Programowanie Internetowe

- Standard ECMA Script
- Obiektowy model dokumentu
- Data, Czas, Wyrażenia regularne
- Przechowywanie danych
- Przetwarzanie asynchroniczne

Opracował: inż. Grzegorz Petri

Przegląd zagadnień

- Czym jest Ajax
- Zasada działania
- Formaty danych
- Nawiązywanie połączenia
- Pobieranie i Wysyłanie danych
- Obsługa odpowiedzi i błędów
- Debugowanie kodu

Czym jest AJAX

Składowe technologii

AJAX oznacza **A**synchronous **J**avaScript **a**nd **X**ML

Technicznie składa się z następujących technologii:

- Obiekt **XMLHttpRequest** zaimplementowany w przeglądarkach
- **Java Script** – język skryptowy wykonywany po stronie klienta
- HTML **DOM** – obiektowe drzewo dokumentu umożliwiające interakcję z węzłami
- Dane w formacie: Text, XML lub JSON

Marketingowo Ajax jest:

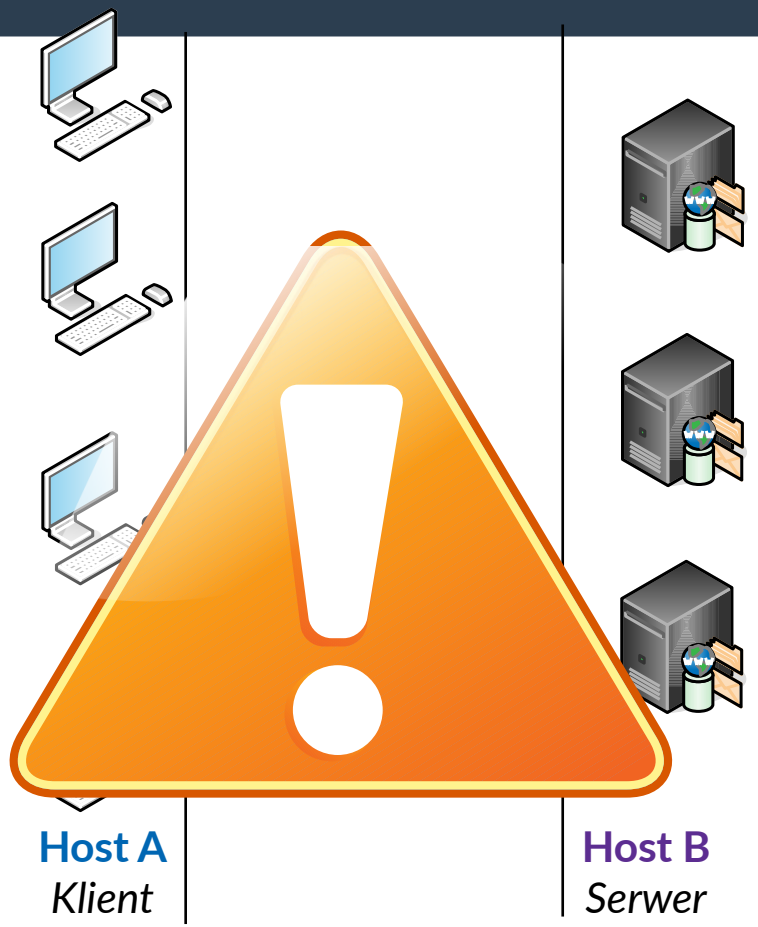
- rozwiązaniem problemu **Przerośniętych** stron WWW (*transfer dużych stron o wielu zasobach*)
- odpowiedzią na potrzeby **Responsywnych** stron i web aplikacji (*dynamiczne doładowywanie treści*)

Ajax **NIE JEST** językiem programowania

Zasada działania

Komunikacja

1. Wystąpienie w przeglądarce klienta zdarzenia aktywującego działanie AJAX
2. Stworzenie obiektu XHR lub Fetch do nawiązania połączenia przez JS
3. Wysłanie żądania do serwera Web
4. Przetwarzanie żądania klienta
5. Serwer odsyła odpowiedź do klienta *(kod, wraz z zasobem, jeżeli znaleziony)*
6. Klient otrzymuje odpowiedź do przetworzenia *(kod, i zasób, jeżeli jest)*
7. Przeglądarka wykonuje działanie na węźle DOM strony Web



Formaty danych

Do trzech razy sztuka

Format TXT

```
# Server settings
# DATA i GODZINA
user=root
port=3306
pass=secret
host=localhost
database=mydb
```

^ *Wymaga odpowiedniego mechanizmu parsującego w zależności od struktury przesłanych danych*

Format XML

```
<settings>
 <meta>
  <name>Server settings</name>
  <desc>DATA i GODZINA</desc>
 </meta>
 <data>
  <user>root</user>
  <port>3306</port>
  <pass>secret</pass>
  <host>localhost</host>
  <database>mydb</database>
 </data>
</settings>
```

^ *Format powstały dla AJAX'a – jednak posiada duży narzut na metadane w stosunku do danych*

Format JSON

```
{
"meta": {
  "name":"Server settings",
  "desc":"DATA i GODZINA"
 },
"data": {
  "user":"root",
  "port":"3306",
  "pass":"secret",
  "host":"localhost",
  "database":"mydb"
 }
}
```

^ *Faktyczna treść pliku znajduje się w pojedynczej linii*

Typ połączenia

Obsługiwane metody

- XMLHttpRequest (XHR)
  - Stworzony na początku wraz ideą AJAX
  - Wymaga większej ilości kodu
  - Nie zwraca obiektu FormData
- Fetch API (Fetch)
  - Nowe rozwiązanie z szerokim wsparciem (*nie działa w IE & Edge 12-13*)
  - Mniejsza ilość kodu, prostsze rozwiązanie
  - Domyślna obsługa: ***promise*** oraz interfejsy: ***Headers***, ***Response***, ***Request***
  - Obsługa Cache API
  - Odpowiedź jako streaming (*przyszłość*)

Metody obsługiwane przez XHR i Fetch:

- ✔ GET (*m.in. FORM*)
- ✔ POST (*m.in. FORM*)
- ✔ PUT (*poł. asynchroniczne JS*)
- ✔ PATCH (*j/w*)
- ✔ DELETE (*j/w*)

Odbieranie danych

Komunikacja XHR oraz Fetch API

```
// stworzenie obiektu do komunikacji
const xhr = new XMLHttpRequest();
// ustawienie formatu danych odpowiedzi
xhr.responseType = "json"; // domyślny text
// formaty: {text,arraybufer,blob,document,json}
// skonfigurowanie połączenia
xhr.open(mathod, /* GET, POST, PUT, DELETE */
         url,    /* adres URL zasobu */
         async,  /* TRUE=async,FALSE=sync */
         login,  /* kiedy wymagane jest */
         passwd); /*uwierzytelnienie HTTP Basic*/
xhr.send(data=null);  // "wysłanie" połączenia
// GET - pobieranie: data = null
// POST - wysyłanie: data = document.form[]
xhr.response; // brak danych..
// ..wymagany nasłuch zdarzenia: load
xhr.addEventListener("load",e=>{
   // kod weryfikujący czy..
   xhr.response; // dane są
});
```

```
fetch(url, [options]);
fetch(url)
  .then(response => {
   // kod weryfikujący czy..
   response    // odpowiedź jest
   } )
  .then(response => {reponse.json()})
// obiekt Response = {
//  ok:{}, status:{}, statusText:{},
//  type:{}, url:{}, body:{}
//}
  .catch(error => {error});
// gdy np. nie ma dostępu do sieci
```

Obsługa odpowiedzi

Kody HTTP oraz HttpXMLRequest

Kody odpowiedzi protokołu HTTP:

- 200 – OK, odebrano zasób
- 301/ 307/308 – zasób przeniesiono...
- 403 / 404 – autoryzuj się, brak zasobu
- 500-505 – kody błędów serwera
- *Obsługa innych kodów*

```
xhr.addEventListener(
  "readystatechange", e => {
    if (xhr.readyState === 4) {
      if (xhr.status === 200)
        // otrzymano zasób
      if (xhr.status === 500)
        // możliwa większa awaria - odpuść "dzisiaj"
      if (xhr.status === 503)
        // spróbuj za kilka chwil
    }
});
```

Stany połączenia właściwości **`readyState`**:

- **`0`** – połączenie **NIE** nawiązane
- **`1`** – połączenie nawiązane
- **`2`** – żądanie odebrane
- **`3`** – przetwarzanie
- **`4`** – *dane zwrócone i gotowe do użycia*

```
xhr.addEventListener(
  "readystatechange", e => {
    if (xhr.readyState !== 4) {
      // komunikat dla użytkownika
    }
    if (xhr.readyState === 4) {
      // obsługa danych, renderowanie, itp
    }
});
```

Wysyłanie danych

Komunikacja XHR oraz Fetch API

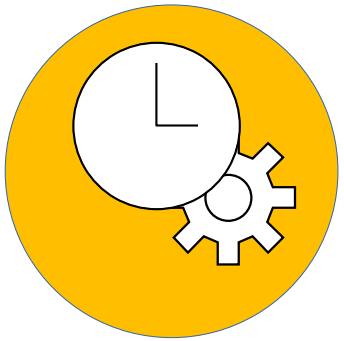
```
form.addEventListener("submit", e=>{
    e.preventDefault();// blokuj przeładowanie
    const xhr = new XMLHttpRequest();
    xhr.addEventListener("load", e=>{
        if (xhr.status === 200)
            console.log(xhr.response);
    });
    xhr.open("POST", "./odbierz.php", true);
// typ przesyłanej zawartości (1z3)
    xhr.setRequestHeader("Content-type",
    "application/x-www-form-urlencoded");
// lub: multipart/form-data (np. pliki)
    xhr.send(encodeURI(
      `name=${inputName.value}
      &surname=${inputSurname.value}`));
});
```

```
fetch(url, {
        method: "post",
        headers: {
          "Content-type":
          "application/x-www-form-urlencoded"
        },
        body: encodeURIComponent(
              "name=Marcin&surname=Nowak")
        // lub obiekt formData
    })
    .then(response => response.json())
    .then(response => {
    // wykonano pomyślnie
        console.log(response);
    });
```

Zdarzenia dla XHR i Fetch

Events

- ✔ `load`
- ✔ `loadstart`
- ✔ `progress`
- ✔ `error`
- ✔ `abort`
- ✔ `loadend`
- ✔ `readystatechange`
- ✔ `timeout`



Obsługa błędów

XHR oraz Fetch

Diagnozowanie błędów w XHR:

- **console.log()** oraz **Komunikaty** w UI
- Obsługa wszystkich **kodów** HTTP ...
- ... wszystkich stanów **`readyState`**
- Narzędzia developerskie przeglądarki
- Powtórna analiza kodu (*najlepiej przez kogoś innego niż jego autor*)
- Użycie **biblioteki** lub **Fetch API**

Diagnozowanie błędów w Fetch API

1. Próbowałeś sprowadzić nieistniejący zasób
2. Nie masz prawa sprowadzać zasobów (*fetch*)
3. Błędnie wprowadziłeś argumenty
4. Serwer rzucił w nas błędem
5. Serwer Cie zignorował (*nie odpowiedział w zadanym czasie*)
6. Serwer gra w candy CRUSH (*uległ awarii*)
7. API uległo zmianie
8. ... cokolwiek ERROR

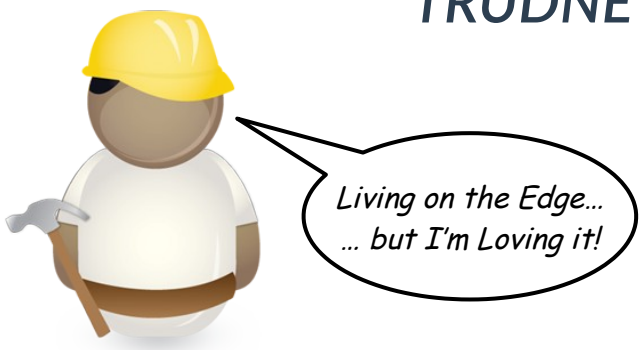
Debugowanie

Techniczne naprawianie błędów

Manualne debugowanie

- DevTools & Breakpoints
- JS console.log();
- PHP echo / print() *
- PHP display_error() **

TRUDNE i czasochłonne!



Automatyczne i półautomatyczne

- **Breakpoints** – punkty wstrzymania pracy skryptu; narzędzia developerskie przeglądarki
- **Postman** – aplikacja/usługa; najlepsze rozwiązanie dla prawdziwego developera
- **Testy** – rozpoczęcie prac nad projektem od napisania testów

```
dialog(
   'Pytania?'
);
```