Technologia AJAX oraz Fetch API

1. Przykład 'Hello World'

1. Wykonaj komendy:

```
npm install express
npm install pug
npm install morgan
```

2. Utwórz plik 'server1.js' o następującej zawartości:

```
const express = require('express');
const logger = require('morgan');
const app = express();

// Configuring the application
app.set('views', __dirname + '/views');
app.set('view engine', 'pug');
app.locals.pretty = app.get('env') === 'development';

// Determining the contents of the middleware stack
app.use(logger('dev'));
// app.use(express.static(__dirname + '/public'));

// *** Route definitions ***

// The first route
app.get('/', function (req, res) {
    res.render('index');
});
```

```
// The second route
app.get('/submit', function (req, res) {
    // Return the greeting in the format preferred by the WWW client
    switch (req.accepts(['html', 'text', 'json', 'xml'])) {
        case 'json':
            // Send the JSON greeting
            res.type('application/json');
            res.json({ welcome: "Hello World" });
            console.log("The server sent a JSON document to the browser");
            break;

        case 'xml':
            // Send the XML greeting
            res.type('application/xml');
            res.send('<welcome>Hello World</welcome>');
            console.log("The server sent an XML document to the browser");
            break;

        default:
            // Send the text plain greeting
            res.type('text/plain');
            res.send('Hello World');
            console.log("The server sent a plain text to the browser");
    }
});

// The application is to listen on port number 3000
app.listen(3000, function () {
    console.log('The application is available on port 3000');
});
```

3. Utwórz plik 'views/index.pug' o następującej zawartości:

```
doctype html
html(lang='en')
    head
        meta(charset='UTF-8')
```

```
title
    | Form
script.
    /*****************************************************************************/
    /* Function that performs (asynchronous) query to the web server using AJAX */
    /*****************************************************************************/
    function requestAJAX() {
        //-----------------------------------------------
        // Create an object representing the request to the web server – see https://developer.mozilla.org/docs/Web/A
        //-----------------------------------------------
        const xhr = new XMLHttpRequest();

        //-----------------------
        // Observers registration
        //-----------------------

        // If the request was successful
        xhr.addEventListener("load", function (evt) {
            if (xhr.status === 200) {
                window.alert(xhr.response);
                console.log(xhr.response);
            }
        });

        // If the request failed
        xhr.addEventListener("error", function (evt) {
            window.alert('There was a problem with this request.');
        });

        //-------------------------------------------------------------------------
        // Configuration and execution of the (asynchronous) query to the web server
        //-------------------------------------------------------------------------

        //*************************************************************
        // Type of data received by the 'load' observer - default - 'text'
        //*************************************************************
```

```
    // Uncomment one of the lines below
    //*******************************
    // xhr.responseType ='json';
    // xhr.responseType ='document';

    xhr.open('GET', '/submit', true);

    //*********************************************************************************
    // What is the acceptable data type - the server part should return data of the given type
    // Default value: '*/*'
    //*********************************************************************************
    // Uncomment one of the lines below
    //*******************************
    // xhr.setRequestHeader('Accept', 'application/json');
    // xhr.setRequestHeader('Accept', 'application/xml');

    xhr.send(null);
}

/*****************************************************************************/
/* Function that performs (asynchronous) query to the web server using Fetch API */
/*****************************************************************************/
function requestFetchAPI() {
    fetch('/submit', {
        headers: {
            //*********************************************************************************
            // What is the acceptable data type - the server part should return data of the given type
            // Default value: '*/*'
            //*********************************************************************************
            // Uncomment one of the lines below
            //*******************************
            // 'Accept': 'application/json'
            // 'Accept': 'application/xml'
        }
    }) // Execution of the (asynchronous) query to the web server – a promise is created
    .then(function (response) { // if the promise is fulfilled
```

```
        if (!response.ok)
            throw Error(response.statusText);
        if (response.headers.get("Content-Type") !== 'application/json') {
            // If the received data is plain text or an XML document
            const result = response.text();
            window.alert(result); // show the Promise object
            console.log(result);
        }
        else {
            //If the received data is a JSON document
            const result = response.json();
            window.alert(result); // show the Promise object
            console.log(result);
        }
    })
    .catch(function (error) { // if the promise is rejected
        window.alert(error);
    });
}

/*********************************************/
/* Same as above but using 'async' and 'await' */
/*********************************************/
/*
async function requestFetchAPI() {
    try {
        response = await fetch('/submit', {
            headers: {
                // 'Accept': 'application/json'
                // 'Accept': 'application/xml'
            }
        }); // Execution of the (asynchronous) query to the web server – a promise is created

        // If the promise is fulfilled, then 'response' has a value
        if (!response.ok)
            throw Error(response.statusText);
```

```
                if (response.headers.get("Content-Type") !== 'application/json') {
                    // If the received data is plain text or an XML document
                    const result = response.text();
                    window.alert(result); // show the Promise object
                    console.log(result);
                }
                else {
                    //If the received data is a JSON document
                    const result = response.json();
                    window.alert(result); // show the Promise object
                    console.log(result);
                }
            }
            catch (error) { // if the promise is rejected
                window.alert(error);
            }
        }
        */

body
    main
        form(method='get' action='/submit')
            label
                | Perform a query to the web server with the
                strong GET
                |  method
            input(type='submit' value='Without using AJAX or Fetch API')
            input(type='button' value='Using AJAX' onclick='requestAJAX()')
            input(type='button' value='Using Fetch API' onclick='requestFetchAPI()')
```

4. Uruchom serwer komendą `npx nodemon server1`
5. Wpisz w przeglądarce adres http://localhost:3000/
6. Naciśnij przycisk "`Without using AJAX or Fetch API`" i zobacz co wyświetlają:
    - Przeglądarka WWW
    - Konsola przeglądarki WWW

    - Terminal w którym jest uruchomiony skrypt 'server1.js' — konsola
7. Naciśnij przycisk "`Using AJAX`" i zaobserwuj to, co powyżej
8. Naciśnij przycisk "`Using Fetch API`" i zaobserwuj to, co w pkt. 6 — zmodyfikuj skrypt tak, aby zamiast napisu "`[object Promise]`" wyświetlała się właściwa treść, tzn. napis "`Hello World`"
9. Zatrzymaj działanie serwera, a następnie sprawdź co się stanie po naciśnięciu każdego z powyższych przycisków
10. Ponownie uruchom serwer — `npx nodemon server1`
11. Wpisz w przeglądarce adres http://localhost:3000/submit?imie=twoje_imię
12. Przeczytaj fragment artykułu nt. funkcji `encodeURI*()` — być może będziesz ich musiał/a użyć w swoim skrypcie
13. Zmodyfikuj skrypt 'server1.js' tak, aby:
    - Wyświetlał (w konsoli) Twoje imię, które przeglądarka przekazała mu w powyższym URL
    - Wysyłał przeglądarce, zamiast "`Hello World`", napis "`Witaj <twoje_imię>`"
14. Zmodyfikuj plik 'views/index.pug':
    - Dodaj drugi formularz (brak atrybutów 'method' oraz 'action' w znaczniku `<form>`) zawierający:
        - pole tekstowe o nazwie 'imie'
        - przycisk — jego naciśnięcie ma powodować wysłanie (metodą 'GET"), do serwera, Twojego imienia, odczytanego z pola formularza, w oparciu o URL z pkt. 11, a następnie wyświetlenie odebranego tekstu powitania za pomocą `window.alert()`

      Sprawdź, czy skrypt serwerowy wyświetla prawidłowo dane dla przypadku gdy pole 'imie' zawiera:
        - polskie znaki w standardzie UTF-8
        - znaki: &, @, + lub spacja

      Jeżeli w ww. przypadkach serwer wyświetla dane nieprawidłowo, to spróbuj zastosować, po stronie klienta, funkcje `encodeURI*()`

15. Dodaj przycisk — jego naciśnięcie ma powodować wysłanie zawartości pola tekstowego (imienia) metodą "POST"

    Zwracam uwagę, że:
    - W przypadku "POST" przesyłane dane nie mogą być elementem URL — muszą być umieszczone w ciele żądania HTTP, czyli w przypadku AJAX-a, muszą być na liście argumentów metody `xhr.send()`, a nie `xhr.open()`. Dodatkowo, za pomocą `xhr.setRequestHeader()` należy określić sposób kodowania danych zawartych w ciele żądania — patrz slajd z wykładu
    - Skrypt 'server1.js' nie potrafi obsługiwać metody "POST" — musisz go uzupełnić o dodatkowy fragment kodu

16. Dodaj dwa przyciski: oraz

17. Korzystając z opis 1 lub opis 2, dopisz obsługę tych przycisków — mają przesyłać aktualną treść pola formularza za pomocą metody, odpowiednio, "GET" / "POST", korzystając z interfejsu Fetch API. Po odebraniu tekstu powitania ("`Witaj <twoje_imię>`"), ma on być wyświetlony tak jak poprzednio, tzn. za pomocą `window.alert()`

Tak więc masz utworzyć skrypty (w pliku 'views/index.pug') obsługujące poniższe przypadki:

1. AJAX + metoda GET
2. AJAX + metoda POST
3. Fetch API + metoda GET
4. Fetch API + metoda POST

2. Odbieranie danych w formacie JSON oraz XML

1. Przeczytaj informacje o CORS oraz dlaczego ten mechanizm został wprowadzony

   Jeżeli będziesz korzystał(a) z Fetch API, to informacje na temat używania CORS znajdziesz na stronie Using Fetch

2. Utwórz plik 'server2.js' o następującej zawartości:

```
const express = require('express');
const logger = require('morgan');
/************* */
const app_3000 = express();
app_3000.use(logger('dev'));
app_3000.get('/', function (req, res) {
    res.send('Response from 3000');
```

```
});
app_3000.listen(3000, function () {
    console.log('The application is available on port 3000');
});
/************* */
const app_3001 = express();
app_3001.use(logger('dev'));
app_3001.get('/', function (req, res) {
    res.send('Response from 3001');
});
app_3001.listen(3001, function () {
    console.log('The application is available on port 3001');
});
/************* */
console.log("To stop the server, press 'CTRL + C'");
```

Pokazany skrypt uruchamia serwer na dwóch portach: 3000 oraz 3001

3. Zmodyfikuj powyższy skrypt — w przypadku połączenia się do portu 3000 (http://localhost:3000/) serwer ma wygenerować dokument HTML zawierający poniższe trzy składniki:
    1. Formularz składajacy się z dwóch pól tekstowych (*area* oraz *location*) oraz przycisku *Pobierz*
    2. Dwa elementy 'div':

```
<h1>Remote</h1>
<div id='remote'>
Remote date and time
</div>
<!-- ***************** -->
<h1>Local</h1>
<div id='local'>
Local date and time
</div>
```

    3. Skrypt JS (przeglądarkowy), uruchamiany naciśnięciem przycisku *Pobierz*, który:
        1. Korzystając z **DOM 4** zastępuje tekstową zawartość elementu *div id='remote'* napisem "`Downloading data`"

2. Łączy się (AJAX lub Fetch API) z usługą sieciową World Time, a następnie pobiera (**w formacie JSON**), w oparciu o API tej usługi, aktualną datę oraz czas dla podanego, w formularzu, *area* oraz *location*

   > Powyższy serwer obsługuje CORS więc nie powinno być problemu z pobraniem danych, ale mogą się pojawić problemy wydajnościowe — serwer może być przeciążony

3. Zastępuje zawartość tekstową elementu *div id='remote'* otrzymanymi danymi — zdalna data oraz czas; w przypadku gdy serwer jest obciążony (tzn. gdy zwrócił dokument HTML, którego zawartością jest napis "`This website is currently experiencing high load.`"), to zastępuje (zawartość tekstową) napisem "`The server is overloaded`"

4.

1. Zmodyfikuj skrypt 'server2.js' — na porcie 3001 (http://localhost:3001/) ma oferować funkcjonalność lokalnego serwera czasu, tzn. ma:
   1. Odczytać aktualną (lokalną) datę oraz czas — obiekt Date
   2. Wygenerować dokument HTML spełniający wymagania stawianie dokumentom XML dobrze uformowanym (ang. well-formed) — przykład:

      ```
      <div>
         <span id='date'> <!-- date -->  </span>
         <span id='time'> <!-- time -->  </span>
      </div>
      ```

   3. Wysłać wygenerowany dokument
2. Zmodyfikuj skrypt przeglądarkowy — ma on:
   1. Po naciśnięciu przycisku *Pobierz*, nawiązać połączenie, nie tylko z serwerem *World Time API*, ale również z lokalnym serwerem czasu http://localhost:3001/
   2. Pobrać lokalną datę oraz czas — dokument HTML z datą oraz czasem
   3. Korzystając z **DOM 4**, zastąpić zawartość tekstową elementu *div id='local'* otrzymanymi danymi — lokalna data oraz czas

   > - Odbierz przesłany dokument, jako dane XML, korzystając z AJAX lub Fetch API
   > - Jeżeli używasz AJAX, to skorzystaj z `xhr.responseType ='json';` / `xhr.responseType ='document';`, aby otrzymać odebrane dane w postaci obiektu 'JSON' / 'XMLDocument'. Ponieważ otrzymane dane nie są napisem — nie musisz ich już parsować
   > **Wskazówka:** W przypadku obiektu 'XMLDocument' przydatna może być własność documentElement
   > - Ponieważ skrypt znajdujący się pod adresem **http://localhost:3000/** próbuje pobrać dane ze strony **http://localhost:3001/** — nie jest spełniona reguła tego samego pochodzenia, dlatego musisz wzbogacić skrypt 'server2.js' o obsługę CORS

5. Zmodyfikuj skrypty (przeglądarkowy oraz serwerowy) z poprzednich ćwiczeń tak, aby otrzymać funkcjonalność określoną przez prowadzącego. Założe
   - Metoda przekazu danych z formularza: GET lub POST
   - Dane otrzymane od serwera: zwykły tekst, JSON lub HTML
   - Sposób działania obydwu skryptów oraz dokładna postać generowanych danych zostanie podana na początku ćwiczeń