ADT

Web API methods

Sipos Miklós

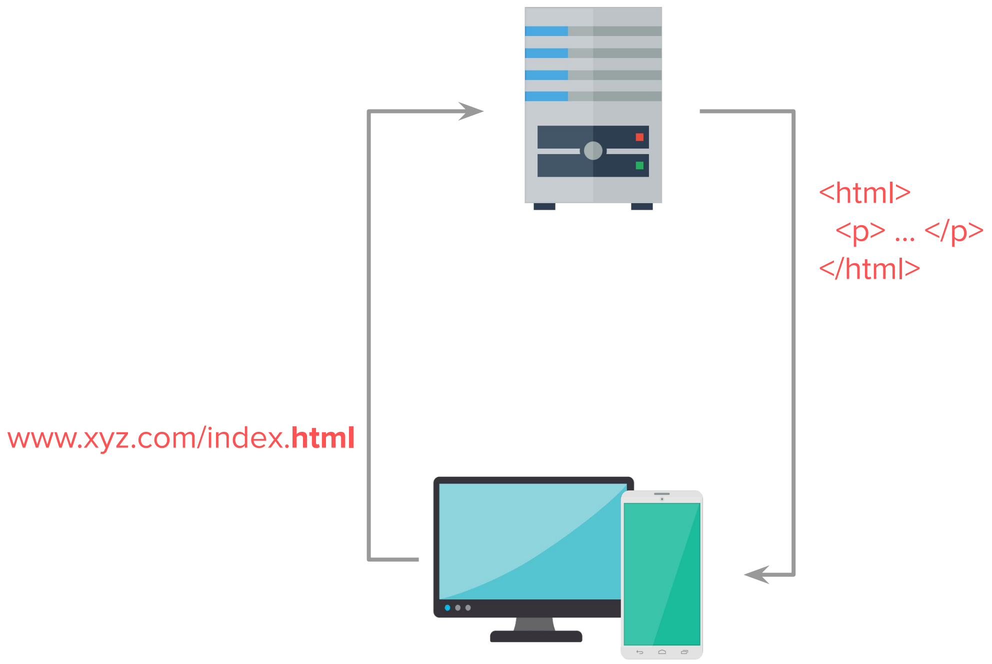
Óbudai Egyetem Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet
2021

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Contents

Server-client connection

Server-client connection



Process:

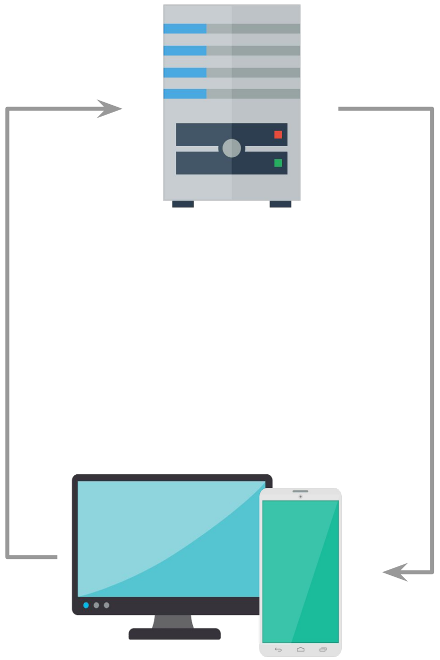
- the request goes to the server (**request**)
- for what the server "looks up" the corresponding file (eg. index.html)
- and then sends it back to the client (**reply**)

request - reply based communication

reply = response

GET & POST

- index.html / index.php
- GET ➜ parameters visible in URL
- POST ➜ parameters NOT visible in URL
- (DELETE & PUT)



www.xyz.com/users.php
?
action=listUsers
&
filter=men

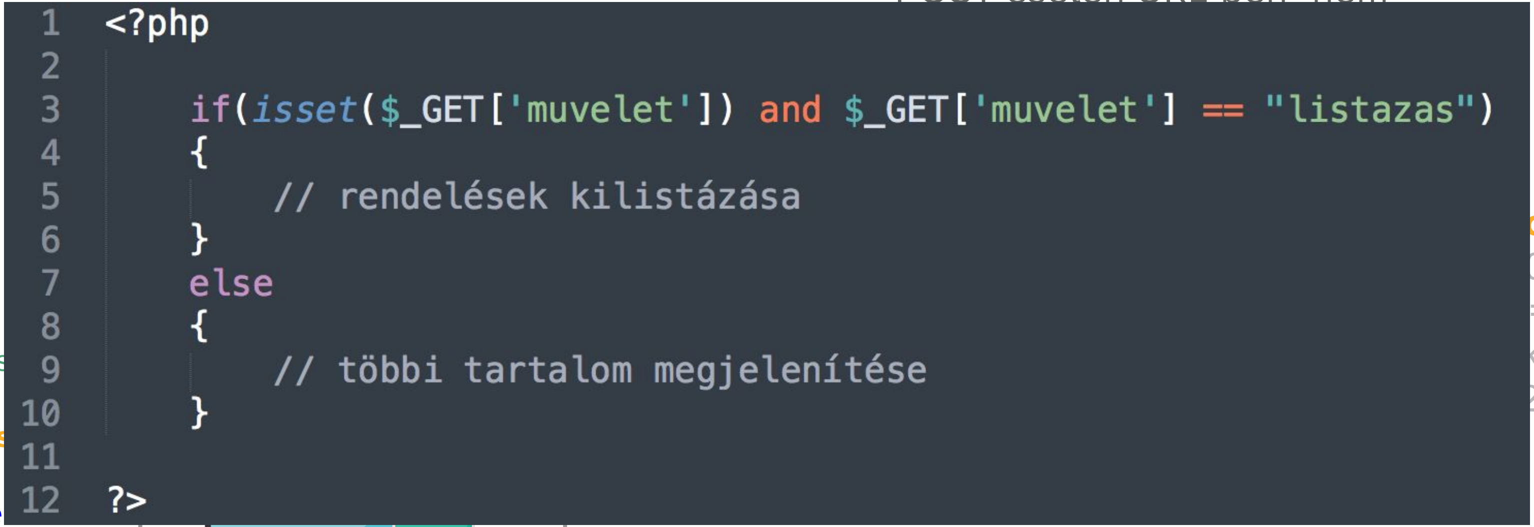
<html>
  <p> … </p>
</html>

html reply???

https://www.google.com/search?q=logo&sxsrf=ALeKk0166hIconsK2WTQCRydcfvi0uhOCg:1600501643890&source=lnms&tbm=isch&sa=X&ved=2ahUKEwje5sif3fTrAhWOk4sKHU0hDaUQ_AUoAXoECA4QAw&biw=2057&bih=929

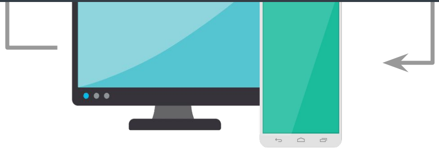
GET & POST

- index.html / index.php szerepe
- GET esetén URL-ben látszódik
- POST esetén URL-ben *nem*



```php
<?php

    if(isset($_GET['muvelet']) and $_GET['muvelet'] == "listazas")
    {
        // rendelések kilistázása
    }
    else
    {
        // többi tartalom megjelenítése
    }

?>
```

www.xyz.com/us
?
action=lis
&
filter=me



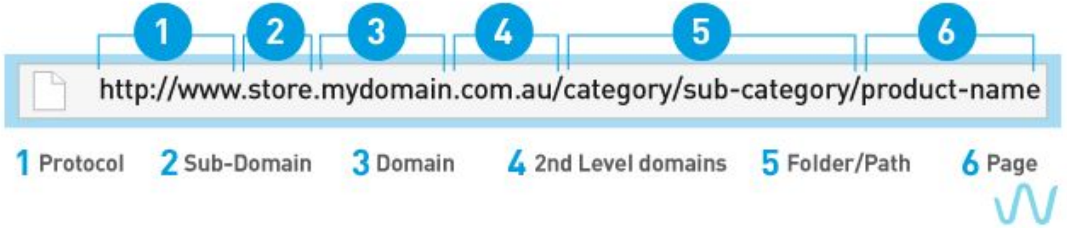
o&sx
0uhO
=isch
x4sK
2057

Friendly URLs

- can be useful for SEO
- in the **background** it is still a GET request
- conversion can be made with eg. htaccess

✓ www.domain.com/home/web-page

✗ www.domain.com/home?id=12654



Web services

input: data send by GET / POST

output: HTML code

many cases we would like to get some dynamic data (like searching, filtering for something), instead of a static like index.html

we would like to ask the male users who are older than 30
➜ we can pass everything based on that query as GET parameters

admin.php ? action=getUsers & age=30 & gender=male

but in this case the output will not be simple HTML ➜ data encoding problem

Data encoding (XML, JSON)

Data encoding

encoding the *bool* variable type:

- 0 - 1
- false - true
- False - True
- FALSE - TRUE
- F - T

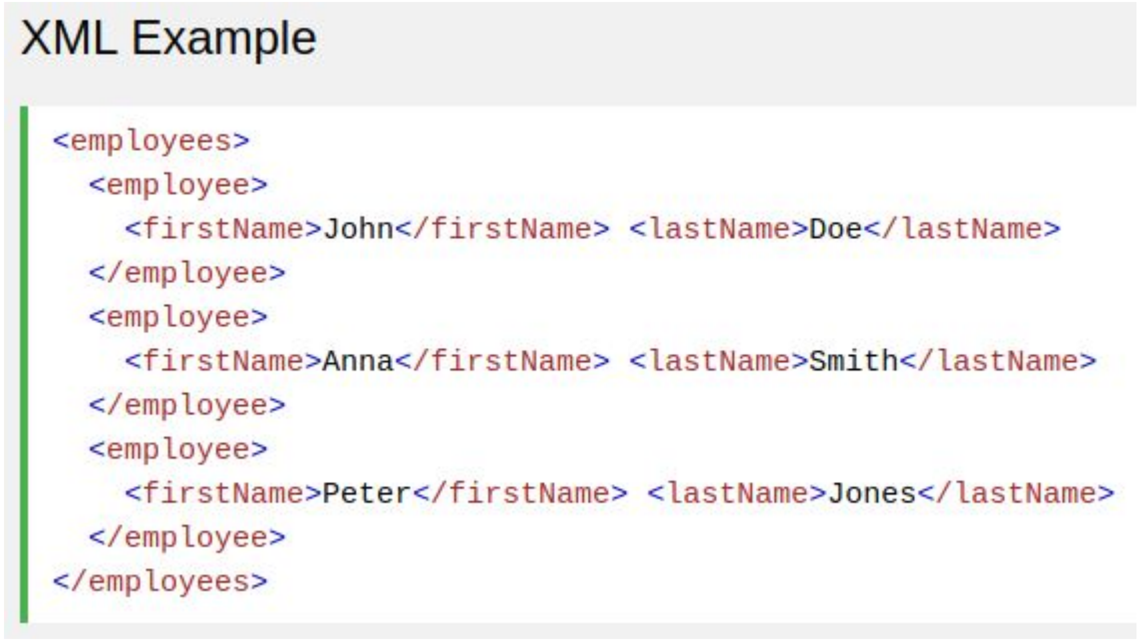
➜ Absolutely not trivial!

We don’t cover this topic deeply in this semester but **what is important**: the encoding’s method and the decoding’s method must be the same!

Data encoding / XML

“encode as string” (UTF-8)

eXtensible Markup Language



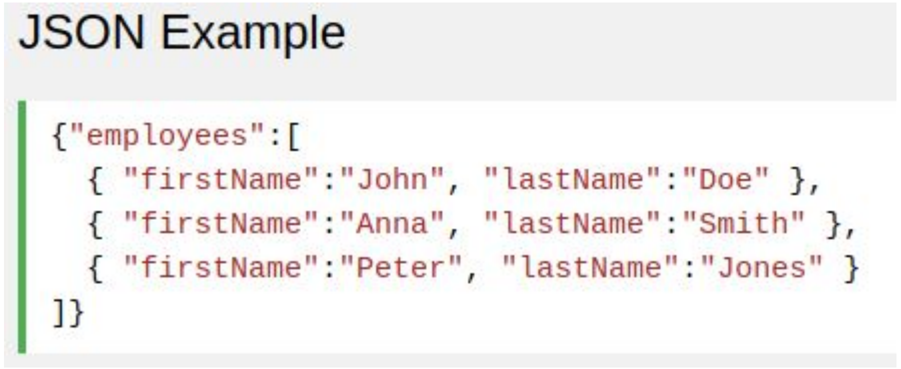
XML Example

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Data encoding / JSON

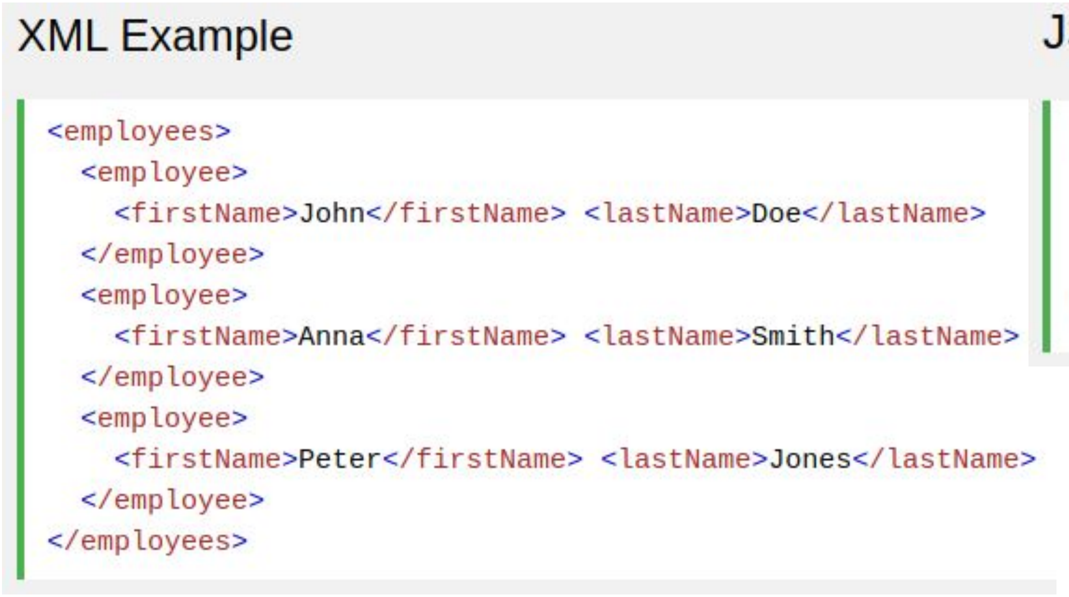
“encode as string” (UTF-8, but not obligatory)

JavaScript Object Notation


JSON Example

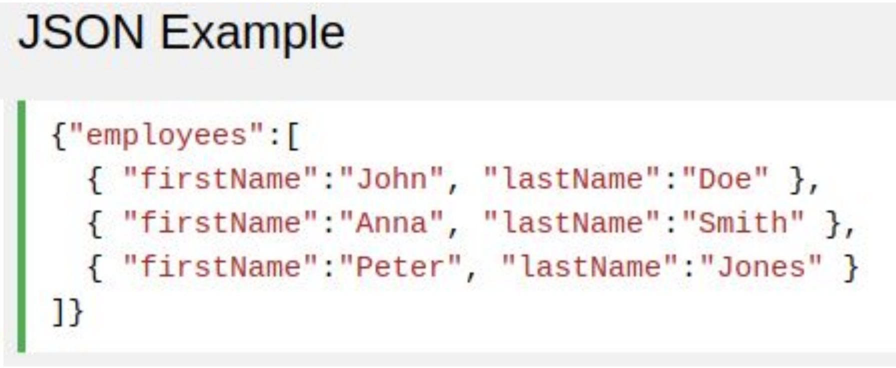
```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Data encoding / JSON vs XML



XML Example

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```



JSON Example

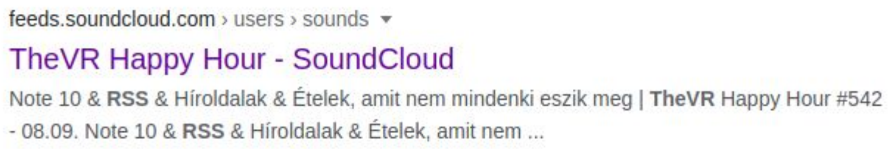
```json
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Storing structured data in text format

Size is important because it has to be forwarded through the network as plain text

XML

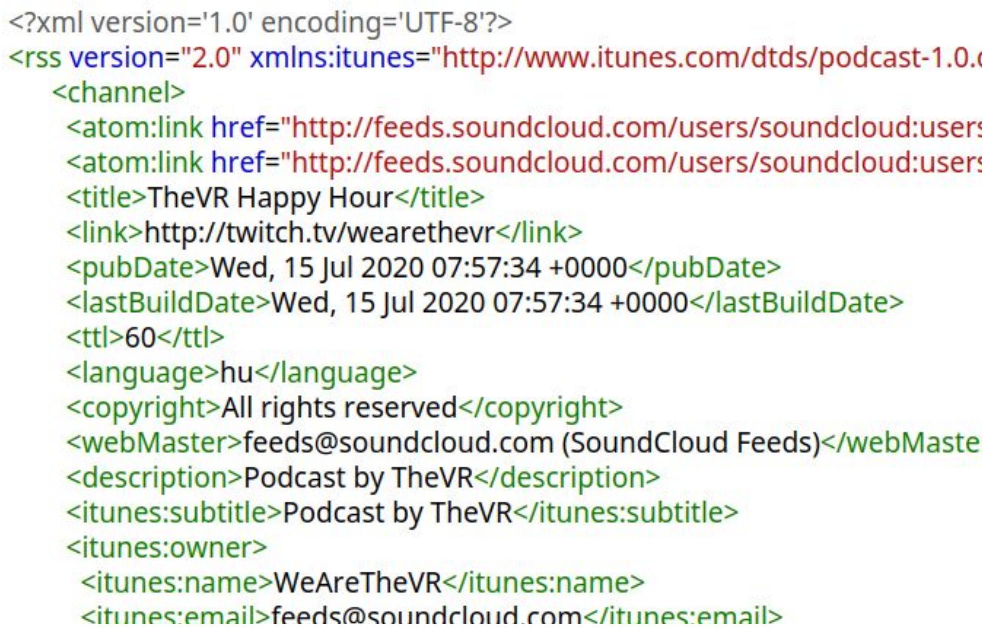
RSS



feeds.soundcloud.com › users › sounds

TheVR Happy Hour - SoundCloud

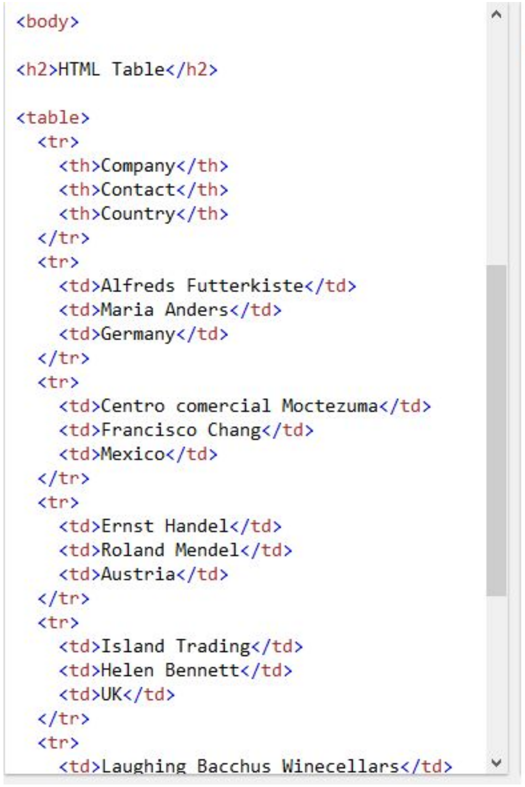
Note 10 & **RSS** & Híroldalak & Ételek, amit nem mindenki eszik meg | **TheVR** Happy Hour #542 - 08.09. Note 10 & **RSS** & Híroldalak & Ételek, amit nem ...



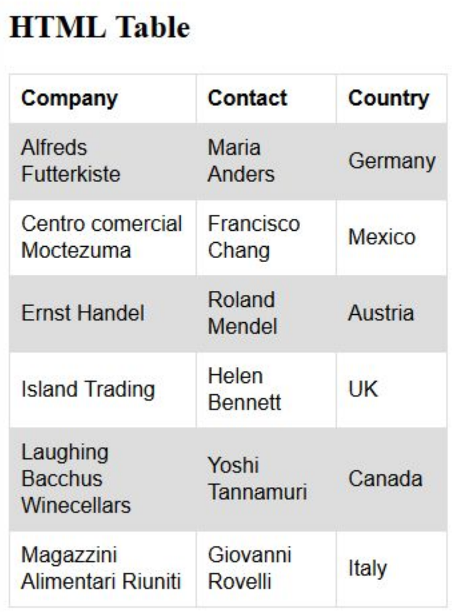
```
<?xml version='1.0' encoding='UTF-8'?>
<rss version="2.0" xmlns:itunes="http://www.itunes.com/dtds/podcast-1.0.
   <channel>
    <atom:link href="http://feeds.soundcloud.com/users/soundcloud:users
    <atom:link href="http://feeds.soundcloud.com/users/soundcloud:users
    <title>TheVR Happy Hour</title>
    <link>http://twitch.tv/wearethevr</link>
    <pubDate>Wed, 15 Jul 2020 07:57:34 +0000</pubDate>
    <lastBuildDate>Wed, 15 Jul 2020 07:57:34 +0000</lastBuildDate>
    <ttl>60</ttl>
    <language>hu</language>
    <copyright>All rights reserved</copyright>
    <webMaster>feeds@soundcloud.com (SoundCloud Feeds)</webMaste
    <description>Podcast by TheVR</description>
    <itunes:subtitle>Podcast by TheVR</itunes:subtitle>
    <itunes:owner>
     <itunes:name>WeAreTheVR</itunes:name>
     <itunes:email>feeds@soundcloud.com</itunes:email>
```

http://feeds.soundcloud.com/users/soundcloud:users:281745775/sounds.rss

HTML



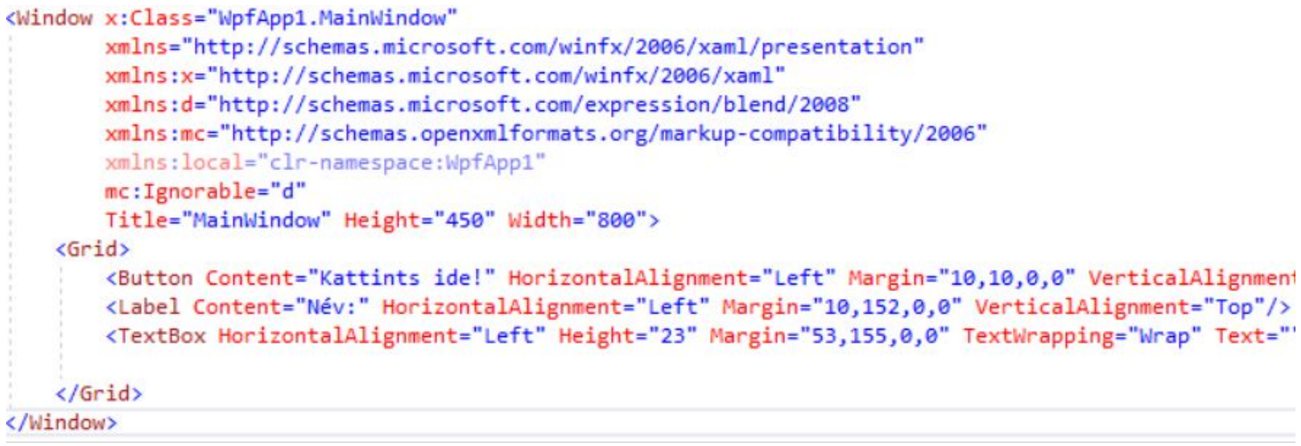
```
<body>

<h2>HTML Table</h2>

<table>
  <tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
  </tr>
  <tr>
    <td>Alfreds Futterkiste</td>
    <td>Maria Anders</td>
    <td>Germany</td>
  </tr>
  <tr>
    <td>Centro comercial Moctezuma</td>
    <td>Francisco Chang</td>
    <td>Mexico</td>
  </tr>
  <tr>
    <td>Ernst Handel</td>
    <td>Roland Mendel</td>
    <td>Austria</td>
  </tr>
  <tr>
    <td>Island Trading</td>
    <td>Helen Bennett</td>
    <td>UK</td>
  </tr>
  <tr>
    <td>Laughing Bacchus Winecellars</td>
```



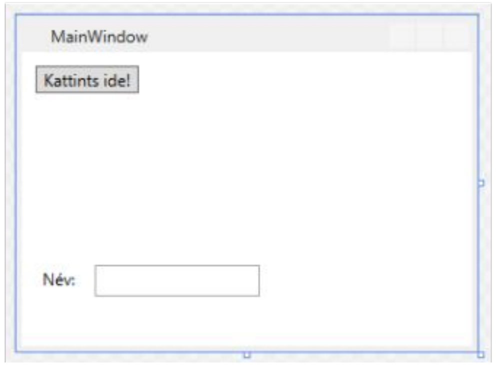
HTML Table

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

XAML



```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp1"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Button Content="Kattints ide!" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignmen
        <Label Content="Név:" HorizontalAlignment="Left" Margin="10,152,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="53,155,0,0" TextWrapping="Wrap" Text="

    </Grid>
</Window>
```



XML vs JSON

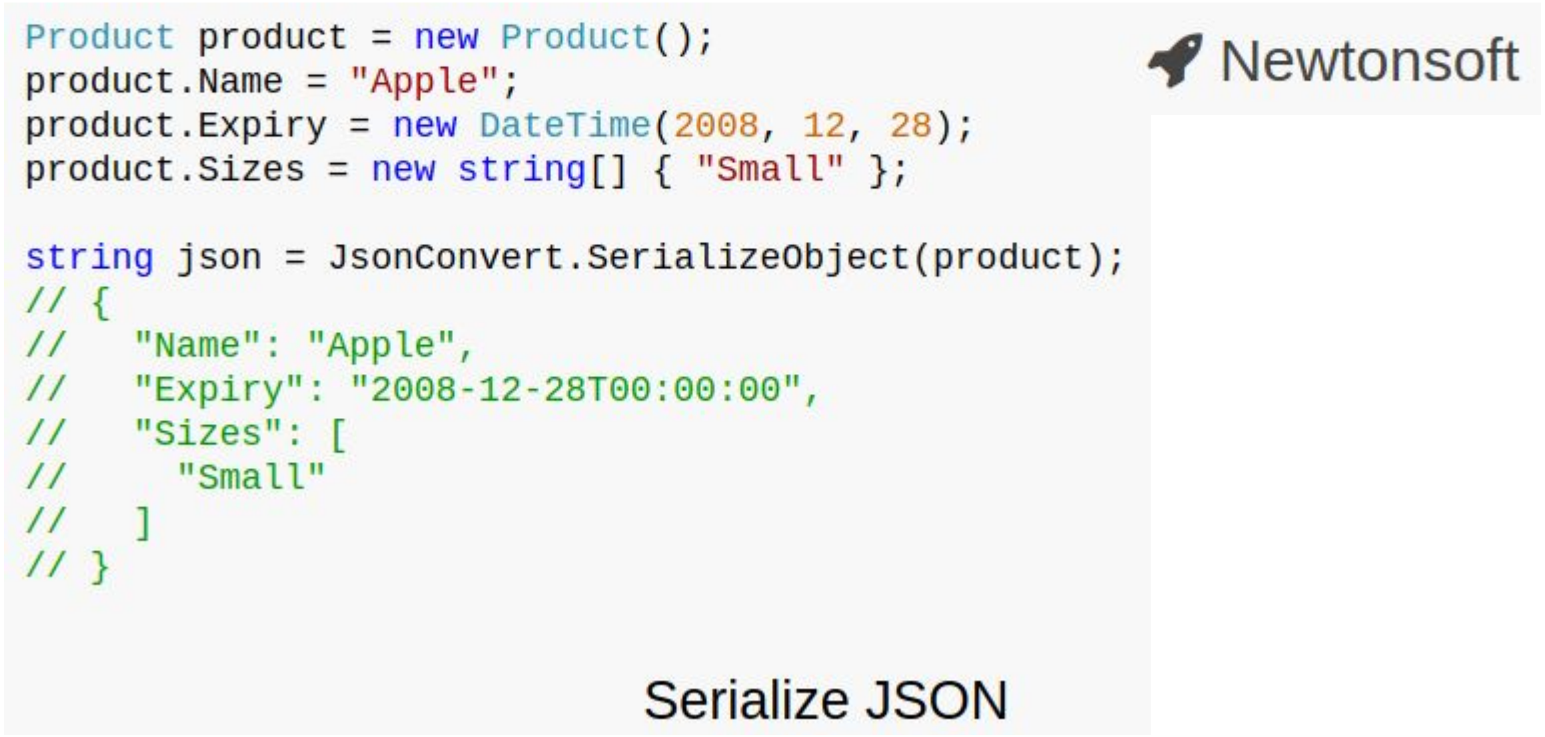
JSON advantage:

- JSON can be parsed more easily (~faster)
- JSON shorter (~smaller/less storage required)
- parson from JSON we immediately have a ready-to-use JS object (at frontend it is important)
- array can be defined
- the default of the web based communication

JSON disadvantage:

- unable to create cyclic / recursive datastructure

Serialization / Deserialization

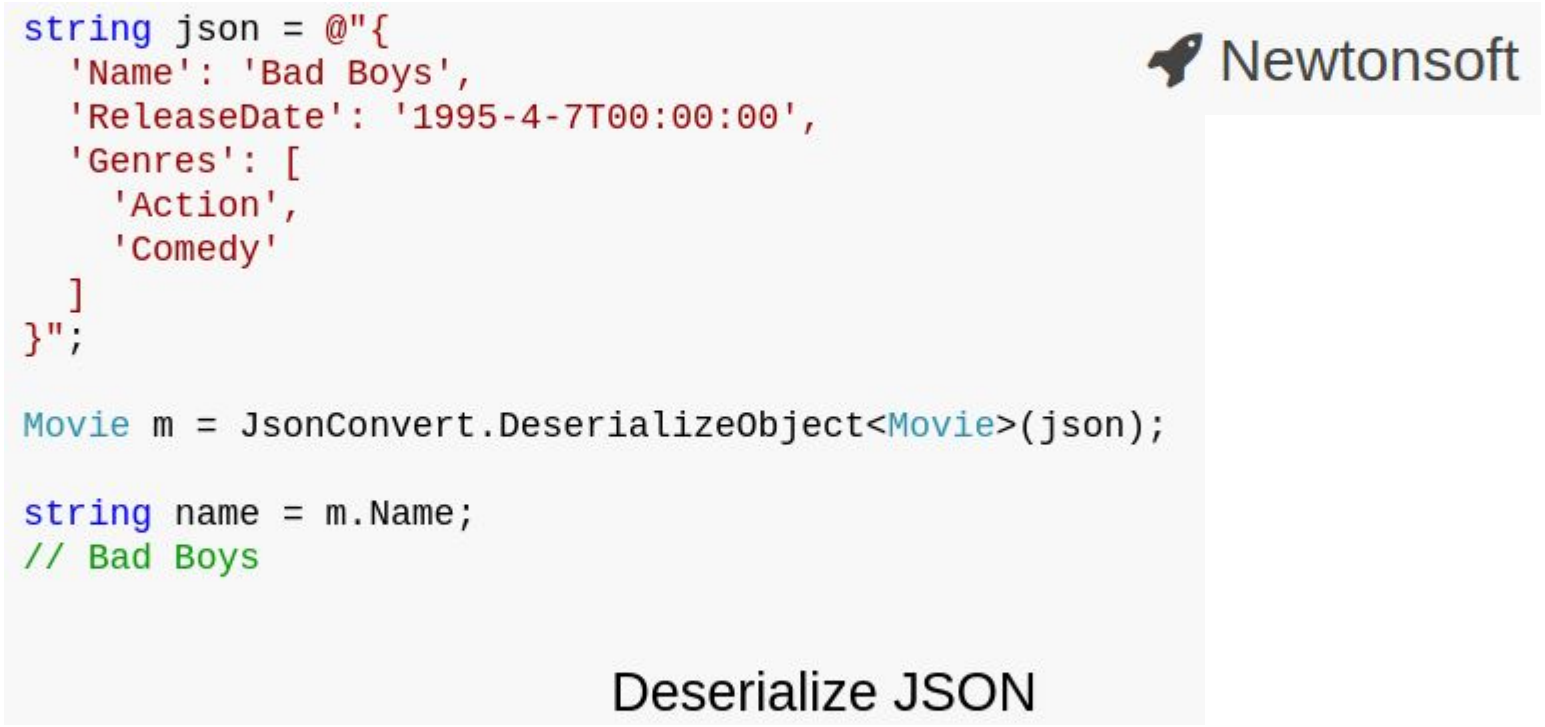


Newtonsoft

```
Product product = new Product();
product.Name = "Apple";
product.Expiry = new DateTime(2008, 12, 28);
product.Sizes = new string[] { "Small" };

string json = JsonConvert.SerializeObject(product);
// {
//   "Name": "Apple",
//   "Expiry": "2008-12-28T00:00:00",
//   "Sizes": [
//     "Small"
//   ]
// }
```

Serialize JSON

Serialization / Deserialization



```
string json = @"{
  'Name': 'Bad Boys',
  'ReleaseDate': '1995-4-7T00:00:00',
  'Genres': [
    'Action',
    'Comedy'
  ]
}";

Movie m = JsonConvert.DeserializeObject<Movie>(json);

string name = m.Name;
// Bad Boys
```

Serialization / Deserialization

Newtonsoft's JSON.Net

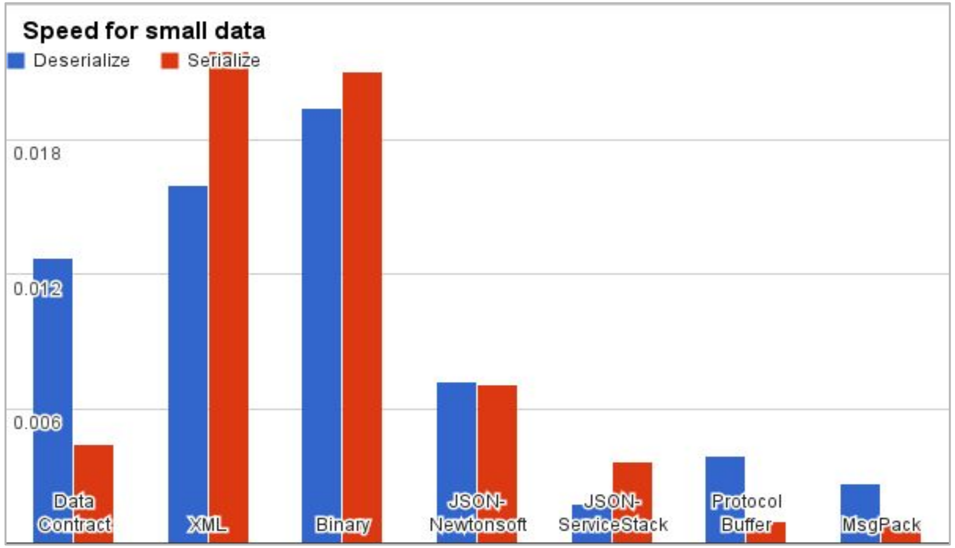
https://www.newtonsoft.com/json

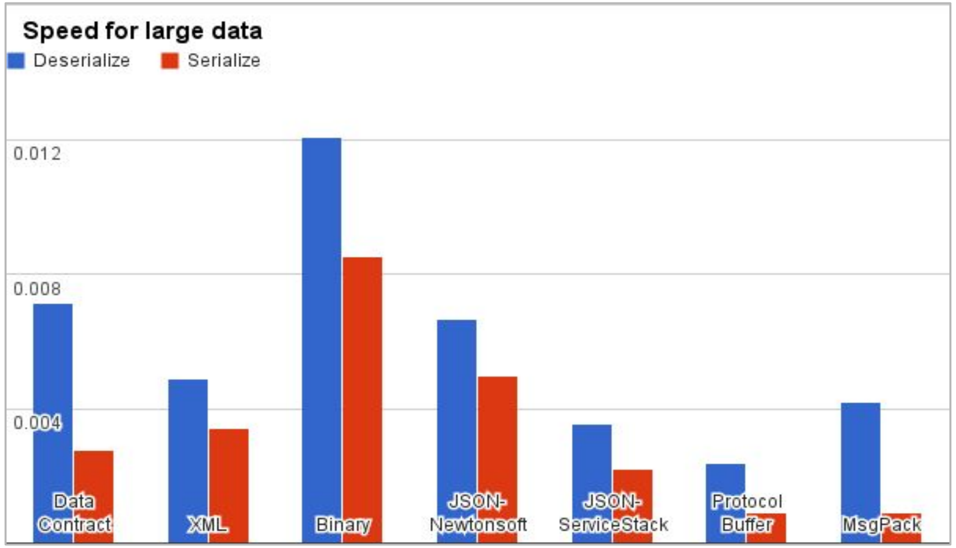
- well optimized
- fast*
- LINQ compatible

But the official JSON can be used as well:

- since .Net Core 3 official native support
- System.Text.Json NuGet package
- https://devblogs.microsoft.com/dotnet/try-the-new-system-text-json-apis/

Speed comparison





In many cases we only send smaller packages / information, so having a fast serialize/deserialize option is great!

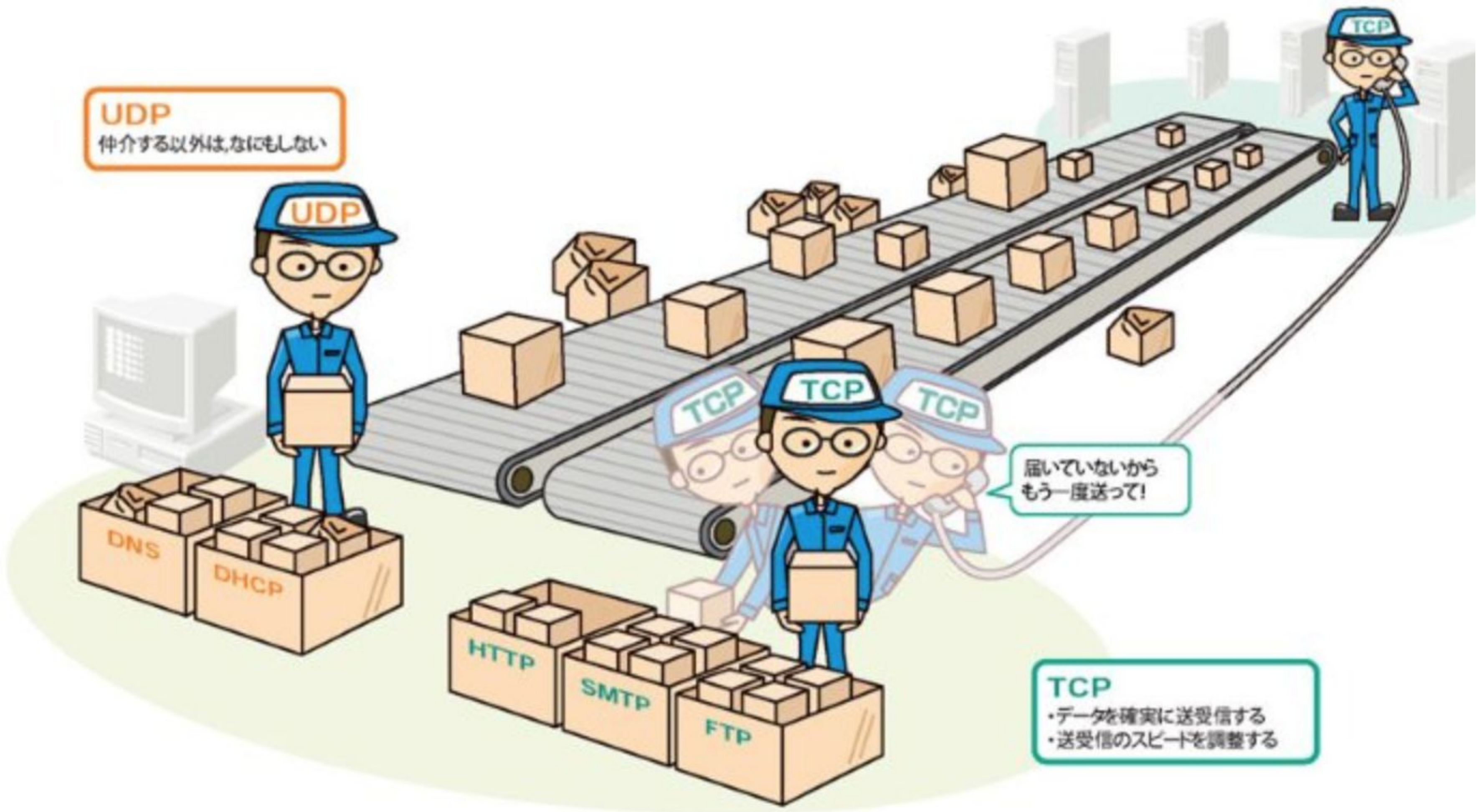
Low level data transmission

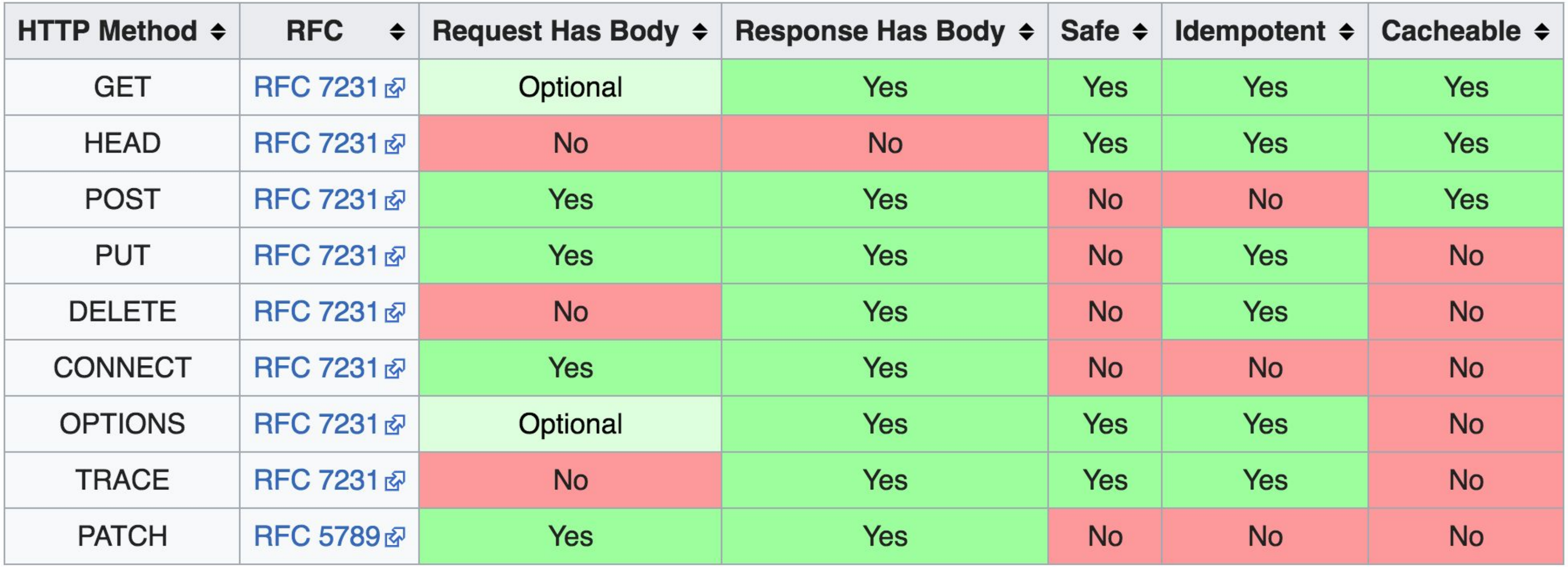
TCP / UDP protocols

- TCP: **there is no** data loss and/or change in order
- UDP: data loss and/or change in order **may be possible**

Every case, encoding, format must be handled by us ➔ do not reinvent the wheel ➔ there are best practices ➔ pre-made methods (SOAP, REST protocols)

HTTP build on top of TCP, as a higher level.


UDP
仲介する以外は,なにもしない
UDP
DNS
DHCP
TCP
TCP
TCP
TCP
届いていないから
もう一度送って!
HTTP
SMTP
FTP
TCP
・データを確実に送受信する
・送受信のスピードを調整する



HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

SOAP, REST

Do not reinvent the wheel

SOAP

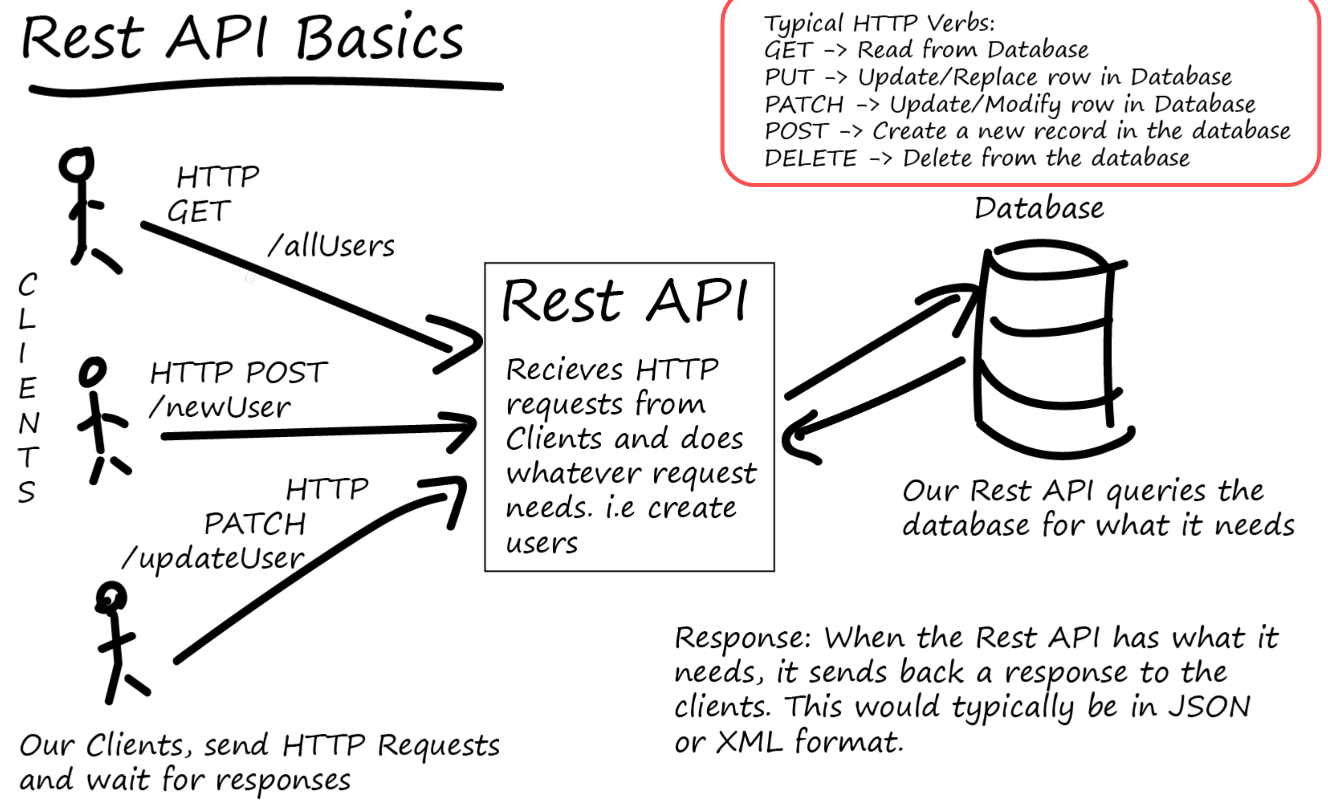
- Simple Object Access Protocol
- old protocol
- pre-defined SOAP XML format to call methods and pass any kind of parameter/result (array, list, object)
- the XML messages can be forwarded via whatever protocol we need, but mostly HTTP is/was used
- easy to implement (language + IDE support)
- slow and has a big overhead

Do not reinvent the wheel

REST

- Representative State Transfer
- in 95% of the cases after the called process the string/int parameters are listed
    - HTTP GET URL is only needed, nothing else
- complex data can be sent
    - using JSON (rarely XML) inside HTTP POST
- the answer usually JSON (rarely XML)
- easy to implement but more work than SOAP
- medium speed, medium overhead
- uses HTTP, so not raw TCP
- the REST is an **architectural approach** (not exactly A protocol)

Rest API



Recieves HTTP requests from Clients and does whatever request needs. i.e create users

Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database

Our Rest API queries the database for what it needs

Our Clients, send HTTP Requests and wait for responses

Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.

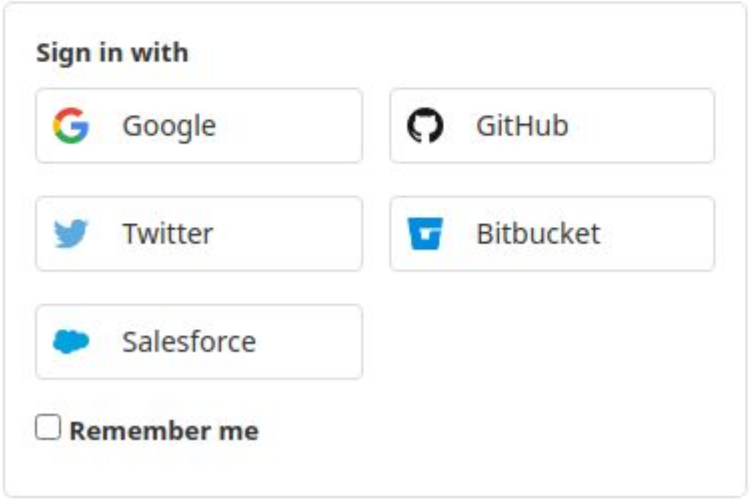
Rest API

Google, Facebook, Twitter etc. ➜ tons of public API endpoints

➜ features of big companies' can be used as a developer

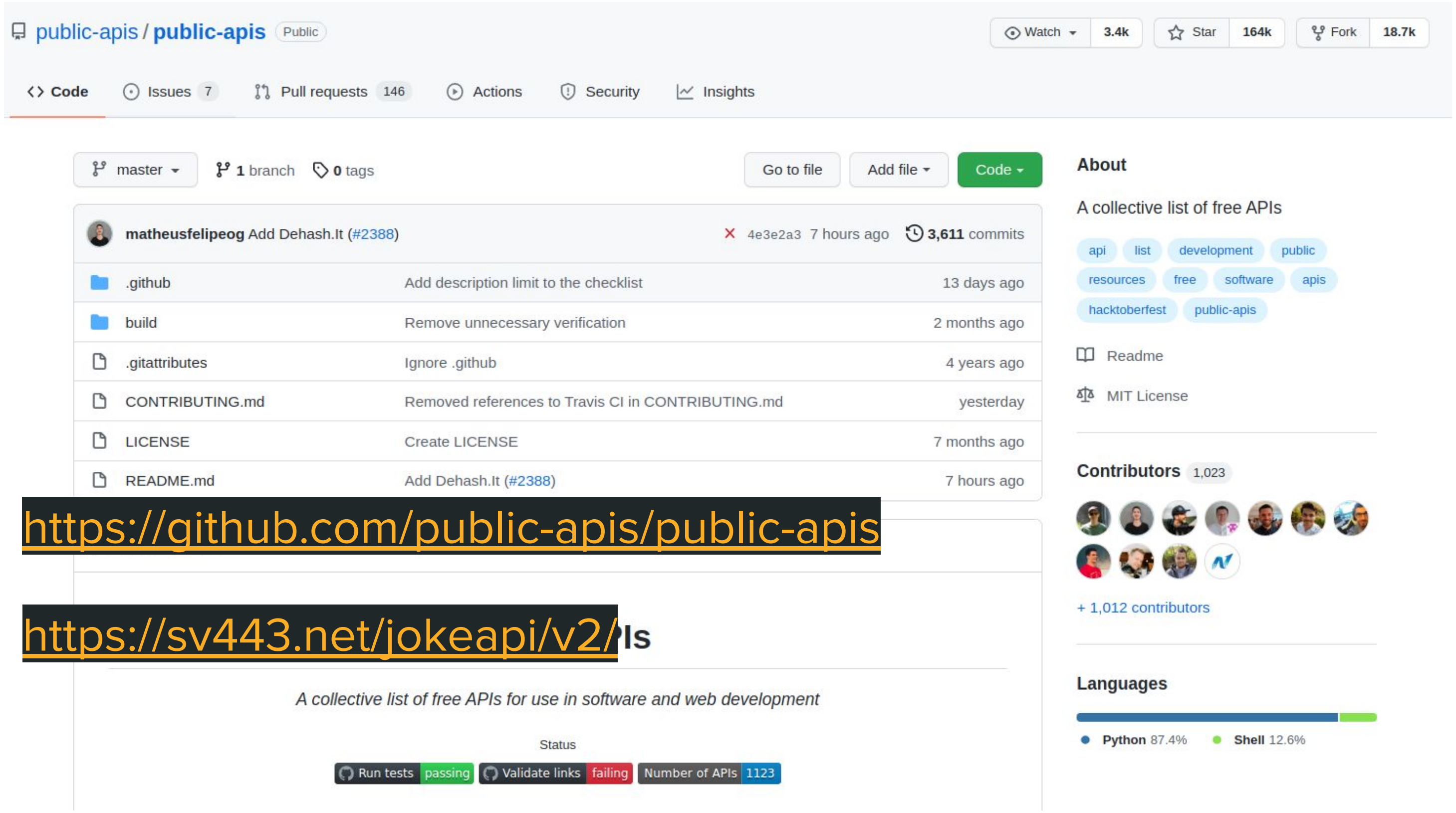
frequent example: "Login with ..."

- API endpoint is called
- request: "is the user valid?"
- reply: "yes or no"
- OAuth


Sign in with

☐ Remember me

Public APIs



public-apis / public-apis Public

Watch 3.4k Star 164k Fork 18.7k

Code | Issues 7 | Pull requests 146 | Actions | Security | Insights

master · 1 branch · 0 tags · Go to file · Add file · Code

matheusfelipeog Add Dehash.lt (#2388) ✕ 4e3e2a3 7 hours ago 3,611 commits

.github	Add description limit to the checklist	13 days ago
build	Remove unnecessary verification	2 months ago
.gitattributes	Ignore .github	4 years ago
CONTRIBUTING.md	Removed references to Travis CI in CONTRIBUTING.md	yesterday
LICENSE	Create LICENSE	7 months ago
README.md	Add Dehash.lt (#2388)	7 hours ago

About

A collective list of free APIs

api list development public resources free software apis hacktoberfest public-apis

Readme

MIT License

Contributors 1,023

+ 1,012 contributors

Languages

● Python 87.4% ● Shell 12.6%

Is

A collective list of free APIs for use in software and web development

Status

Run tests passing | Validate links failing | Number of APIs 1123

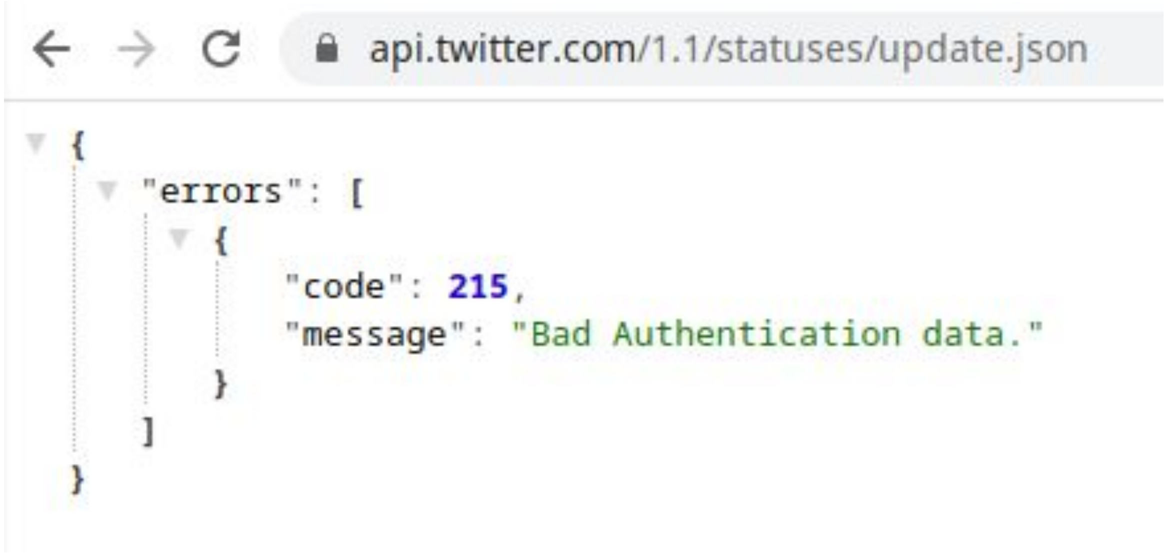
https://github.com/public-apis/public-apis

https://sv443.net/jokeapi/v2/

Twitter API example

Twitter API

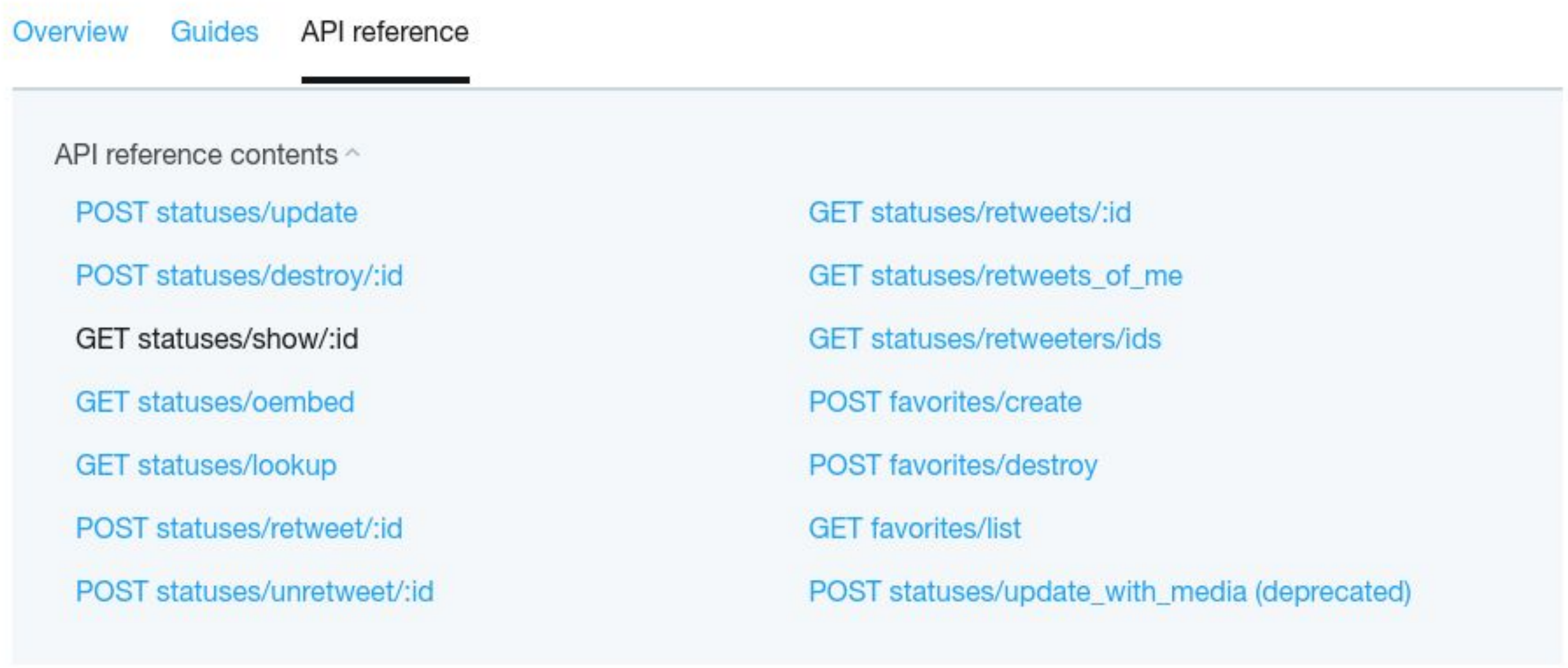
1. step: OAuth identification
2. step: GET / POST request (eg. get all the tweets)



api.twitter.com/1.1/statuses/update.json

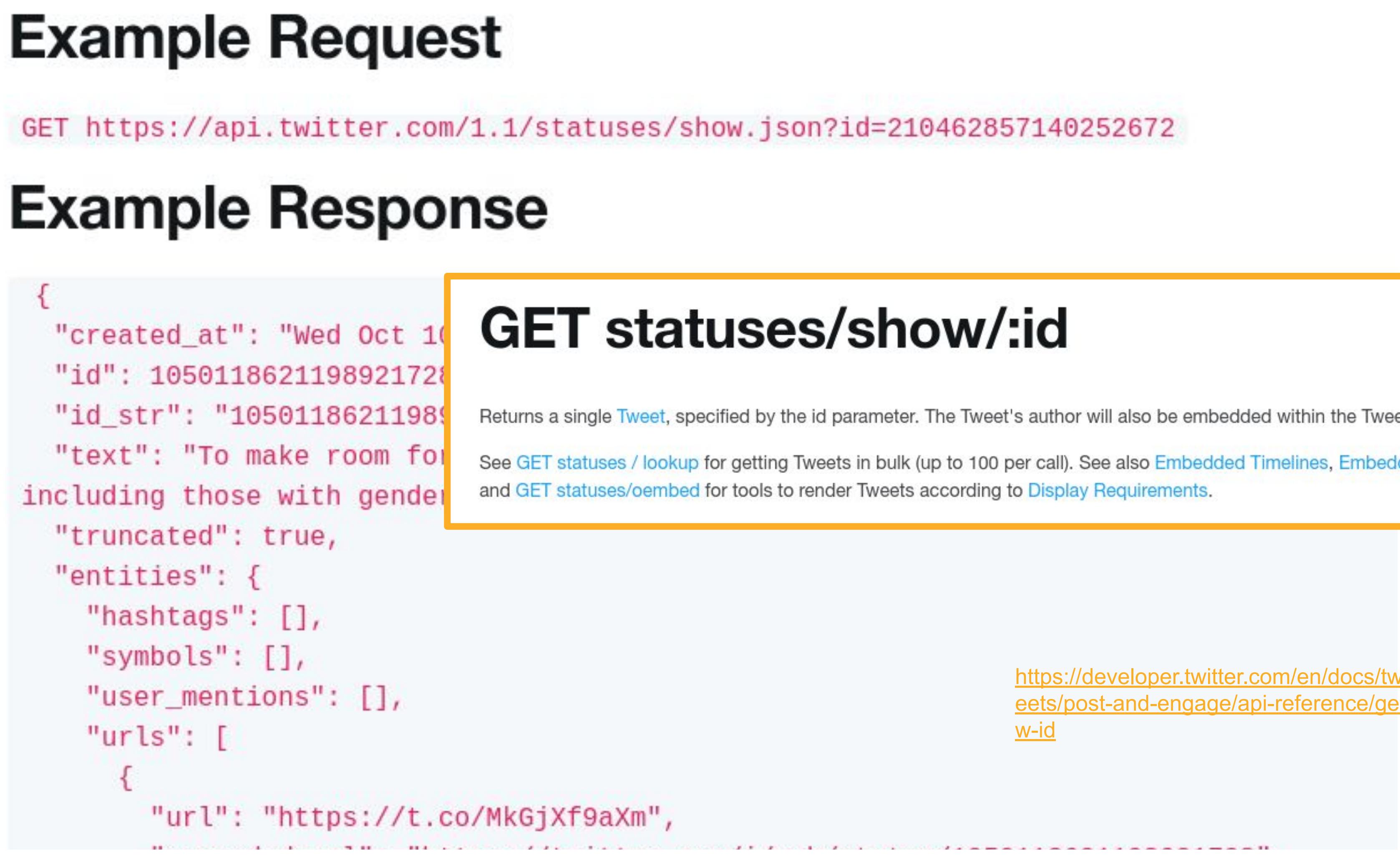
```
{
    "errors": [
        {
            "code": 215,
            "message": "Bad Authentication data."
        }
    ]
}
```

Twitter API


Overview Guides API reference

API reference contents

POST statuses/update	GET statuses/retweets/:id
POST statuses/destroy/:id	GET statuses/retweets_of_me
GET statuses/show/:id	GET statuses/retweeters/ids
GET statuses/oembed	POST favorites/create
GET statuses/lookup	POST favorites/destroy
POST statuses/retweet/:id	GET favorites/list
POST statuses/unretweet/:id	POST statuses/update_with_media (deprecated)

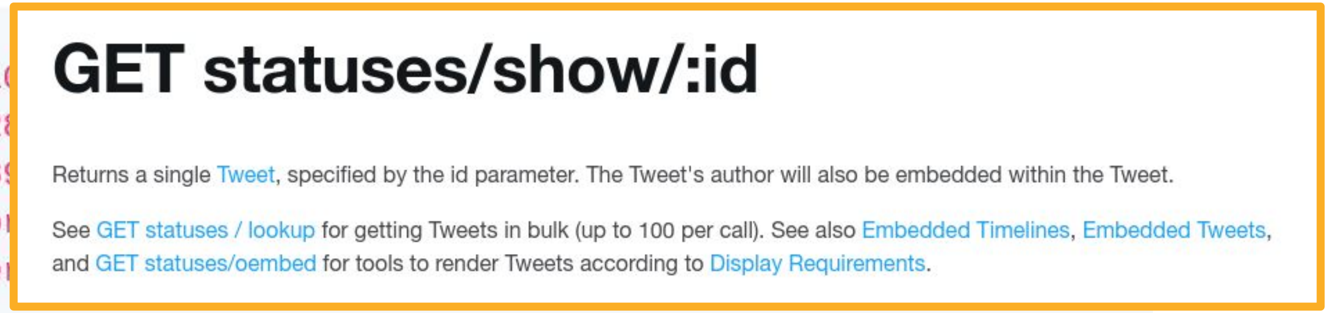


Example Request

```
GET https://api.twitter.com/1.1/statuses/show.json?id=210462857140252672
```

Example Response

```
{
  "created_at": "Wed Oct 1
  "id": 105011862119892172
  "id_str": "10501186211989
  "text": "To make room fo
including those with gende
  "truncated": true,
  "entities": {
    "hashtags": [],
    "symbols": [],
    "user_mentions": [],
    "urls": [
      {
        "url": "https://t.co/MkGjXf9aXm",
```

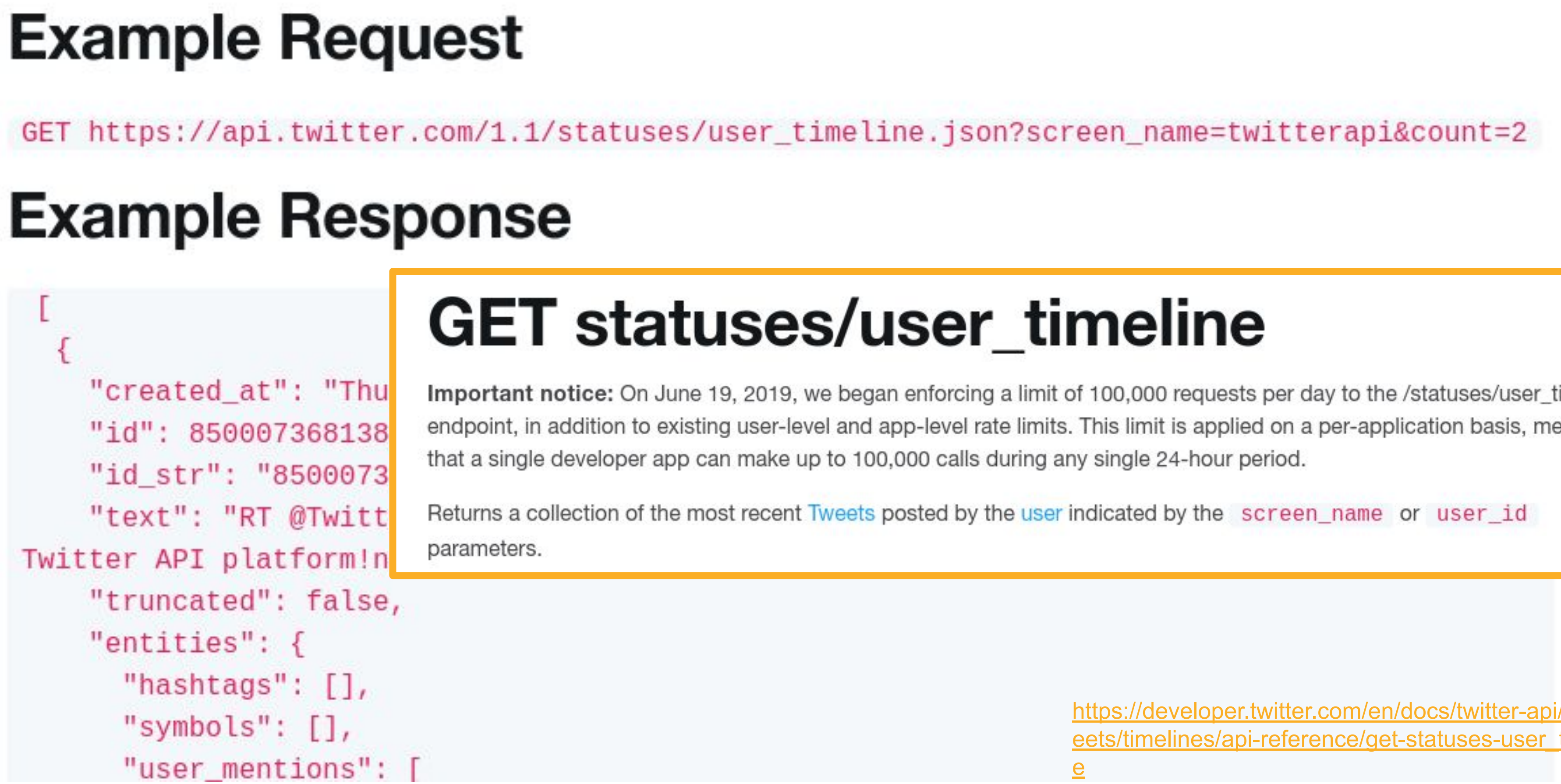


GET statuses/show/:id

Returns a single Tweet, specified by the id parameter. The Tweet's author will also be embedded within the Tweet.

See GET statuses / lookup for getting Tweets in bulk (up to 100 per call). See also Embedded Timelines, Embedded Tweets, and GET statuses/oembed for tools to render Tweets according to Display Requirements.

https://developer.twitter.com/en/docs/twitter-api/v1/tweets/post-and-engage/api-reference/get-statuses-show-id

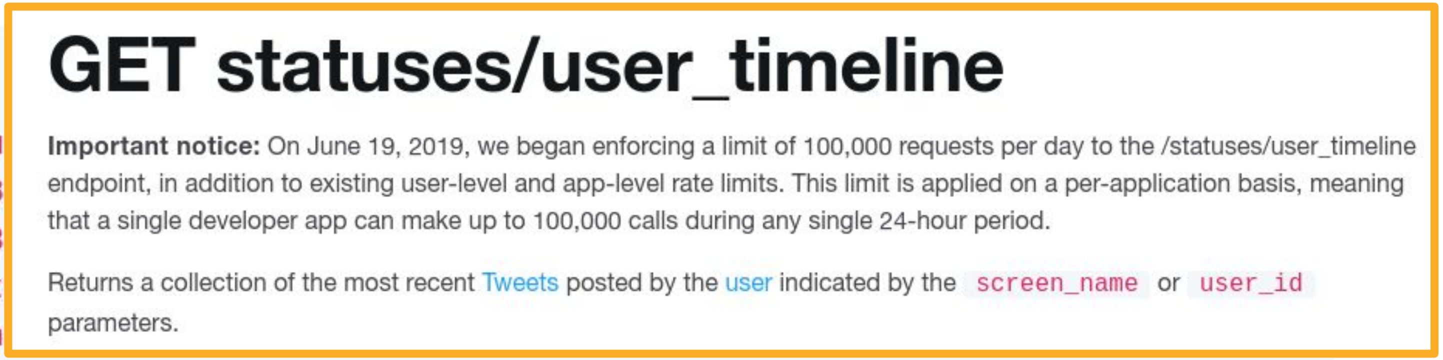


Example Request

```
GET https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2
```

Example Response

```
 [
  {
    "created_at": "Thu
    "id": 850007368138
    "id_str": "8500073
    "text": "RT @Twitt
Twitter API platform!n
    "truncated": false,
    "entities": {
      "hashtags": [],
      "symbols": [],
      "user_mentions": [
```



GET statuses/user_timeline

Important notice: On June 19, 2019, we began enforcing a limit of 100,000 requests per day to the /statuses/user_timeline endpoint, in addition to existing user-level and app-level rate limits. This limit is applied on a per-application basis, meaning that a single developer app can make up to 100,000 calls during any single 24-hour period.

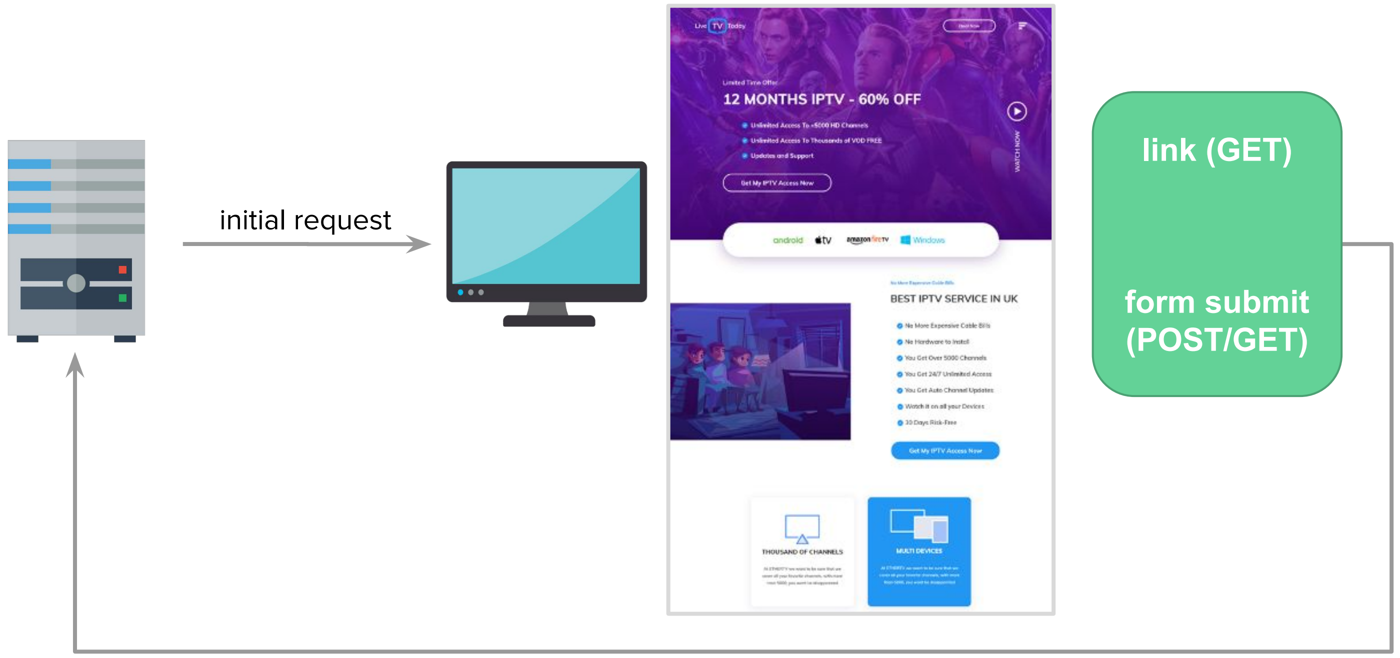
Returns a collection of the most recent Tweets posted by the user indicated by the `screen_name` or `user_id` parameters.

https://developer.twitter.com/en/docs/twitter-api/v1/tweets/timelines/api-reference/get-statuses-user_timeline

Websites’ working principle

REST API (before)

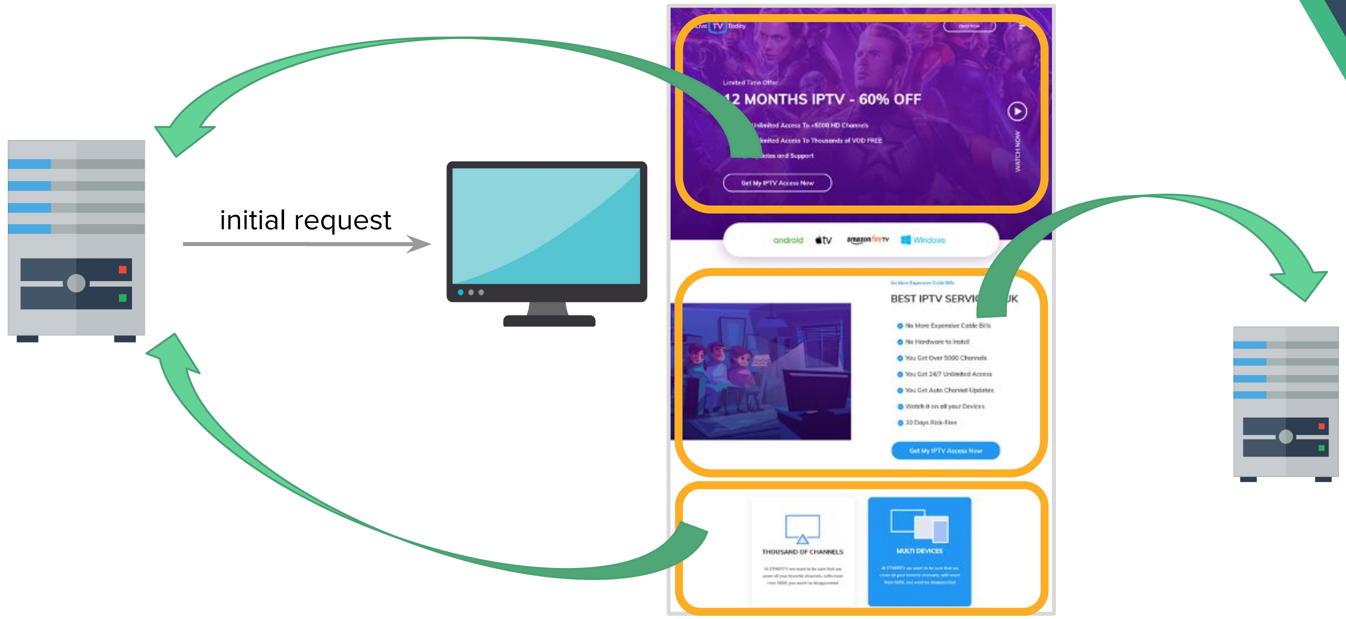
1. server creates the HTML and sends it
2. “I do some action on the website”
3. we goes back to the server which creates the HTML again, based on my action, and sends it again
4. repeat



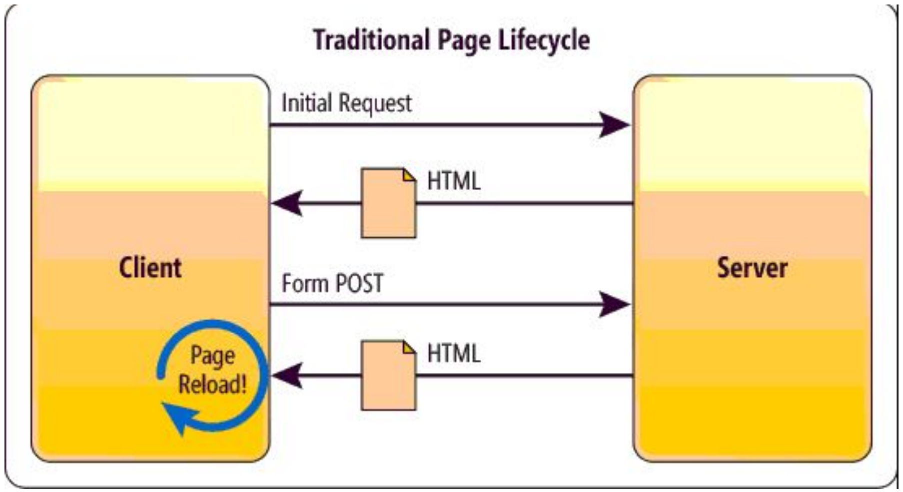
REST API (after)

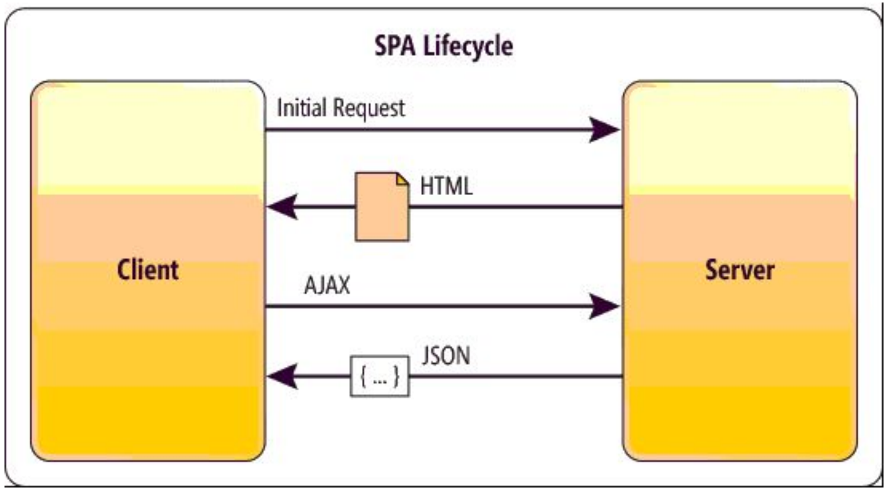
1. the client receives x amount of code (not the full) generated on the server
2. the client creates the content by calling different API endpoints and thus and builds up the site

advantage: after some action on the site the full content shouldn’t need to change, only some parts of the site ➜ early implementation of this was AJAX



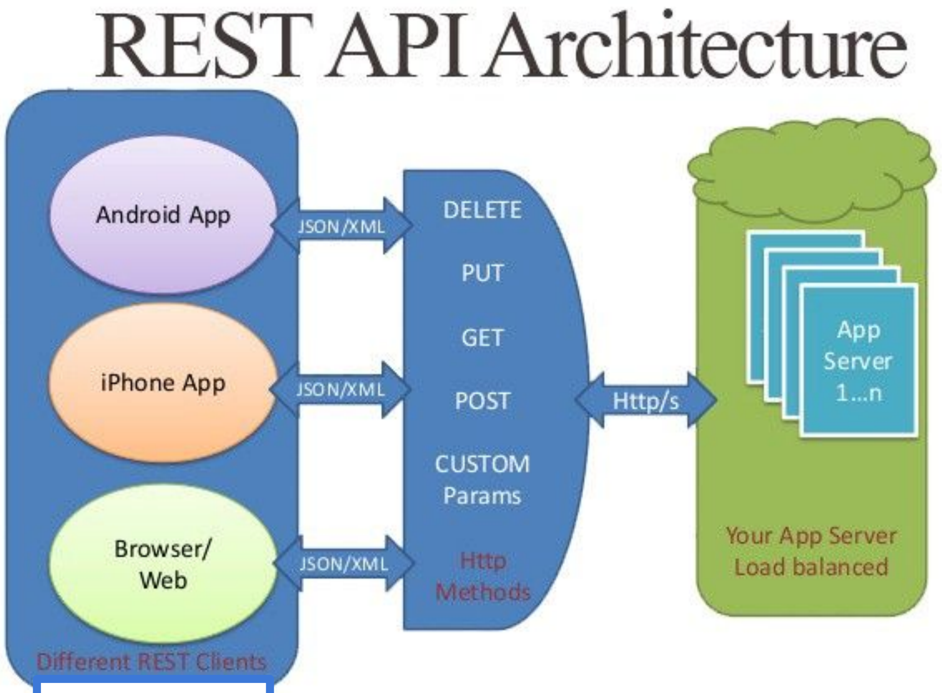
REST API


Traditional Page Lifecycle
Client
Initial Request
HTML
Form POST
HTML
Page Reload!
Server

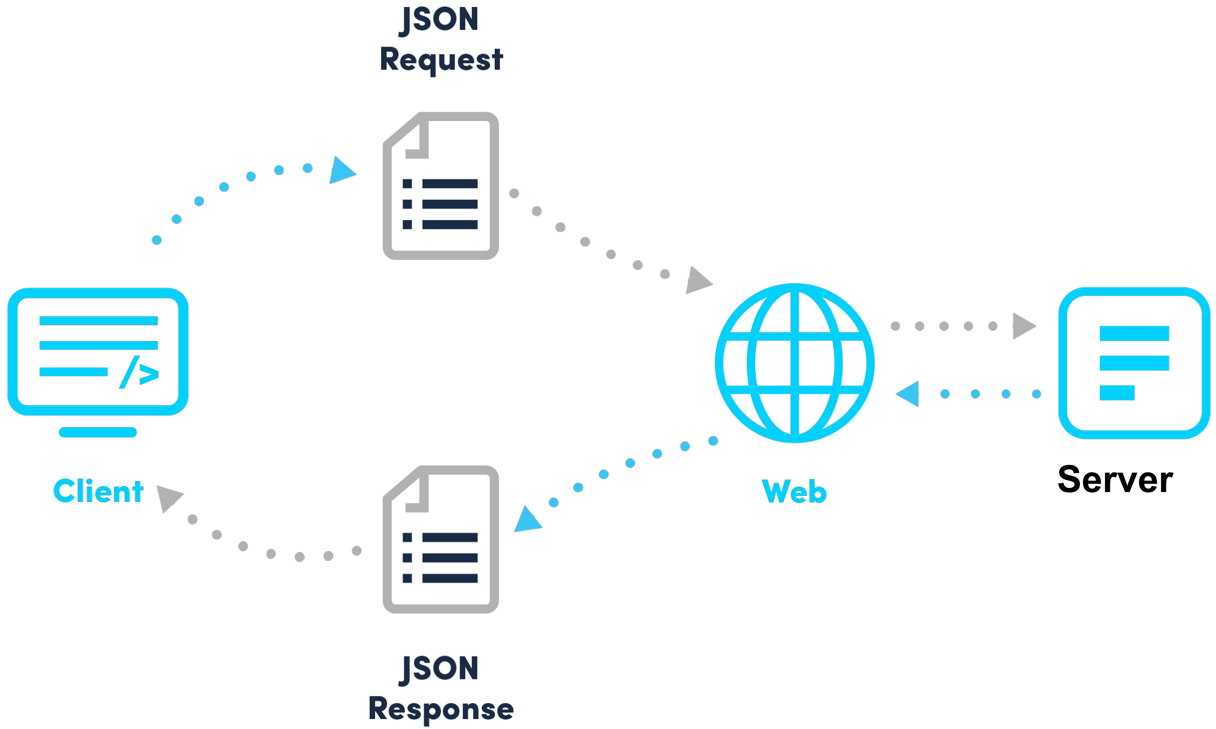

SPA Lifecycle
Client
Initial Request
HTML
AJAX
JSON
{ ... }
Server

REST API

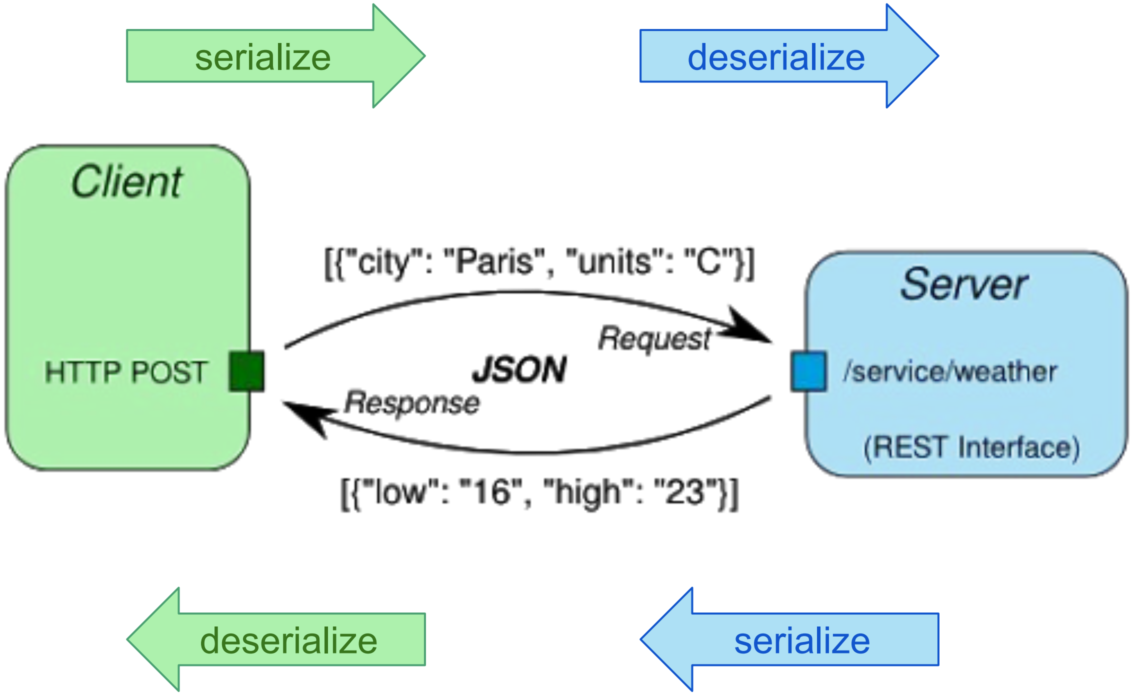
- not important who is the client
  - for a long time only web browsers can interact, so browsers were clients
  - but since not only HTML can be sent (but JSON) it's a whole new game
- because whatever is the client what's matter is what JSON package is sent



JSON communication



JSON communication



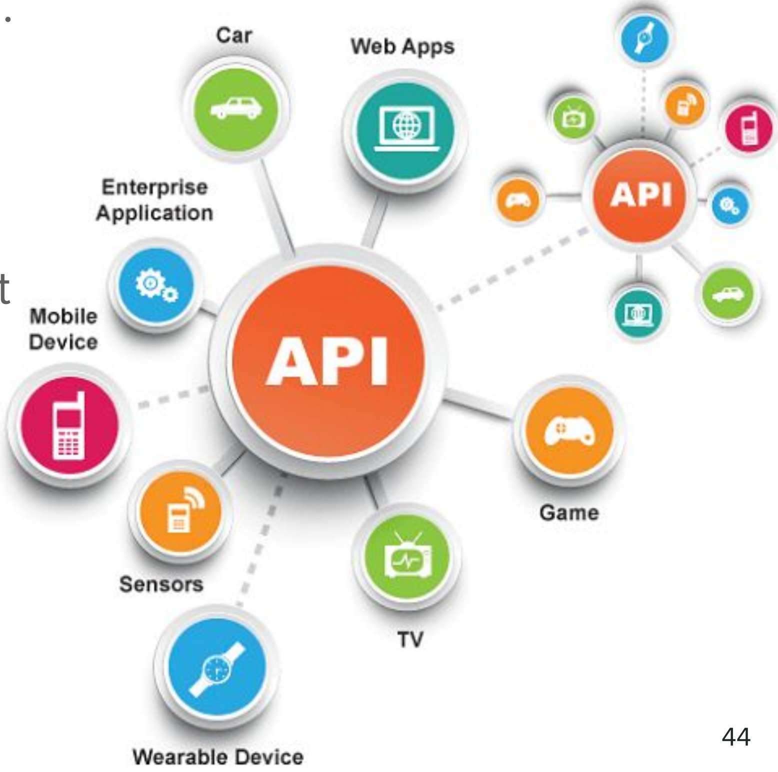
API-first development

API-first approach

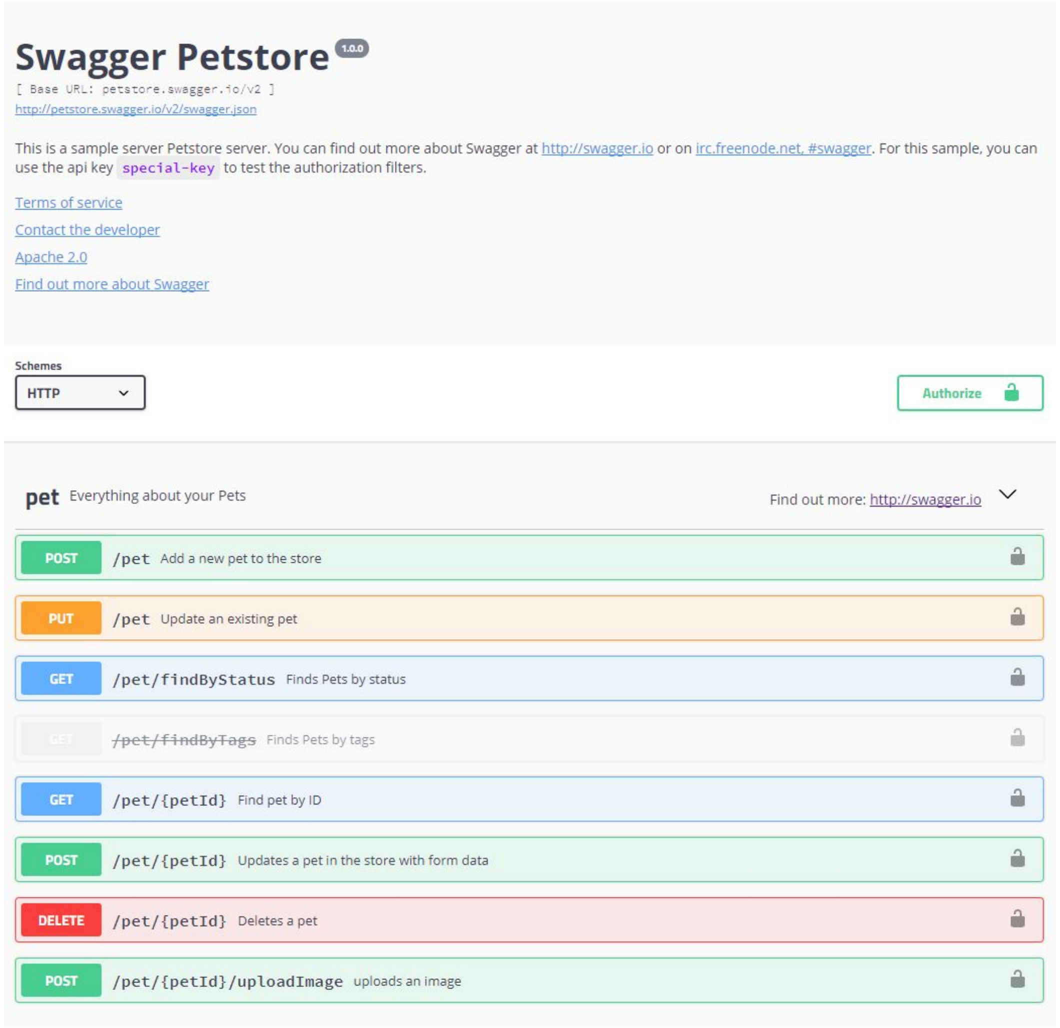
In the web development world it’s a new standard.

Not a design pattern!

“Web APIs have been around for nearly 20 years, but it is only in the past few years that the concept of “API first” has gained traction with software teams.”
https://swagger.io/resources/articles/adopting-an-api-first-approach/



Swagger


Swagger Petstore 1.0.0

[Base URL: petstore.swagger.io/v2]

http://petstore.swagger.io/v2/swagger.json

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Terms of service

Contact the developer

Apache 2.0

Find out more about Swagger

Schemes: HTTP

Authorize

pet Everything about your Pets — Find out more: http://swagger.io

POST `/pet` Add a new pet to the store

PUT `/pet` Update an existing pet

GET `/pet/findByStatus` Finds Pets by status

GET ~~`/pet/findByTags`~~ Finds Pets by tags

GET `/pet/{petId}` Find pet by ID

POST `/pet/{petId}` Updates a pet in the store with form data

DELETE `/pet/{petId}` Deletes a pet

POST `/pet/{petId}/uploadImage` uploads an image

API-first approach

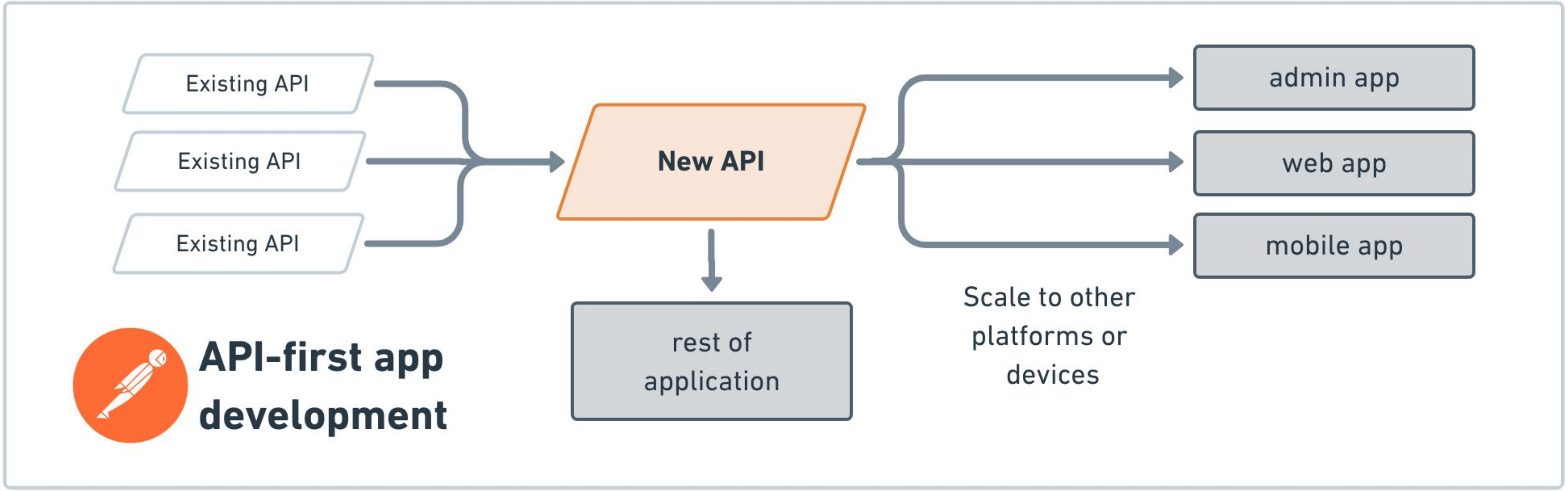
Advantages

- Development teams can work in parallel
- Reduces the cost of developing apps
- Increases the speed to market
- Ensures good developer experiences
- Reduces the risk of failure

https://swagger.io/resources/articles/adopting-an-api-first-approach/

API-first approach

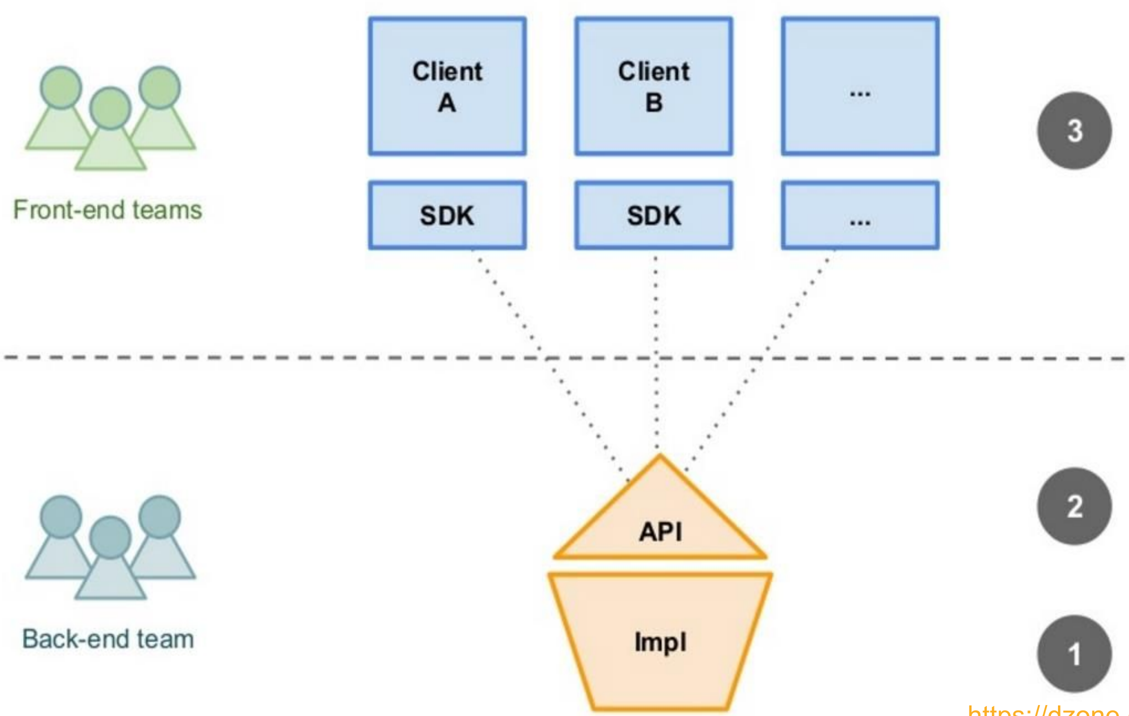
APIs can be re-used to create new, more complex APIs.



code first approach

API-first approach

Backend and frontend can be more separated and parallelize.

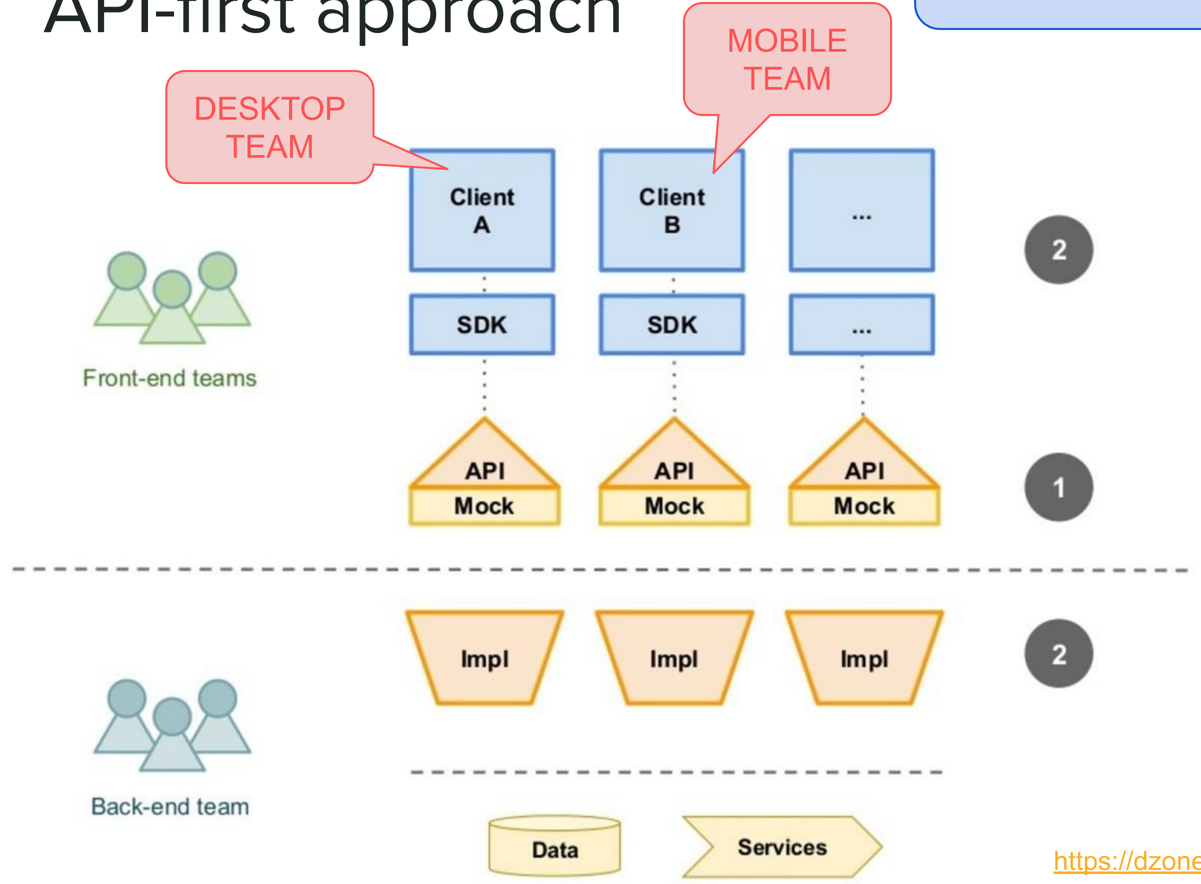


- As you can see in the picture, first, the back-end team is starting to develop and implement a new api.
- Second, the api is being given to the front-end teams and testers for using and testing it.
- Third, the front-end teams and testers are building sdks, tests, and more to interact with the api.
- This is synchronous development.

https://dzone.com/articles/an-api-first-development-approach-1

API-first approach

Backend and frontend can be more separated and parallelize.

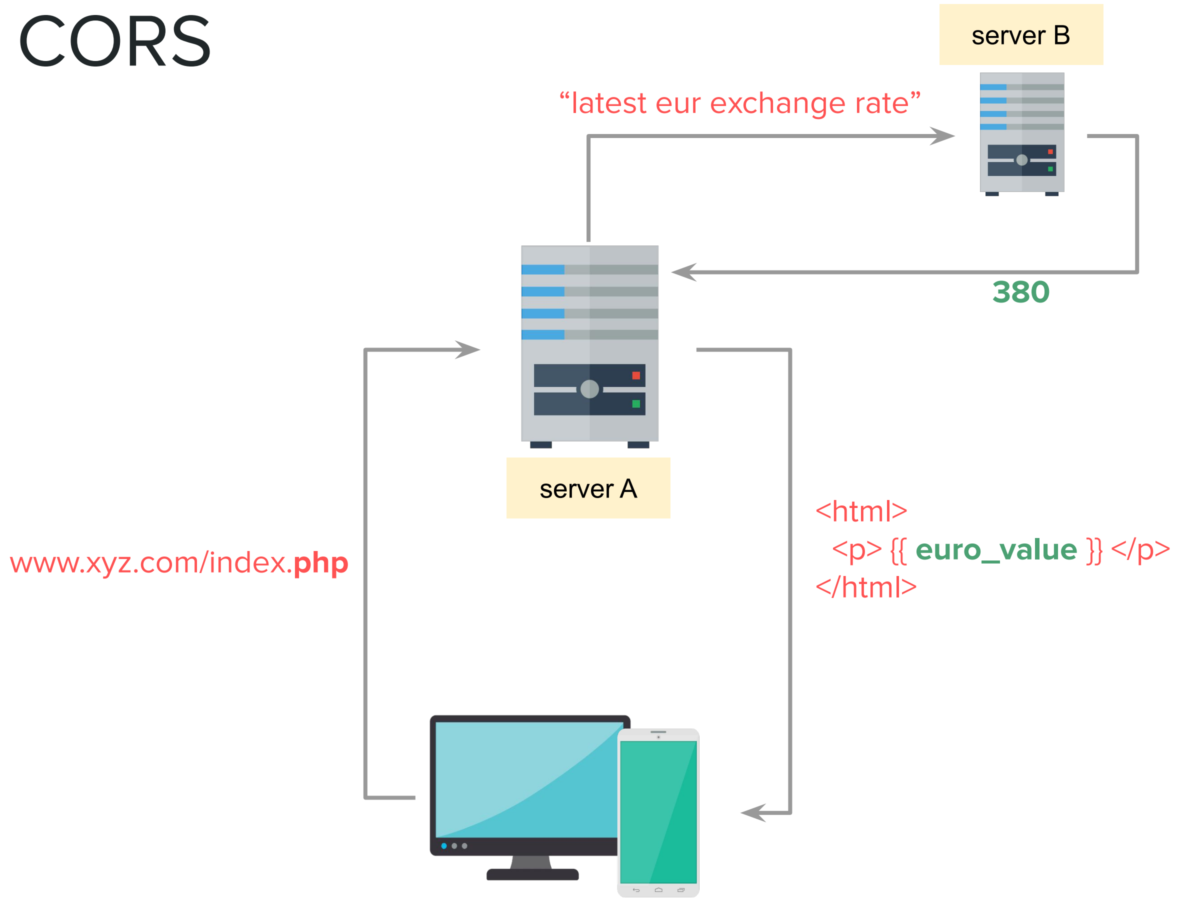


- Here we can see the first the apis that are created are mocks.
- Second, both back-end, front-end, and test teams are starting to work with the mocked apis.
- Once the api is ready, all teams can switch to the production or staging api.
- This saves a lot of development time.

https://dzone.com/articles/an-api-first-development-approach-1

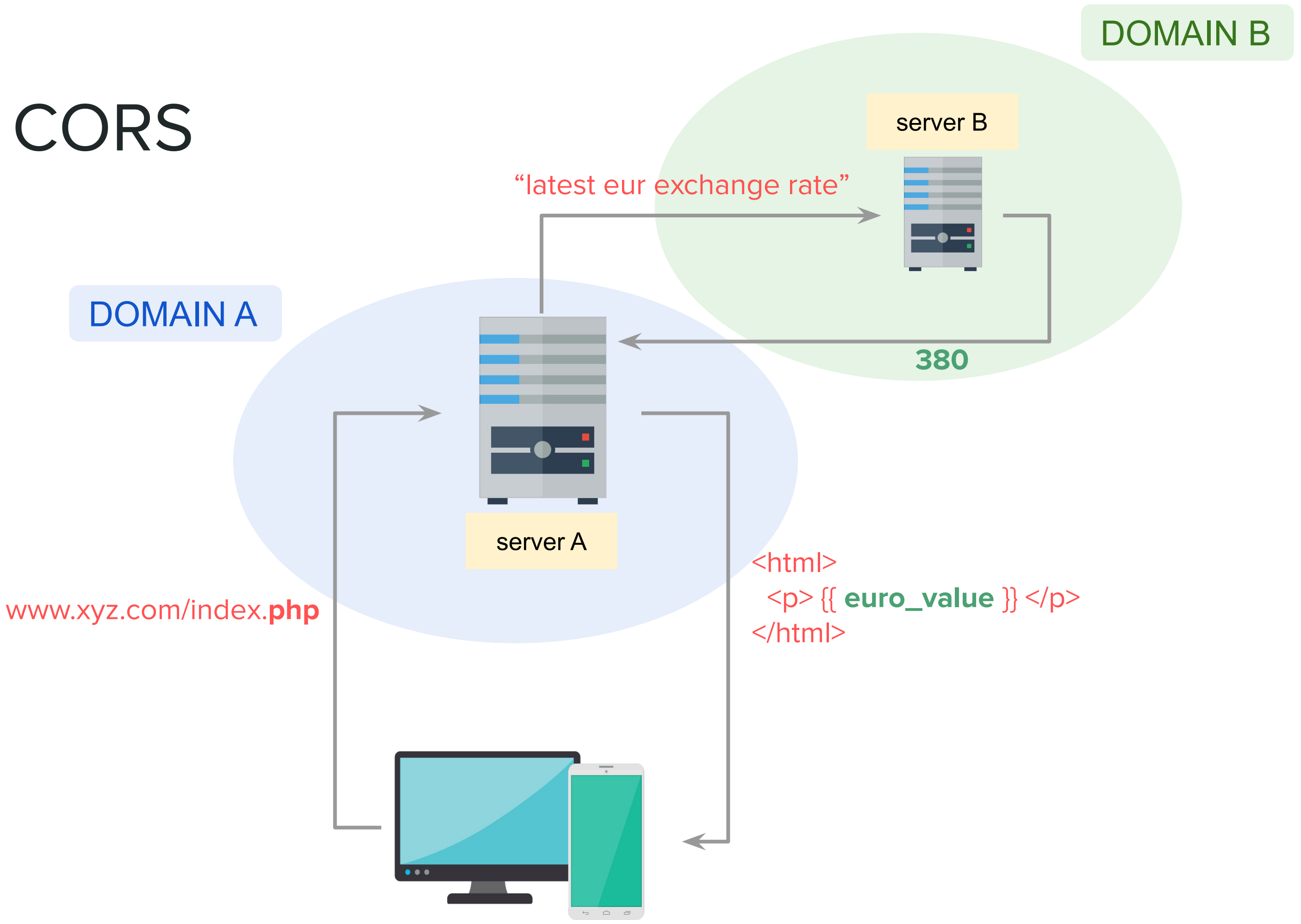
CORS

CORS

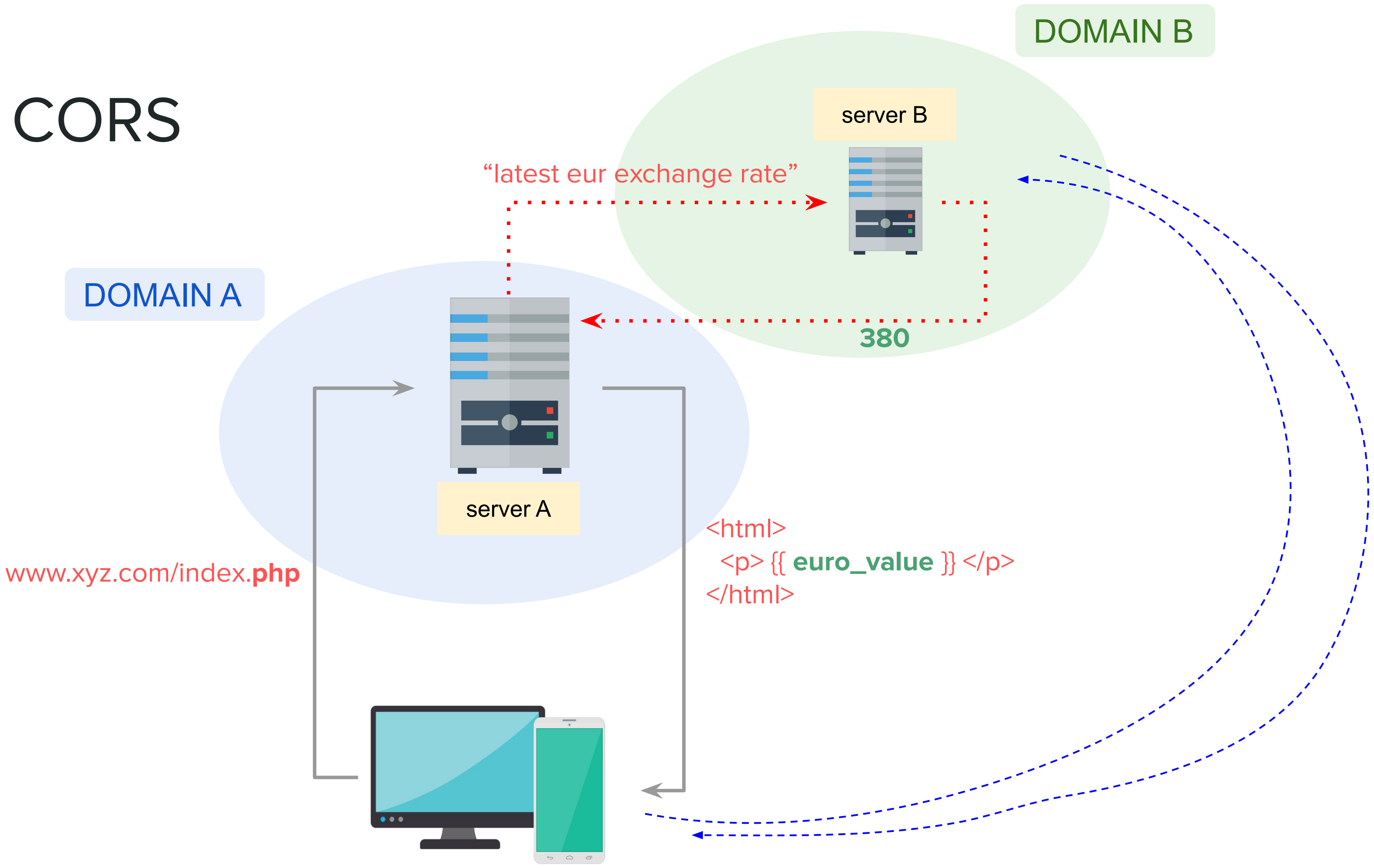


```
<html>
  <p> {{ euro_value }} </p>
</html>
```

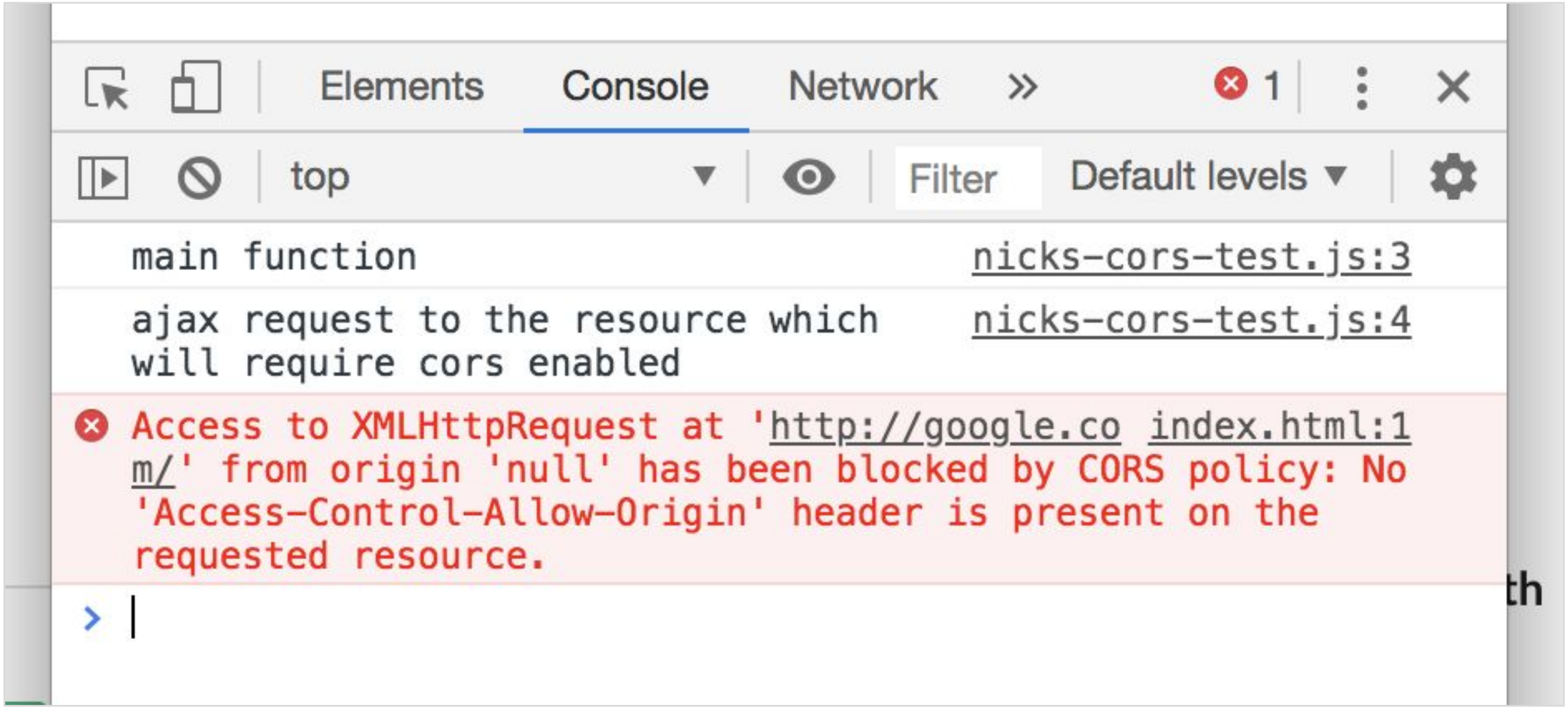
CORS


DOMAIN B
server B
"latest eur exchange rate"
380
DOMAIN A
server A
www.xyz.com/index.php
<html>
<p> {{ euro_value }} </p>
</html>

CORS


DOMAIN B
server B
"latest eur exchange rate"
DOMAIN A
380
server A
www.xyz.com/index.php
<html>
<p> {{ euro_value }} </p>
</html>

CORS in console



```
main function                                      nicks-cors-test.js:3
ajax request to the resource which                 nicks-cors-test.js:4
will require cors enabled
Access to XMLHttpRequest at 'http://google.co  index.html:1
m/' from origin 'null' has been blocked by CORS policy: No
'Access-Control-Allow-Origin' header is present on the
requested resource.
>
```

CORS (Cross-Origin Resource Sharing)

“Cross-Origin Resource Sharing (CORS) is an **HTTP-header based mechanism that allows a server to indicate any other origins** (domain, scheme, or port) than its own from which a browser should permit loading of resources. […]”

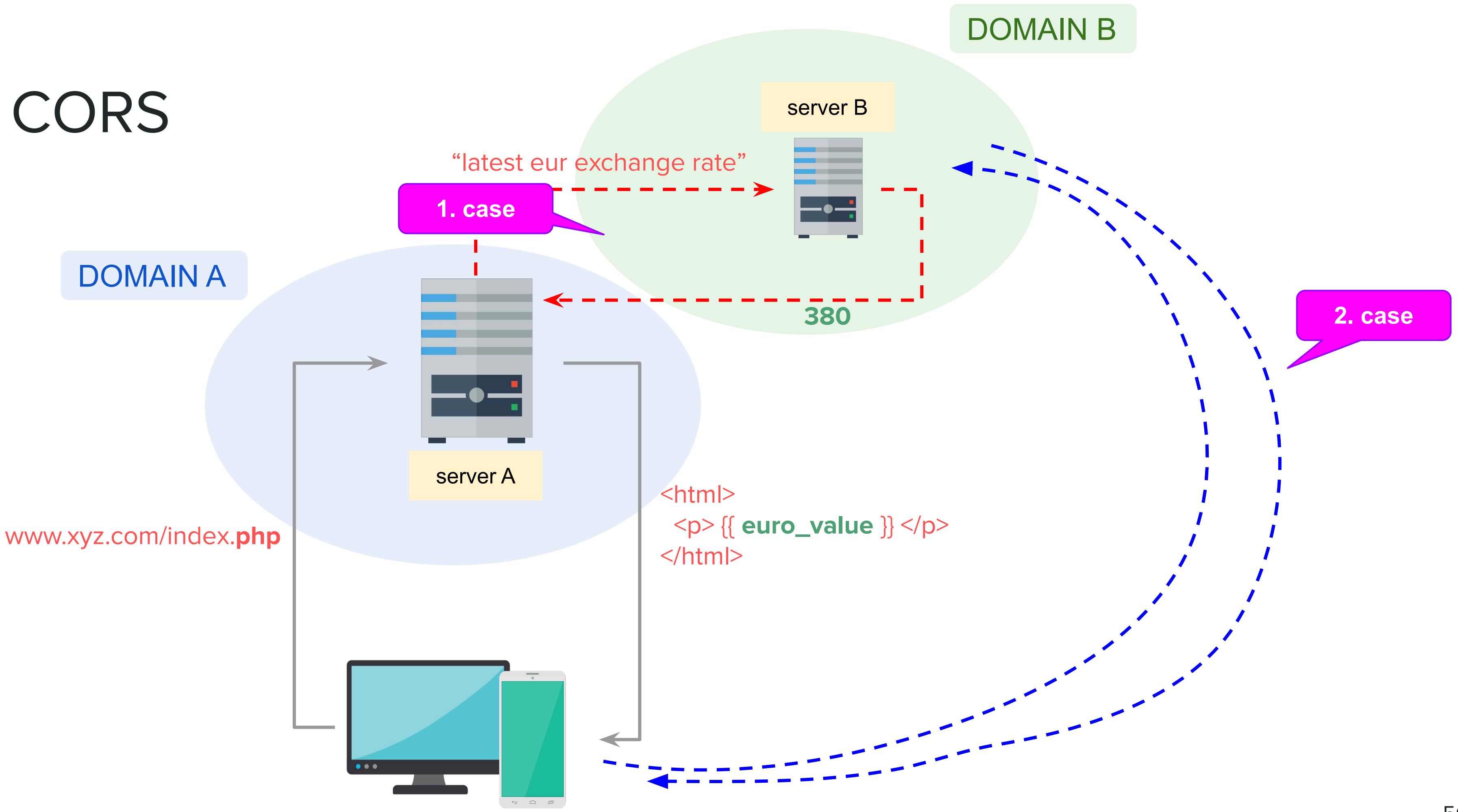
> *HTTP header alapú mechanizmus amely engedélyezi a szervernek egyéb források jelzését.*
> ***[1. case] server-A ↔ server-B***

“For **security reasons, browsers restrict cross-origin HTTP requests initiated from scripts**. For example, XMLHttpRequest and the Fetch API follow the same-origin policy. This means that a web application using **those APIs can only request resources from the same origin the application was loaded from** unless the response from other origins includes the right CORS headers.”

> *Biztonsági okokból a böngészők korlátozzák az ilyen cross-origin jellegű, script-ből kezdeményezett HTTP request-eket. Ez azt jelenti, hogy a webalkalmazás csak azokat az API-kat tudja használni (onnan kérhet adatot) ahonnan ő maga (az alkalmazás) betöltésre került. (app @ domain-a → API request @ domain-a)*
> ***[2. case] client ↔ server-B***

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

CORS

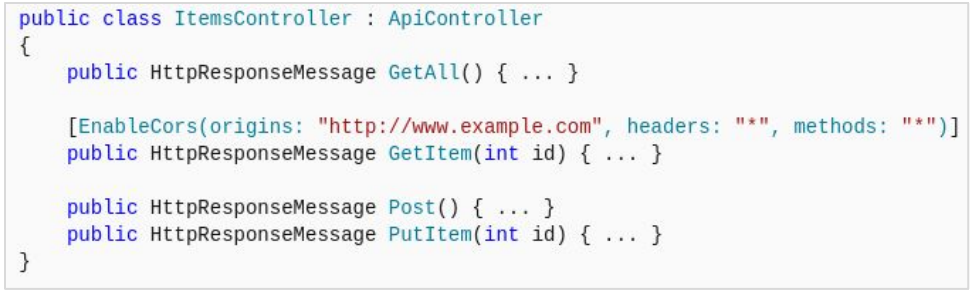

DOMAIN B
server B
"latest eur exchange rate"
1. case
DOMAIN A
380
server A
2. case
www.xyz.com/index.**php**
<html>
<p> {{ **euro_value** }} </p>
</html>

CORS (Cross-Origin Resource Sharing)

Security is great but we have to develop our application! :)

Enable CORS:

- on the server side (if we have access and right to do)
    - MS ➜ Enable cross-origin requests in ASP.NET Web API 2
    - ASP web API 2: You can enable CORS per *action*, per *controller*, or *globally* for all Web API controllers in your application.



```
public class ItemsController : ApiController
{
    public HttpResponseMessage GetAll() { ... }

    [EnableCors(origins: "http://www.example.com", headers: "*", methods: "*")]
    public HttpResponseMessage GetItem(int id) { ... }

    public HttpResponseMessage Post() { ... }
    public HttpResponseMessage PutItem(int id) { ... }
}
```

- by configuring a devserver (if there is no access to backend server)
    - using webpack and proxies for example (not a topic of this semester!)

Thanks for your attention!

Sipos Miklós

sipos.miklos@nik.uni-obuda.hu

https://users.nik.uni-obuda.hu/siposm/