APIs

Презентация Антона Бузанова

Клиент-сервер

- **Клиент-серверная архитектура** — это такой подход к организации вашего приложения (сайта/бота/etc.), который основывается на обмене данными между компьютером, с которого отправляются запросы к данным (*клиент*), и компьютером, на котором эти запросы обрабатываются (*сервер*).

Запросы

- Запрос — это сформулированное обращение от одного компьютера к другому, у которого *всегда* есть несколько составляющих:

  1. метод/method
  2. текст запроса/query name/endpoint
  3. заголовок/header
  4. тело/body

Методы

- Метод отвечает на вопрос «что сделать?» (как сказуемое). Основных методов существует 4:

- GET — получить данные,
- POST — создать данные,
- PUT — обновить данные,
- DELETE — удалить данные.

- Есть ещё методы CONNECT, OPTIONS, HEAD, TRACE

Текст запроса/endpoint

- Это та часть, что пишется после символа / — /search, /results, /api и т.п.

Заголовки/headers

- В заголовке мы прописываем общие параметры нашего подключения, например:

- **user-agent**: какой мы браузер?
- **charset**: какие символы мы можем распознать?
- **content-type**: какой тип данных мы можем принять? (текст, JSON, картинку)
- **cookie**: сохранённые данные от прошлых запросов, которые нужно фиксировать (логин пользователя, ID конкретной сессии, служебная информация и т.п.)
- другие параметры, сконструированные авторами запроса,

и прочее. Они могут отправляться с дефолтными значениями, но мы можем менять при желании (как мы меняли **user-agent**, чтобы скрыть то, что мы питон).

Тело/body

- Все детали запроса мы передаём в его теле. Все они должны в итоге сформировать пары «ключ-значение», причём значение может быть и строкой, и массивом, и другим объектом.

- В адресной строке параметры и значения расположены за знаком ? после названия запроса, каждая пара отделена знаком амперсанда &. Некоторые методы (например, POST) не пишут параметры в адресную строку

Ответ/response

- Ответ сервера точно так же, как и запрос, состоит из нескольких частей:

- статус-код,
- cookies,
- тело ответа.

Статус-код

- Короткий числовой код, который быстро рассказывает об успешности запроса. Всегда состоит из трёх цифр, первая — самая важная:

- 1 — информационный,
- 2 — всё ок,
- 3 — перенаправление на другой адрес,
- 4 — ошибка со стороны клиента,
- 5 — ошибка со стороны сервера.

- Подробный список можно посмотреть на MDN Net Docs.

Тело ответ/response body

- В теле хранятся те самые запрошенные данные. Это может быть и JSON-объект, и картинка, и видео, и просто строка — ограничиться можно фантазией автора.

API

- ***A****pplication* ***P****rogramming* ***I****nterface*

API

- Вычислительный интерфейс для компонента программного обеспечения или системы, который определяет как другие компоненты или системы могут с ним/ней взаимодействовать.

- какие запросы можно делать
- как их делать
- какие форматы данных использовать
- каким правилам следовать

API vs Application

- Приложение
  - кусок кода, который чё-то делает
  - на код можно посмотреть, понять как он работает

- API
  - способ, с помощью которого код может обращаться к приложению/библиотеке/сервису с просьбой что-то сделать
  - нет доступа к исходному коду: даём запрос на вход – получаем на выходе ответ

API vs интерфейс пользователя

- КОД <--> API <--> интерфейс пользователя

- Есть штуки, у которых есть только API и нет графического интерфейса
- У приложений часто есть UI, но нет API
- Сейчас круто иметь и то, и другое

Браузер

- у него есть интерфейс пользователя, чтобы мы могли взаимодействовать с документами
- у него есть DOM API, чтобы наш код мог взаимодействовать с документом

- NB: DOM – это не Differential **Object** Marking, a Document **Object** Model

API

- данные
  - какие данные можно давать API и какие получать от API
- методы
  - что можно просить сделать?
  - функции, аргументы, ретёрны
- синтаксис
  - как правильно просить?

Вернёмся в прошлое

- HTTP, 90-ые годы

- API для общения с серверами
- 6 главных методов, которые мы уже обсуждали

- OPTIONS (расскажи о себе)
- GET (дай)
- POST (возьми)
- PUT (замени)
- PATCH (поменяй)
- DELETE (удали)

Параметры метода

request

- URL: адрес, который мы просим что-то сделать
- body: что положить в PUT/POST/PATCH
- headers:
  - referer: страница, которая делает запрос
  - accept: что принимать в качестве ответа
  - cookie: кукис

response

- status code: самый норм 200, 301 – редирект, и тд
- body: содержимое ответа
- headers:
  - Content-Type: тип вёрнутого объекта
  - Set-Cookie: опять про кукис

Сериализация

- Сеть передаёт поток байтов/текста
- Каждый HTTP запрос сериализуется в большую строку с особенным синтаксисом
- Веб-сервис получает эту строку и десериализует обратно в запрос
- Сервер затем сериализует ответ
- Ответ затем десериализуется и отдаётся клиенту

request

```
:authority: stackoverflow.com
:method: GET
:path: /questions/9604460/
  how-to-find-out-the-entropy-of-the-english-language
:scheme: https
accept: text/html,application/xhtml+xml,application/
  xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: ru-RU,ru;q=0.6
cache-control: max-age=0
cookie: prov=5612c2f0-6a48-7e45-4cde-fc6ff0169b97
referer: https://www.google.com/
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: cross-site
sec-fetch-user: ?1
sec-gpc: 1
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
  110.0.0.0 Safari/537.36
```

Реальность

- разрабы ленивые
- методы не различают
- думают что только 2 имеют смысл
- всё делают через них

GET

- все аргументы в строке запроса (?)
- легко сериализовать весь запрос в строку
- удобно когда мало коротких параметров

POST

- тело может содержать сериализованные аргументы
- нужно поработать, чтобы нормально построить тело запроса
- необходимо использовать когда много аргументов или они большие (ограничение на длину URL)

HTTP эхо серверы

- Возвращают то, что им послано
- Хороши для тестирования кода
- Можно написать самим, а можно попробовать уже готовые

- https://httpbin.org/anything + ?arg1=1

- можно почитать про то, что бывает например тут

Немного переберёмся в будущее: нулевые

Популярный подход

- в 90-х веб приложения крутились на сервере
- чтобы что-то сделать, юзер отправлял форму
- чтобы дождаться изменений, надо было перезагружать страницу
- браузер просто отображал результат и принимал инпут

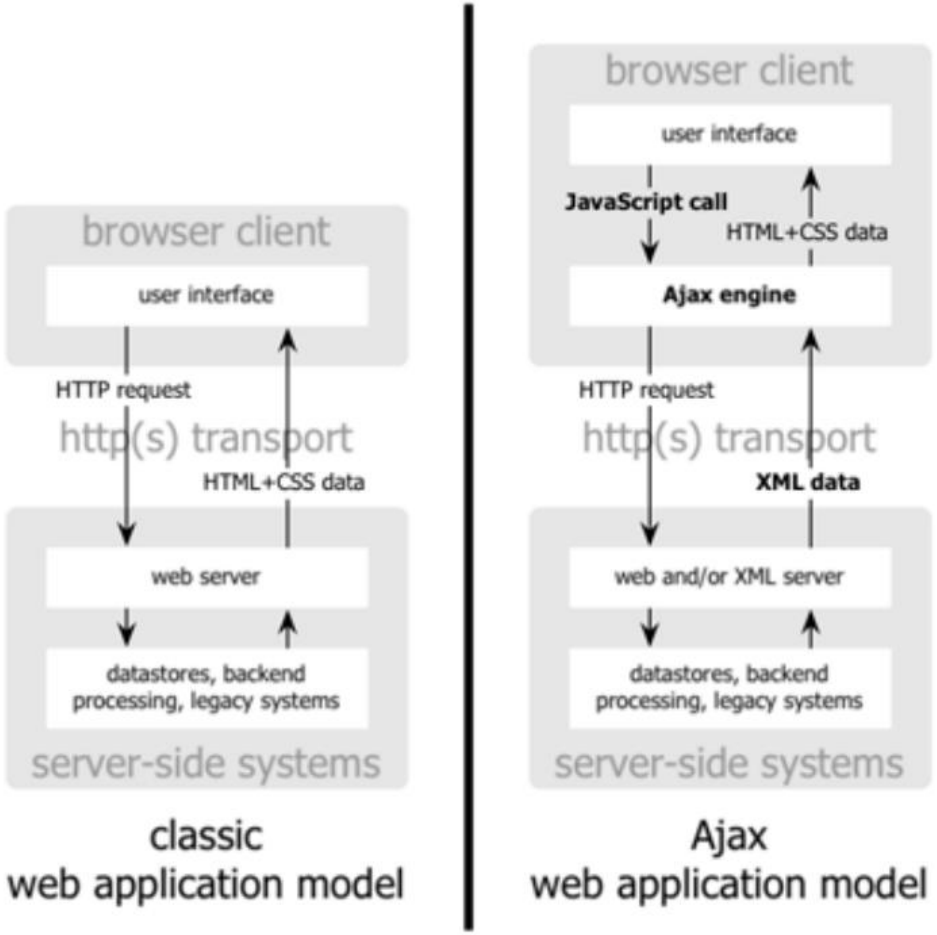
НО

- многие действия лишь немного меняют интерфейс (нет смысла загружать ВСЁ заново)
- клиент вообще-то может и сам что-то вычислять
- зачем лезть каждый раз на сервер?
  - сервер тратит свои мощности на каждого пользователя
  - коннекшн проблемс
  - перезагрузка перебрасывает на верх страницы – тупо

Что из этого вышло?

- Письмо: https://immagic.com/eLibrary/ARCHIVES/GENERAL/ADTVPATH/A050218G.pdf
- The classic web application model works like this: Most user actions in the interface trigger an HTTP request back to a web server. The server does some processing — retrieving data, crunching numbers, talking to various legacy systems — and then returns an HTML page to the client. It’s a model adapted from the Web’s original use as a hypertext medium, but as fans of The Elements of User Experience know, what makes the Web good for hypertext doesn’t necessarily make it good for software applications.

AJAX



- Asyncrounous
- Javascript
- and XML (сейчас JSON, но AJAJ звучит тупо)

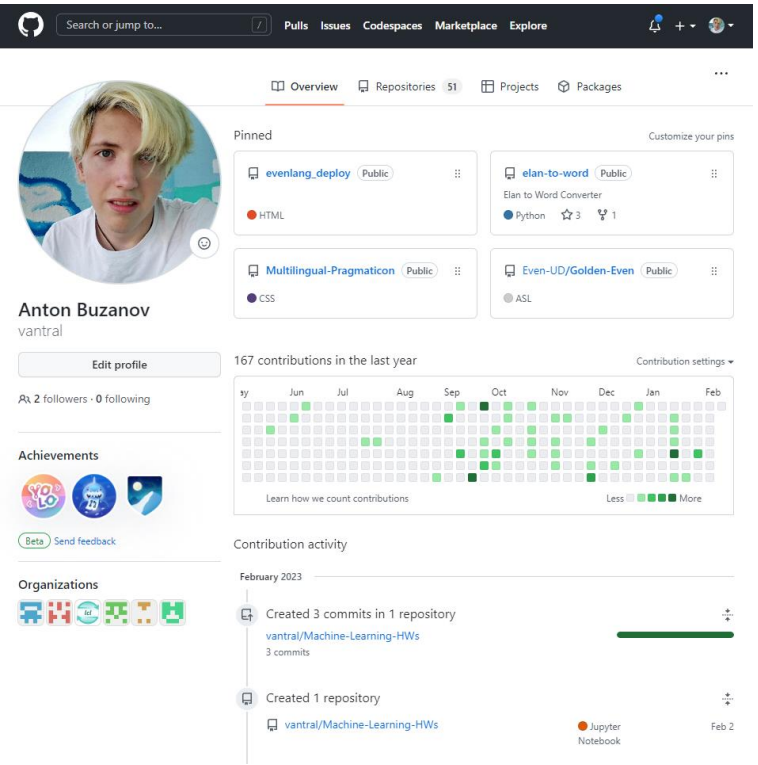
Вернёмся к API

- Изначально сайты создавались для людей
- Но сейчас бывает так, что мы хотим с ними взаимодействовать с помощью кода
- Идея: таскать веб страницы и манипулировать ими

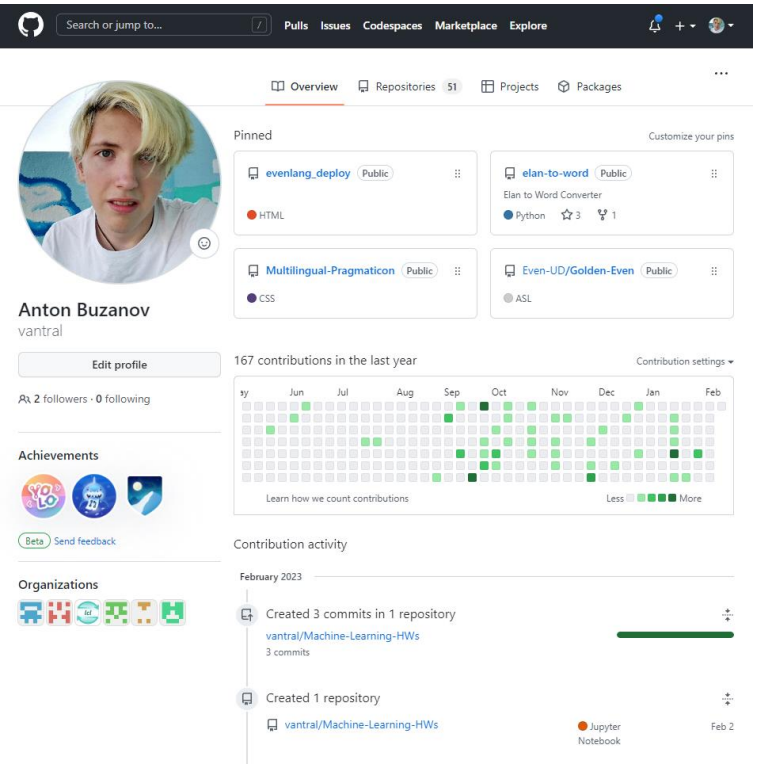
Вернёмся к API

- Изначально сайты создавались для людей
- Но сейчас бывает так, что мы хотим с ними взаимодействовать с помощью кода
- Идея: таскать веб страницы и манипулировать ими
  - большинство страницы сделано для восприятия человеком
  - люди понимают информацию исходя из собственного бэкграунда
  - в HTML много ненужной информации для того чтобы всё нормально выглядело
  - компьютеры хотят что-нибудь машиночитаемое
  - заставлять код имитировать клики мышкой – кринж

Человек vs Машина



Человек vs Машина

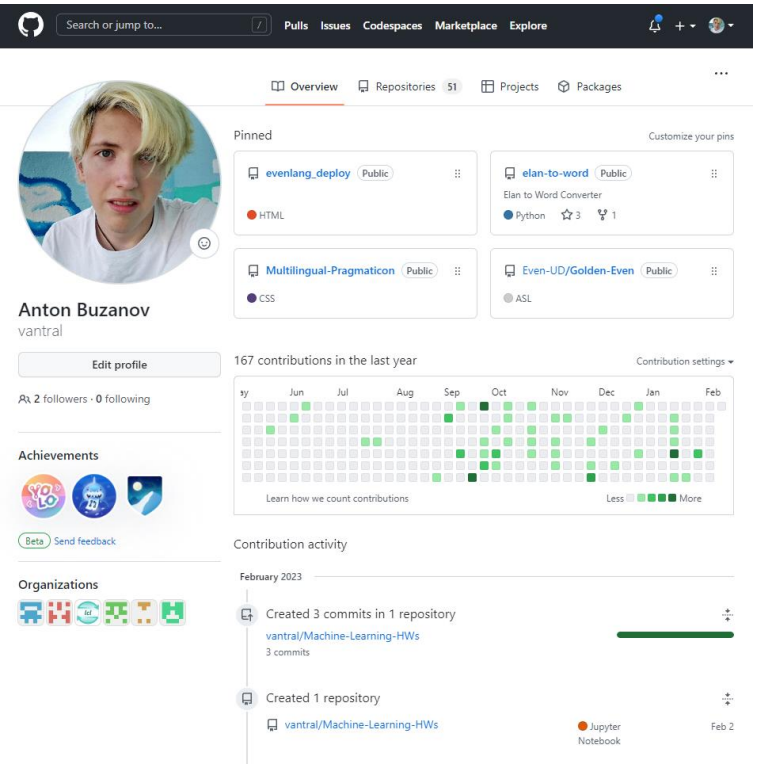



```
<!DOCTYPE html>
<!-- saved from url=(0026)https://github.com/vantral -->
<html lang="en" data-color-mode="light" data-light-theme="light
="system" data-turbo-loaded=""><head><meta http-equiv="Content-
type="text/css">.turbo-progress-bar {
  position: fixed;
  display: block;
  top: 0;
  left: 0;
  height: 3px;
  background: #0076ff;
  z-index: 2147483647;
  transition:
    width 300ms ease-out,
    opacity 150ms 150ms ease-in;
  transform: translate3d(0, 0, 0);
}
</style>

  <link rel="dns-prefetch" href="https://github.githubassets.co
  <link rel="dns-prefetch" href="https://avatars.githubusercont
  <link rel="dns-prefetch" href="https://github-cloud.s3.amazon
  <link rel="dns-prefetch" href="https://user-images.githubuser
  <link rel="preconnect" href="https://github.githubassets.com/
  <link rel="preconnect" href="https://avatars.githubuserconten

  <link crossorigin="anonymous" media="all" rel="stylesheet" hr
  light-719f1193e0c0.css"><link data-color-theme="dark" crosso
  data-href="https://github.githubassets.com/assets/dark-0c343b
```

Человек vs Машина




```
<!DOCTYPE html>
<!-- saved from url=(0026)https://github.com/vantral -->
<html lang="en" data-color-mode="light" data-light-theme="light
="system" data-turbo-loaded=""><head><meta http-equiv="Content-
type="text/css">.turbo-progress-bar {
  position: fixed;
  display: block;
  top: 0;
  left: 0;
  height: 3px;
  background: #0076ff;
  z-index: 2147483647;
  transition:
    width 300ms ease-out,
    opacity 150ms 150ms ease-in;
  transform: translate3d(0, 0, 0);
}
</style>

  <link rel="dns-prefetch" href="https://github.githubassets.co
  <link rel="dns-prefetch" href="https://avatars.githubusercont
  <link rel="dns-prefetch" href="https://github-cloud.s3.amazon
  <link rel="dns-prefetch" href="https://user-images.githubuser
  <link rel="preconnect" href="https://github.githubassets.com/
  <link rel="preconnect" href="https://avatars.githubuserconten

  <link crossorigin="anonymous" media="all" rel="stylesheet" hr
  light-719f1193e0c0.css"><link data-color-theme="dark" crosso
  data-href="https://github.githubassets.com/assets/dark-0c343b
```

api.github.com/users/vantral


```
{
  "login": "vantral",
  "id": 42929116,
  "node_id": "MDQ6VXNlcjQyOTI5MTE2",
  "avatar_url": "https://avatars.githubusercontent.com/u/42929116?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/vantral",
  "html_url": "https://github.com/vantral",
  "followers_url": "https://api.github.com/users/vantral/followers",
  "following_url": "https://api.github.com/users/vantral/following{/other_user}",
  "gists_url": "https://api.github.com/users/vantral/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/vantral/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/vantral/subscriptions",
  "organizations_url": "https://api.github.com/users/vantral/orgs",
  "repos_url": "https://api.github.com/users/vantral/repos",
  "events_url": "https://api.github.com/users/vantral/events{/privacy}",
  "received_events_url": "https://api.github.com/users/vantral/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Anton Buzanov",
  "company": null,
  "blog": "",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": null,
  "twitter_username": null,
  "public_repos": 47,
  "public_gists": 3,
  "followers": 2,
  "following": 0,
  "created_at": "2018-09-03T07:43:42Z",
  "updated_at": "2023-01-26T09:23:11Z"
}
```

Application-Specific APIs

- HTTP API это просто – создай, прочитай, обнови, удали
- Каждое приложение умеет что-то своё
  - login, sendMessage, и тд
- Обычно один URL на метод
  - чтобы получить инфу по юзеру: https://api.github.com/user
- Аргументы либо в пути GET или в качестве параметров при POST
  - /user/vantral или /user?vantral
- Тело ответа (response body) содержит результат
- HTTP статус-код может подсказывать о каких-то исключениях
- JSON – основной формат ответа
  - и иногда и для запроса

RESTful APIs

проблема

- на многих классических серверах сервер запоминает состояние сессии с клиентом:
  - кто юзер
  - как далеко продвинулся в приложении
  - что будет дальше
- большая часть информации становится ненужной как только юзер покидает сервер
- серверу нужно много хранить
  - потому что сервер не знает что юзер закрыл страницу
  - нужно хранить всё про всех юзеров, которые когда-то посещали это сервис

RESTful APIs

проблема

- на многих классических серверах сервер запоминает состояние сессии с клиентом:
  - кто юзер
  - как далеко продвинулся в приложении
  - что будет дальше
- большая часть информации становится ненужной как только юзер покидает сервер
- серверу нужно много хранить
  - потому что сервер не знает что юзер закрыл страницу
  - нужно хранить всё про всех юзеров, которые когда-то посещали это сервис

решение

- REpresentational State Transfer
- Хранит всё, что потеряет важность после закрытия страницы, на клиенте
- Каждый вызов метода должен включать всё нужное, чтобы сервер ответил
- Сервер всё ещё хранит постоянное состояние:
  - например, то сколько вы потратили денег в интернет-магазине

Security & Authorization

- Зачем?

Security & Authorization

- Зачем?
- Сервер это конечный ресурс, нужно ограничивать использование
- Сервер содержит информацию, которую нельзя получать всем

Авторизация на стороне клиента?

- Могут ли клиенты быть ответственны за авторизацию?
- скажем, ваш код сам отказывается посылать запросы, которые не стоит делать

Авторизация на стороне клиента?

- Могут ли клиенты быть ответственны за авторизацию?
- скажем, ваш код сам отказывается посылать запросы, которые не стоит делать
  - доверия клиентам быть не может
  - а вдруг это мошенники!
  - могут менять данные
  - могут менять код
  - могут менять что-то в браузере
  - доверия клиентам быть не может!
- Клиенты могут ПЫТАТЬСЯ, но одобрение запроса всегда дело сервера

Авторизация с токенами

- Клиенту выдаётся какой-то токен, который говорит, что этот клиент – авторизованный и ему можно доверять
  - классические пример: (логин и) пароль
  - идентифицирует клиент однозначно (если на шарить аккаунт на всех своих друзей)
  - сервер понимает, авторизованный ли перед ним юзер
- REST: токен отправляется с каждым запросом!
- Если токен не подделываемый, то сервер понимает, что клиент может делать запрос
- Если токен украден, то вор выдаёт себя за хорошего юзера

Типы веб токенов

- В URL-ах
- https://some-site.ru/?user=vantral&pwd=12345678
- ПЛОХО в общем случае. Почему?

Типы веб токенов

- В URL-ах
- https://some-site.ru/?user=vantral&pwd=12345678
- ПЛОХО в общем случае. Почему?
  - URL-ы копируются, сохраняются, шарятся
  - Токены утекают
  - Но может быть нормально. В каких случаях?

Типы веб токенов

- В URL-ах
- https://some-site.ru/?user=vantral&pwd=12345678
- ПЛОХО в общем случае. Почему?
  - URL-ы копируются, сохраняются, шарятся
  - Токены утекают
  - Но может быть нормально. В каких случаях?
    - инвайты новых юзеров, восстановление пароля
    - потому что они одноразовые

Типы веб токенов

- Cookies
- сервер посылает Set-Cookies: заголовок в ответе от домена
- клиент посылает Cookie: заголовок в запросе к домену
- REST: клиент хранит состояние, всегда посылает токен на сервер

Типы веб токенов

- HTTP Basic Auth header
  - авторизация: просто user:password
- HTTP Digest Auth header
  - авторизация: хэш (user:passwd:nonce)
  - хэширует логин и пароль с помощью одноразового кода для безопасности
- HTTP Bearer Auth header
  - токен это не username/password
  - случайная строка, которую сервер способен опознать: абровауц6 Ощщщй
  - авторизация: абровауц6 Ощщщй
  - сервер больше не знает, кто вы, но знает, что вам сюда можно

Как это всё реализуется?

- URL и cookies браузер может и так смотреть
- для всего остального придётся писать свой JavaScript

Как получить токен?

- Как получить токен изначально?
  - доказать что ты это ты, чтобы получить токен
  - сервер посылает токен клиенту
  - токен может доказать, что ты это ты, но
- изначальная аутентификация должна быть защищена:
  - <место для тейка про кибер-безопасность>

  - НЕ ХРАНИТЕ ТОКЕНЫ НА ГИТХАБЕ

Ещё

- Бывает так, что сайту не важно, что это именно вы используете его токен
- А важно то, сколько запросов вы делаете!

Ещё

- Не все API созданы для внешнего пользования

- например, есть сеть кинотеатров КАРО. Давайте перейдём на их сайт: https://karofilm.ru/