



# AJAX

- + budowa url
- + JSON
- + Web serwer
- + endpointy i API



# AJAX - dziś i kiedyś

AJAX, akronim od Asynchronous JS and XML

Zbiór technik, które pozwalają stronom/aplikacjom komunikować się w sposób asynchroniczny z serwerem. AJAX pozwala na dynamiczne (bez odświeżania) aktualizowanie zawartości strony/aplikacji.

W starych (dobrych?) czasach korzystało się z metody `XMLHttpRequest` czy opartej na niej metodzie `ajax` w jQuery (`$.ajax()`). Obecnie najpopularniejszym, natywnym (dostępnym w obiekcie globalnym `window`) sposobem na asynchroniczne pobieranie danych jest skorzystanie z metody `fetch` (czy szerzej API Fetch).



# Asynchroniczność w JavaScript

JavaScript jest **jednowątkowy**. Dlatego w niektórych sytuacjach, by nie blokować działania aplikacji, wykorzystujemy metody asynchroniczne. Tak jest też przy technikach AJAX.

Asynchroniczność oznacza, że do czasu rozstrzygnięcia/odpowiedzi/wykonania nie jest wstrzymywane działanie programu (strony). Zobaczmy to na przykład działanie metody `setTimeout`. Jeśli wywołamy `setTimeout(()=>{}, 2000)`, to oczywiście nie wstrzyma programu na 2 sekundy (wstrzymalibyśmy gdyby to była metoda synchroniczna). Program będzie działał dalej, a po 2 sekundach **funkcja callback** (funkcja podana jako pierwszy argument) trafi do **pętli zdarzeń (Event Loop)** - czyli ustawi się w kolejce do wykonania.



# Asynchroniczny - AJAX

Jeśli chodzi o asynchroniczność przy AJAX, to oczekiwanie na odpowiedź serwera nie powoduje zablokowania strony/aplikacji.

Wysyłamy żądanie (request) do serwera.

Aplikacja/Strona działa dalej (oczekiwanie na odpowiedź nie blokuje programu).

Gdy aplikacja/strona otrzyma odpowiedź, to kod do wykonania (jakaś funkcja) trafi do kolejki (pętli zdarzeń) i zostanie następnie wykonany np. zaktualizuje DOM bez odświeżania strony.



# XML i JSON - format przesyłania danych

XML (Extensible Markup Language) - dziś bardzo rzadko wykorzystywany do przesyłania danych. Od niego pochodzi akronim AJAX.

JSON (JavaScript Object Notation)- obecny standard do przesyłania danych. Dane które są wysyłane do klienta z serwera (czy od klienta do serwera) mają format JSON. Tak naprawdę dziś bardziej pasowałby skrótowiec AJAX.

```
<ArrayOfCenaZlota xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CenaZlota>
    <Data>2019-08-16</Data>
    <Cena>189.51</Cena>
  </CenaZlota>
</ArrayOfCenaZlota>
```

```
[{"data": "2019-08-16", "cena": 189.51}]
```

Źródło (endpointy):  
<http://api.nbp.pl/api/cenyzlota>  
<http://api.nbp.pl/api/cenyzlota?format=json>



# JSON - kluczowy format przesyłania danych

JSON jest formatem tekstowym, który łatwo (wsparcie natywne poprzez obiekt JSON zarówno w przeglądarce jak i Node.js) możemy parsować na obiekt JavaScript oraz zamieniać obiekt JavaScript na format JSON.

```
JSON.stringify({key:"value"})
```

```
-> '{"key":"value"}' //zamiana obiektu JavaScript na JSON (string)
```

```
JSON.parse('{ "key": "value" })
```

```
-> {key: "value"} //zamiana stringa (JSON) na obiekt JavaScript
```



# URL (Uniform Resource Locator) - budowa

`https://adres-strony.pl:443/path/to/subpage`

protokół - https, http czy ftp

host - adres-strony.pl

port - :443 (zawiera go wysyłany request, jest częścią adresu - domyślny 80 dla HTTP i 443 dla HTTPS)

ścieżka url - /path/to/subpage

Serwer na podstawie adresu URL, plus innych elementów przesyłanych w żądaniu, przede wszystkim metody HTTP (domyślnie GET) i parametrów, udzieli odpowiedzi.



# URL - query string i parametry

Dodatkowa część adresu url. Dodatkowe informacje dla serwera.

`https://adres-strony.pl:443/path/?users=100&gender=female`

query string: `?users=100&gender=female`

początku query stringa: `?`

łącznik parametrów: `&` (gdy więcej niż jeden parametr)

parametr (query parameter): `users=100` (para klucz-wartość)

Przykład: `https://allegro.pl/listing?string=rower` ; `https://www.x-kom.pl/szukaj?q=laptop&f[manufacturers][396]=1` ;





# Web serwer (HTTP serwer, serwer www)

Web serwer to program na serwerze obsługujący żądania i odpowiedzi oparte o protokół HTTP (zbiór reguł dla przesyłania danych m.in. budowę żądania i odpowiedzi).

- Serwer ma swój adres IP (nazwa hosta/domeny, prowadzi do tego adresu). Web serwer nasłuchuje żądanie (request) pod określonym portem.
- Przetwarza żądanie - analizuje żądanie (m.in. url) może też m.in. łączy się z bazą danych - mówimy tutaj o aplikacji serwerowej uruchomionej na web serwerze (web serwer napisany w Node.js jest sam w sobie aplikacją serwerową).
- Wysyła odpowiedź (response)



# Budowa żądania (request)

- ścieżka URL
- metoda HTTP (GET domyślnie, inne popularne to POST, PUT, DELETE)

GET /article/?id=202

- nagłówki - para klucz-wartość np {host: 'nasza-domena.pl:80'}
- body - gdy coś wysyłamy (np. dane z formularza)



# Odpowiedź (response)

- status (np. 200 - wszystko ok, dostaliśmy odpowiedź, 404 - nie odnalazłem zasobów, 500 - błąd serwera)

- headers np. jaki typ danych jest odsyłany

```
{"Content-type" : 'application/json'}
```

```
{"Content-type" : 'text/html'}
```

```
{"Content-type" : 'image/png'}
```

- body (czyli zawartość odpowiedzi)



# API (Application Programming Interface)

Zbiór narzędzie czy danych, które jeden program (np. web serwer) udostępnia innemu (np. przeglądarce, aplikacji mobilnej czy innemu web serwerowi).

Bardzo często mówiąc o API dla stron i aplikacji myślimy o RESTful API (często określamy też Web API, przy czym nie każde Web API musi być zgodne z zasadami REST). Web API (RESTful API) to dane/narzędzia udostępniane przez protokół HTTP za pomocą endpointów (adresów url) i dla RESTful API spełniające dodatkowe wymogi REST (kilka zasad - wpisz w Google "zasady REST" jeśli interesuje Cię ten temat).

Tworzenie Webowego API odbywa się oczywiście po stronie serwera - polega na przygotowaniu web serwer/aplikacji serwerowej na przyjęcie requesta (od przeglądarki/aplikacji) zawierającego dany adres url (endpoint) i metodę HTTP oraz przygotowaniu całej logiki potrzebnej do obsługi takiego żądania - zadanie dla backendowca.



# Endpoint

API jest najczęściej udostępniane przez zdefiniowany (na serwerze) adresy URL.

<https://jakis-host.com.pl/api/articles/id>

Nasz program wysyła request (żądanie) pod dany adres, określa również metodę np. GET (domyślna), POST (do przesyłania danych), PUT, DELETE czy inną. W zależności od wysłanej metody endpoint może zachować się inaczej.



# API publiczne, uwierzytelnienie i autoryzacja

API może być prywatne i publiczne. Bardzo wiele aplikacji udostępnia swoje publiczne API (Google, Facebook, Udemy itd.) dzięki czemu inne serwisy/aplikacje mogą korzystać z ich narzędzie/danych w swoich rozwiązaniach.

**Uwierzytelnianie** (autentykacja ang. authentication) - technika uwierzytelnienia (dostęp do API) to najczęściej API key lub OAuth.

**Autoryzacja** - Wiem kim jesteś, sprawdzę czy masz uprawnienia do tych konkretnie danych



# API FETCH

- obecny standard dla asynchronicznych połączeń zgodnych z protokołem HTTP
- dostarcza metodę fetch (fetch oznacza pobierać), która umożliwia wysyłanie requestów.
- dostarcza także obiekt odpowiedzi i metody do pracy z odpowiedzią (np. metodę json do pracy z JSON czy metodę redirect do przekierowań).



# Funkcja fetch

```
fetch('adres-zasobu')  
  .then((obiekt_odpowiedzi) => obiekt_odpowiedzi.json())  
  .then((dane) => /* wyświetl w dom */)  
  .catch((błąd) => /*wyświetl błąd */)
```

oparte na obietnicach (promises)







# Mały projekt - zrobmy

