to create any photos in there, not to delete, but only to get your public photos. It cant access your private photos because you only authorized to see your public photos. It also cant get your username and password, because if it happened, this client will have access to everything: all photos, ability to create/delete, do anything that it want.

So that the idea of Oauth, meaning other authorization (OAuth). You are giving other client the authorization that gives limited access to some resources. OR: 1)-allows 3-rd party access. 2)-When it says limited access it means-authorized. 3)-It gates limited access to web service, to API.

All of OAuth 2.0 is HTTP request only you can't use anything else

Since theris problem to give your credentials (username+password) to 3-rd party, the third party is given an access token. The 3-rd party uses this access token to get authorized access (that is limited access) to your resources, a HTTP must be used when going through all.

Roles:

- Resource owner (person ho owns the content)

-Resource server, thats what provides photos when given access token (I think its API).

-Client / Application, the 3-rd party that wants access to what you own as the resource owner.

-Authorization server, is what provides access token to the client or APP to get the resources.

**SERVER-**we have a computer out there called Windows server, but this window server is not a server - this is a computer on which there are servers (list of services). And these services are either running or not. And why it is important for this servers / services to run? When its running its looking for something to come in so it can serve. That's why its call server because it waiting for the client to serve. Say, for example client trying to get access to public photo. the API has a running server waiting for request because its running, its always looking for this request. If it's stopped its might be put another server redundant (типа избыточный/добавочный).

**PROTOKOL FLOW FOR OAuth 2.0**

1)- Client requests limited resources from the resource owner. It sends it in the form of an authorization request.

2)- The resource owner says yes you have a grant, we enable you to access the resources

3)-The client than sends an authorization grant to the authorization server that is responsible for resources.

4)-The authorization server than sends back to client an access token.

5)-Than client sends access token to resource server.

6)-The resource server then sends to the client requested content.

**There 4 kinds of authorization grant: (**то что описано во 2-ом пункте выше).

1)-authorization code flow is where the client asks the resource owner for permission through the authorization server.In that case the resource owner has to to log into the authorization server if not already logged into and than say ok. The client does not ask the resource owner directly. This case is the most used.

2)-Implicit is were a call is made for the resources by Ajax or JavaScript.That is code in HTML itself.

3)-Resource owner is where resource owner or somebody who has full permission.

4)-The client credentials method is where the resource owner is never asked for permission

**Flow of authorization code grant** (detailed description of getting code grant)

Before this flow starts the client must register with the authorization server and get a client **ID**.

The user clicks on authorization endpoint and authorization endpoint points to to the authorization server. When the user on the browser goes on the authorization server it brings alone with it client ID and the redirect URI (is where to sent the user back to the client after the user authenticates. So the user and the client starts at the client. The client specifies the link to the authorization endpoint that is the authorization server ← **вот эта часть не понятна**

ok one more time about OAuth - it is when you dont access the API but you give responsibility to another app to run the API. The same as would give permission to the navigator to use your location on the phone.

**The difference between basic authorization and OAuth.** When use basic authorization **you** send password and username to the server. When it is OAuth there is an **app** that has your permission (authorization - limited access) to access the API using your password and username and give you the result, you don't need to do anything.

**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.

API - это описание способов, которыми одна компьютерная программа может взаимодействовать с другой компьютерной программой.

REST API - (Representational State Transfer) архитектурный стиль взаимодействия компонентов распределённого

приложения в сети. Т.Е. это не какая то технология это просто подход, это архитектурный стиль который диктует определённые правила, чтобы одни программы могли по определённым правилам общаться с другими программами, Опять же - это набор правил с помощью которых мы можем выстроить это взаимодействие.

ТРЕБОВАНИЯ К АРХИТЕКТУРЕ REST

**1**) - Клиент - серверное взаимодействие. Важно понимать, что сервер сам по себе никогда никакие запросы посылать никуда не будет. Ключивой момент - сервер отправляет ответ только когда клиент делает запрос. Пример: просматриваем страницу в youTube там определённое количество “лайков” оно не меняется динамически, чтоб увидеть изменения необходимо обновить страницу, а обновление это и есть запрос, по которому и придёт ответ - таже страница с обновлёнными данными (в том числе и с другим числом лайков).

**2**) - Операции над ресурсами. Ресурсами выступает например у нас есть интернет магазин и продукты всевозможные в нём: телефоны, телевизоры и т.д. это и есть ресурсы и мы можем их покупать, ложить в корзину, бронировать, совершать над ними какието действия эти действия и есть операции . Ещо к примеру YouTube, видео на этом сайте это и есть ресурсы которыми можно пользоватся - просматривать , а это и есть операции..

3) - Передача данных с помощью HTTP. HTTP - это основа любого API,

4) - Отсутствие состояния (STATELESS). Например я пользуюсь YouTube и просматривая контент сервер предлагает мне схожие тематически видео, вот этот момент и есть состояние (т.е. мои интересы это и есть состояние), но сам сервер не хранит информацию о клиентских предпочтениях (состоянии), это очень много инф., вместо этого состояние хранится на машине клиента, это называется COOCKIES, но сам принцип видимо по отношению к серверу называется STATELESS

5) - Форматы передачи данных:JSON, XML, любой тип данных можно передать с помощью REST, это не SOAP, где в body есть только XML.

**HTTP**

**любой текст на веб-страничке заключен в специальные теги, которые указывают браузеру какой размер текста использовать, его цвет, расположение на странице (слева, справа или по центру). Это касается не только текста, но и картинок, форм, активных элементов и вообще всего контента, т.е. того, что есть на страничке. Браузер, обнаруживая теги, действует согласно их предписанию, и показывает Вам обработанные данные, которые заключены в эти теги.**

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

**put**- изменение всей сущности, без её удаления

**post**- создание какой то новой сущности

**delete**- удаление существующей сущности,

**patch**-изменение/модернизация части сущности не всей

**Head**-не имеет response or request body, в основном используется для проверки, что endpoint is valid, при этом не ставится цель получения какой-либо инф., просто так сказать лёгкая проверка без нагромождения всякой логикой, чисто посмотреть что endpoint работает.

**!!!** APIs implement request methods differently. While one API may use PUT/PATCH for updating a resource, another API might use POST. For this reason, it is best to check the API documentation first.**!!!**

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 + 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

\*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)

