✓ Wykonano

Przykład zastosowania komunikacji asynchronicznej

Implementacja ściany à la Facebook Wall:

1. Zdalny serwer, obsługujący CORS, udostępnia przykładowe dane (tu: posty) w formacie JSON
2. Za pomocą Fetch API skrypt łączy się z tym serwerem, a następnie pobiera *N* postów
3. Przy użyciu DOM modyfikowana jest treść strony WWW — wyświetlane są pobrane posty
4. Jeżeli użytkownik przewinie treść strony aż do ostatniego posta, pobieranych jest kolejnych *N* postów

Szczegółową implementację pokazano na ponizszym filmie.

🏠 1. Zapytania AJAX oraz FetchAPI

1. Wykonaj komendy:

```
mkdir -p cw6/views
cd cw6
npm init es6
npm install express pug morgan entities
```

2. Utwórz skrypt *app1.js* o poniższej zawartości:

```
/**
 * @author Stanisław Polak <polak@agh.edu.pl>
 */

import express from 'express';
import morgan from 'morgan';
import { encodeXML } from 'entities';

const app = express();

app.set('view engine', 'pug');
app.locals.pretty = app.get('env') === 'development';
/* ********************************************** */
morgan.token('accept', function (req, res) { return req.headers['accept'] })
app.use(morgan(':method :url :status\tAccept: :accept'));
app.use(express.urlencoded({ extended: false }));
/* ********************************************** */
app.get('/', function (request, response) {
    response.render('index');
});

app.all('/submit', function (req, res) {
    let name = ['GET','DELETE'].includes(req.method) ? req.query.name: req.body.name;

    console.log();
    console.log('--------------');
    console.count('Request');
    console.log('--------------');
    console.group('\x1B[35mreq.query\x1B[0m');
    console.table(req.query);
    console.groupEnd();
    console.group('\x1B[35mreq.body\x1B[0m');
    console.table(req.body);
    console.groupEnd();
    console.group('\x1B[35mname\x1B[0m');
    console.log(name);
    console.groupEnd();
    console.log();

    // 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 '${name}'` });
            console.log(`The server sent a \x1B[31mJSON\x1B[0m document to the browser for the request below`);
            break;

        case 'xml':
            // Send the XML greeting
            name = name !== undefined ? encodeXML(name) : '';
            res.type('application/xml');
            res.send(`<welcome>Hello '${name}'</welcome>`);
            console.log(`The server sent an \x1B[31mXML\x1B[0m document to the browser for the request below`);
            break;

        default:
            // Send the text plain greeting
            res.type('text/plain');
            res.send(`Hello '${name}'`);
            console.log(`The server sent a \x1B[31mplain text\x1B[0m document to the browser for the request below`);
```

```
    }
});
/* ********************************************** */
app.listen(8000, function () {
    console.log('The server was started on port 8000');
    console.log('To stop the server, press "CTRL + C"');
});
```

Powyższy skrypt implementuje serwer — w zależności od wartości nagłówka **Accept** (w żądaniu klienta), wysyła on odpowiedź do przeglądarki WWW w jednym z trzech formatów:

1. Dokument JSON
2. Dokument XML
3. Zwykły tekst

3. Utwórz szablon *views/index.pug* o poniższej zawartości

```
1.  //- @author Stanisław Polak <polak@agh.edu.pl>
2.
3.  doctype html
4.  html(lang='en')
5.      head
6.          meta(charset='UTF-8')
7.          title Form
8.          link(rel="stylesheet" href="https://cdn.jsdelivr.net/npm/mocha/mocha.css")
9.          style.
10.             table {
11.                 width: 100%;
12.             }
13.             td {
14.                 border: 1px solid #000;
15.                 padding: 15px;
16.                 text-align: left;
17.             }
18.             th {
19.                 background-color: #04AA6D;
20.                 color: white;
21.             }
22.         script.
23.
/*****************************************************************************************/
24.             /* Function that retrieves the content of one of the selected text fields of
*/
25.
/*****************************************************************************************/
26.             function getName(http_method){
27.                 let name = '';
28.
29.                 // TODO: Here put the code that, depending on the value of the 'http_meth
variable — GET / POST — assigns the 'name' variable to the value of the 'name_GET' / 'nam
form field
30.
31.                 return name;
32.             }
33.
34.
/*****************************************************************************************/
35.             /* Function that performs (asynchronous) query to the web server using AJAX
*/
36.
/*****************************************************************************************/
37.             /* http_method ∈ ["GET", "POST"]
*/
38.             /* response_type ∈ ["text", "json", "document"]
*/
39.             /* name - Contents of the form's text box - data that needs to be sent asynch
*/
40.
/*****************************************************************************************/
41.             function requestAJAX(http_method, response_type, name, show_alert=false) {
42.                 //-----------------------------------------------
43.                 // Create an object representing the request to the web server — see
https://developer.mozilla.org/docs/Web/API/XMLHttpRequest
44.                 //-----------------------------------------------
45.                 const xhr = new XMLHttpRequest();
46.
47.                 //-----------------------
48.                 // Observers registration
49.                 //-----------------------
50.
51.                 // If the request was successful
52.                 xhr.addEventListener("load", function (evt) {
53.                     if (xhr.status === 200) {
```

```
                        console.group('AJAX');
                        console.log(`HTTP method →\t\t${http_method}\nResponse type →\t${response_type}\nInput data →\t\t${name}`);
                        console.log(xhr.response);
                        console.groupEnd();
                        if(show_alert)
                            window.alert(xhr.response);
                        else {
                            results.set(`ajax ${http_method} ${response_type}`, xhr.respo
                            dispatchEvent(received);
                        }
                    }
                });

                // If the request was 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 se
                //--------------------------------------------------------------------------
                xhr.responseType = response_type;  // Type of data received by the 'load
                xhr.withCredentials = true; // Do not modify or remove

                //*********************************
                // Specifying connection parameters
                //*********************************
                if(http_method === 'GET') {
                    xhr.open('GET', 'http://localhost:8000/submit', true); // TO BE MODI
                }
                if(http_method === 'POST'){
                    xhr.open('POST', 'http://localhost:8000/submit', true);
                }

//*****************************************************************************************
                // What is the acceptable data type - the server part should return data
given type
                // Default value: '*/*'

//*****************************************************************************************
                switch(response_type){
                    case 'json':
                        xhr.setRequestHeader('Accept', 'application/json');
                        break;
                    case 'document':
                        xhr.setRequestHeader('Accept', 'application/xml');
                        break;
                }

                //*****************************
                // Making an asynchronous call
                //*****************************
                if(http_method === 'GET') {
                    xhr.send(null);
                }
                if(http_method === 'POST') {
                    // TO BE ADDED: you must specify the value of the 'Content-type' hea
must inform the server that the body content contains data of the "application/x-www-form
urlencoded" type
                    xhr.send(null); // TO BE MODIFIED
                }
            }

```

```
112.
/*****************************************************************************************/
113.              /* Function that performs (asynchronous) query to the web server  usingFetch
*/
114.
/*****************************************************************************************/
115.              /* http_method ∈ ["GET", "POST"]
*/
116.              /* response_type ∈ ["text", "json", "xml"]
*/
117.              /* name - Contents of the form's text box - data that needs to be sent asynch
*/
118.
/*****************************************************************************************/
119.              function requestFetchAPI(http_method, response_type, name, show_alert=false)
120.                  let accept = '*/*';
121.
122.                  switch(response_type){
123.                      case 'json':
124.                          accept = 'application/json';
125.                          break;
126.                      case 'xml':
127.                          accept = 'application/xml';
128.                          break;
129.                  }
130.                  //-------------------------------------------------------------------------
131.                  // Configuration and execution of the (asynchronous) query to the web ser
132.                  //-------------------------------------------------------------------------
133.                      ((http_method) => {
134.                          if(http_method === 'GET')
135.                              return fetch('http://localhost:8000/submit', { // TO BE MODI
136.                                  method: 'GET',
137.                                  credentials: "include", // Do not modify or remove
138.                                  headers: {
139.
//*****************************************************************************************
140.                                      // What is the acceptable data type—the server part s
return data of the given type
141.
//*****************************************************************************************
142.                                      Accept: accept
143.                                  }
144.                              });
145.                          if(http_method === 'POST')
146.                              return fetch('http://localhost:8000/submit', {
147.                                  method: 'POST',
148.                                  credentials: "include", // Do not modify or remove
149.                                  // TO BE ADDED - you need to determine the content of the
150.                                  headers: {
151.                                      // TO BE ADDED: you must specify the value of the '
type' header — you must inform the server that the body content contains data of the
"application/x-www-form-urlencoded" type
152.
153.
//*****************************************************************************************
154.                                      // What is the acceptable data type—the server part s
return data of the given type
155.
//*****************************************************************************************
156.                                      Accept: accept
157.                                  }
158.                          });
159.
160.                  })(http_method) // a promise is returned
161.                  .then(function (response) { // if the promise is fulfilled
162.                      if (!response.ok)
```

```
                        throw Error(response.statusText);

                    console.group('Fetch API');
                    console.log(`HTTP method →\t\t${http_method}\nResponse type →\t${response_type}\nInput data →\t\t${name}`);
                    let result;

                    if (!response.headers.get('content-type')?.includes('application/jso
                        // If the received data is plain text or an XML document
                        result = response.text();
                    }
                    else {
                        //If the received data is a JSON document
                        result = response.json();
                    }
                    console.log(result);
                    console.groupEnd();
                    if(show_alert)
                        window.alert(result);
                    else {
                        results.set(`fetch ${http_method} ${response_type}`, result);
                        dispatchEvent(received);
                    }
                })
                .catch(function (error) { // if the promise is rejected
                    window.alert(error);
                });
            }
        script(src="https://cdn.jsdelivr.net/npm/mocha/mocha.js")
        script(type="module").
            import { expect } from 'https://cdn.jsdelivr.net/npm/chai/chai.js'
            window.expect = expect
    body
        script(class="mocha-init").
            mocha.setup('bdd');
            mocha.checkLeaks();
        main
            table
                tr
                    th
                    th GET
                    th POST
                tr
                    th(colspan='3' style=' background-color: #04556D;') Without AJAX and
                tr
                    th HTTP
                    td
                        form(action="http://localhost:8000/submit" method="GET")
                            label(for="name_GET") Your name
                            input(type="text" id="name_GET" name="name")
                            br
                            input(type="submit" value="text")
                    td
                        form(action="http://localhost:8000/submit" method="POST")
                            label(for="name_POST") Your name
                            input(type="text" id="name_POST" name="name")
                            br
                            input(type="submit" value="text")
                tr
                    th(colspan='3' style=' background-color: #04556D;')  Asynchronous re
                tr
                    th AJAX
                    each method in ["GET", "POST"]
                        td
```

```
                                each type in ["text", "json", "document"]

                                    button(onclick=`console.clear() ; requestAJAX("${method}","${type}", getName('${method}'), true)`) #{type}
                tr
                    th Fetch API
                    each method in ["GET", "POST"]
                        td
                            each type in ["text", "json", "xml"]
                                button(onclick=`console.clear() ; requestFetchAPI("${method}","${type}", getName('${method}'), true)`) #{type}
        h1 Unit tests
        button(onclick='window.location.reload();') Restart
        div(id="mocha")
        script.
            const name = 'John Doe a/?:@&=+$#';

            if(window.location.port == 8000) {
                window.addEventListener("load", (event) => {
                    for(let method of ["GET","POST"]){
                        for(let type of ["text", "json", "document"])
                            requestAJAX(method, type, name);
                        for(let type of ["text", "json", "xml"])
                            requestFetchAPI(method, type, name);
                    }
                })
            };
        script(class="mocha-exec").
//************************************************************************************
            // Unit tests
//************************************************************************************
            var results = new Map();
            var received = new Event('received');
            var test_executed = false;

            function decodeHtml(html) {
                var txt = document.createElement("textarea");
                txt.innerHTML = html;

                return txt.value;
            }

            addEventListener('received', (e) => {
                if(!test_executed && results.size === 12){
                    const parser = new DOMParser();
                    const xml_document= parser.parseFromString("<welcome>Hello 'John Doe a/?:@&amp;=+$#'</welcome>","text/xml");

                    describe('AJAX requests', function() {
                        it(`Returns "Hello '${name}'" for requestAJAX('GET','text')`, fu
                            expect(results.get('ajax GET text')).to.equal(`Hello '${name
                        });
                        it(`Returns "Hello '${name}'" for requestAJAX('GET','json')`, fu
                            expect(results.get('ajax GET json')).to.eql({welcome: `Hello '${name}'`});
                        });
                        it(`Returns "Hello '${name}'" for requestAJAX('GET','document')`, function() {
                            expect(results.get('ajax GET document').documentElement.firstChild.data).to.equal(xml_document.documentElement.firstCh
                        });
                        it(`Returns "Hello '${name}'" for requestAJAX('POST','text')`, fu
                        {
```

```
                                expect(results.get('ajax POST text')).to.equal(`Hello '${name
                            });
                            it(`Returns "Hello '${name}'" for requestAJAX('POST','json')`, fu
{
                                expect(results.get('ajax POST json')).to.eql({welcome: `Hello
'${name}'`});
                            });
                            it(`Returns "Hello '${name}'" for requestAJAX('POST','document')
function() {
                                expect(results.get('ajax POST
document').documentElement.firstChild.data).to.equal(xml_document.documentElement.firstCh
                            });
                        });

                        describe('Fetch API requests', function() {
                            it(`Returns "Hello '${name}'" for requestFetchAPI('GET','text')`,
function() {
                                const result  = await results.get('fetch GET text')
                                expect(result).to.equal(`Hello '${name}'`);
                            });
                            it(`Returns "Hello '${name}'" for requestFetchAPI('GET','json')`,
function() {
                                const result  = await results.get('fetch GET json')
                                expect(result).to.eql({welcome: `Hello '${name}'`});
                            });
                            it(`Returns "Hello '${name}'" for requestFetchAPI('GET','xml')`,
function() {
                                const result  = await results.get('fetch GET xml');
                                expect(decodeHtml(result)).to.equal(`<welcome>Hello
'${name}'</welcome>`);
                            });
                            it(`Returns "Hello '${name}'" for requestFetchAPI('POST','text')
function() {
                                const result  = await results.get('fetch POST text')
                                expect(result).to.equal(`Hello '${name}'`);
                            });
                            it(`Returns "Hello '${name}'" for requestFetchAPI('POST','json')
function() {
                                const result  = await results.get('fetch POST json')
                                expect(result).to.eql({welcome: `Hello '${name}'`});
                            });
                            it(`Returns "Hello '${name}'" for requestFetchAPI('POST','xml')`,
function() {
                                const result  = await results.get('fetch POST xml');
                                expect(decodeHtml(result)).to.equal(`<welcome>Hello
'${name}'</welcome>`);
                            });
                        });

                        mocha.run();
                        test_executed = true;
                    }
                });
```

Powyższy szablon zawiera, między innymi, definicje dwóch funkcji JavaScript (`requestAJAX()` oraz `requestFetchAPI()`) pozwalających na wykonywanie asynchronicznych zapytań HTTP do serwera

4. Uruchom aplikację — `node app1`
5. Wykonaj, pojedynczo, poniższe komendy curl

```
# Use of basic HTTP methods
curl --include http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
curl --include --header "Accept: application/json" http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
curl --include --header "Accept: application/xml"  http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
curl --include --request POST --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
curl --include --request POST --header "Accept: application/json" --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
curl --include --request POST --header "Accept: application/xml"  --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
# Use of other HTTP methods
curl --include --request DELETE http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
curl --include --request DELETE --header "Accept: application/json" http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
curl --include --request DELETE --header "Accept: application/xml" http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
curl --include --request PUT --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
curl --include --request PUT --header "Accept: application/json" --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
curl --include --request PUT --header "Accept: application/xml" --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
```

6. Zaobserwuj:
   - Co wyświetla terminal, w którym jest uruchomiony skrypt 'app1.js' — co wyświetla konsola serwera?
   - Jaka jest wartość nagłówka odpowiedzi "Content-Type" — typ MIME — w każdym z poniższych wywołań tej komendy oraz w jakim formacie są dane zawarte w ciele odpowiedzi?

```
$ curl --include ...
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: typ MIME
...

Treść ciała odpowiedzi
```

7. Wybrane znaki przestankowe, zawarte w URL, są metaznakami — sprawdź, co wyświetlają ww. konsole dla następujących wywołań *curl*:

```
curl 'http://localhost:8000/submit?name=Jane&0a=4' ; echo # Użyto znaku przestankowego '&'
curl 'http://localhost:8000/submit?name=Jane+0a=4' ; echo # Użyto znaku przestankowego '+'
curl 'http://localhost:8000/submit?name=Jane#0a=4' ; echo # Użyto znaku przestankowego '#'
curl 'http://localhost:8000/submit?name=Jane%0a=4' ; echo # Użyto znaku przestankowego '%'
curl 'http://localhost:8000/submit?name=Janeą0a=4' ; echo # Użyto polskiej litery; system kodowania Unicode
```

8. Wpisz w przeglądarce adres http://localhost:8000/

Tabela widoczna na stronie *http://localhost:8000/* składa się z dwóch sekcji:

1. Without AJAX and Fetch API
2. Asynchronous requests

W każdej z nich znajdują się dwie grupy przycisków:

1. Przyciski wykonujące zapytanie do serwera za pomocą metody *GET*
2. Przyciski wykonujące zapytanie do serwera za pomocą metody *POST*

Etykiety przycisków określają wartość nagłówka *Accept* dla żądania HTTP, a więc typ odpowiedzi generowanej przez serwer:

text
Odpowiedź generowana przez serwer ma być tekstem

json
Odpowiedź generowana przez serwer ma być dokumentem JSON

xml
Odpowiedź generowana przez serwer ma być dokumentem XML

9. Wpisz swoje imię w polu formularza, naciśnij dowolny z przycisków sekcji "Without AJAX and Fetch API" i zaobserwuj, czy wynik przetwarzania pojawia się na stronie formularza, czy na osobnej stronie WWW?
10. Zatwierdź dane dowolnym z przycisków sekcji "Asynchronous requests" i zaobserwuj, czy miejsce pojawiania się wyniku uległo zmianie?
11. Zobacz, co wyświetlają:
    - Konsola przeglądarki WWW — sekwencja `Ctrl+Shit+I` otwiera konsolę
    - Konsola serwera

Dla ciekawskich

- W przypadku formularza HTML i wysyłania danych poprzez naciśnięcie przycisku typu *submit* nie mamy możliwości określenia formatu odpowiedzi serwera
- Wpisanie do przeglądarki WWW adresu *http://localhost:8000/submit?name=Jane* wywołuje metodę 'GET' skryptu serwerowego, a format generowanej odpowiedzi to, w tym konkretnym przypadku, zwykły tekst
- Odpowiednikiem wpisania powyższego adresu jest wykonanie komendy `curl http://localhost:8000/submit?name=Jane` — domyślną wartością opcji `--request` jest 'GET'
- Za pomocą formularza HTML można wysyłać, tylko, dwa podstawowe typy żądań: 'GET' oraz 'POST' — **pozostałe typy** ('DELETE', 'PUT', ...) są traktowane jako żądania 'GET'
- Jeżeli jednak chcesz użyć jednego z pozostałych typów, to musisz skorzystać z komendy *curl* — patrz **przykład użycia**, albo wysłać żądanie za pomocą AJAX lub Fetch API

12. W pliku *index.pug* uzupełnij treść funkcji `getName()` — patrz treść komentarza w ciele funkcji
13. Korzystając z przykładów omówionych na wykładzie — AJAX oraz Fetch API, zmodyfikuj (plik *index.pug*) treść funkcji `requestAJAX(HTTP_method, response_type, name, ...)` oraz `requestFetchAPI(http_method, response_type, name, ...)`:
    1. Skoryguj wysyłanie danych (GET oraz POST) — zmodyfikuj treść linii opatrzonych komentarzem "`TO BE MODIFIED`" oraz dodaj właściwy kod w liniach opisanych komentarzem "`TO BE ADDED`"
    2. Proszę spowodować, aby **okno alertu**, w zależności od typu odpowiedzi (argument *response_type*), zamiast domyślnej reprezentacji tekstowej obiektu (`[object Object]`, `[object XMLDocument]` lub `[object Promise]`), wyświetlało otrzymany komunikat w czytelnej postaci:

Czytelna postać komunikatu	**Wartość argumentu *response_type***
Hello 'dane z pola tekstowego formularza'	text
{"welcome":"Hello 'dane z pola tekstowego formularza'"}	json
<welcome>Hello 'dane z pola tekstowego formularza'</welcome>	• document • xml

    3. Przeczytaj fragment artykułu nt. funkcji `encodeURI*()`, a następnie zastosują ją w swoim skrypcie — wynik testu jednostkowego ma być pozytywny

🏠 2. Kwestie bezpieczeństwa — mechanizmy 'SOP' oraz 'CORS'

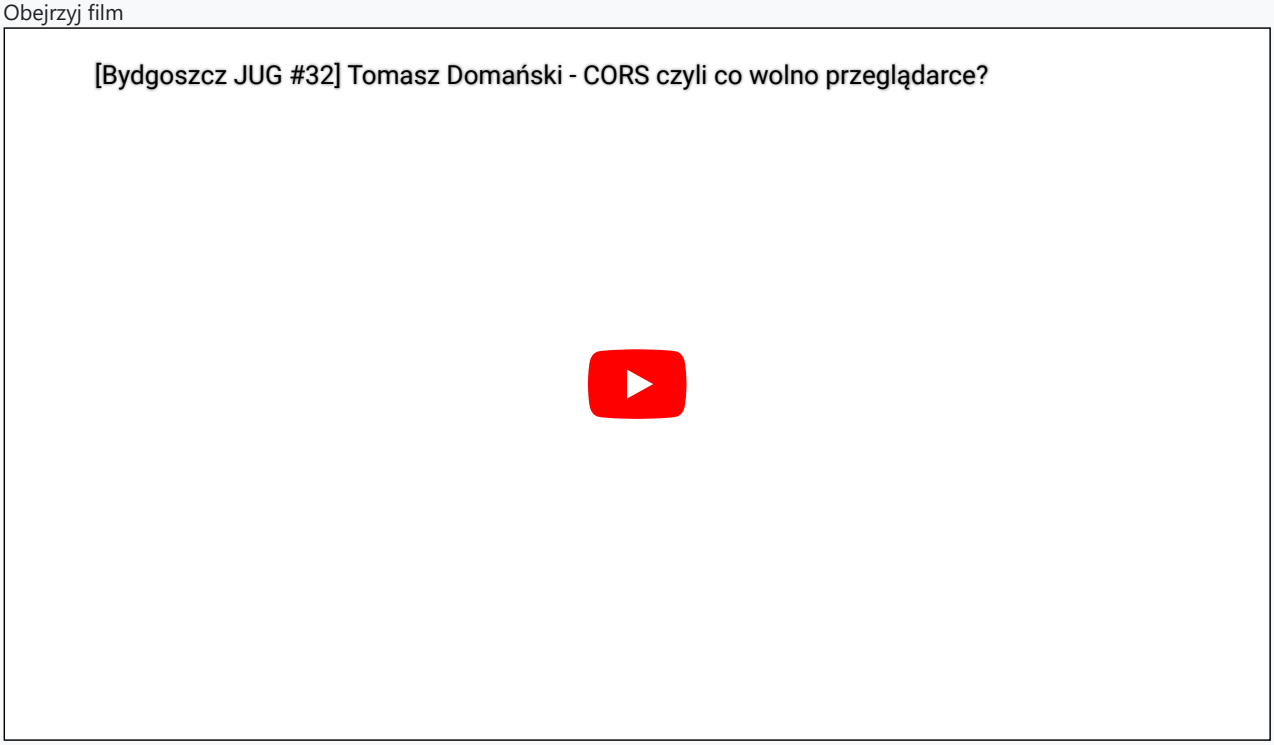
1. Utwórz skrypt *app2.js* o poniższej zawartości:

```
/**
 * @author Stanisław Polak <polak@agh.edu.pl>
 */

import express from 'express';
import morgan from 'morgan';
import { encodeXML } from 'entities';

const app1 = express();
const app2 = express();

app1.set('view engine', 'pug');
app1.locals.pretty = app1.get('env') === 'development';
/* ********************************************** */
app2.use(morgan('dev'));
app2.use(express.urlencoded({ extended: false }));
/* ********************************************** */
app1.get('/', function (request, response) {
    response.render('index');
});

app2.all('/submit', function (req, res) {
    // Return the greeting in the format preferred by the WWW client
    let name = ['GET','DELETE'].includes(req.method) ? req.query.name: req.body.name;

    switch (req.accepts(['html', 'text', 'json', 'xml'])) {
        case 'json':
            // Send the JSON greeting
            res.type('application/json');
            res.json({ welcome: `Hello '${name}'` });
            console.log(`\x1B[32mThe server sent a JSON document to the browser using the '${req.method}' method\x1B[0m`);
            break;

        case 'xml':
            // Send the XML greeting
            name = name !== undefined ? encodeXML(name) : '';
            res.type('application/xml');
            res.send(`<welcome>Hello '${name}'</welcome>`);
            console.log(`\x1B[32mThe server sent an XML document to the browser using the '${req.method}' method\x1B[0m`);
            break;

        default:
            // Send the text plain greeting
            res.type('text/plain');
            res.send(`Hello '${name}'`);
            console.log(`\x1B[32mThe server sent a plain text to the browser using the '${req.method}' method\x1B[0m`);
    }
});
/* ********************************************** */
app2.listen(8000, function () {
    console.log('The server was started on port 8000');
    app1.listen(8001, function () {
        console.log('The server was started on port 8001');
        console.log('To stop the servers, press "CTRL + C"');
    });
});
```

Kod zawiera zmodyfikowaną wersję poprzedniej aplikacji — strona z formularzem HTML oraz strona wyników są dostępne pod dwoma różnymi adresami, odpowiednio, `http://localhost:8001/` oraz `http://localhost:8000/submit`

2. Uruchom aplikację — `node --watch app2` (v18.11.0+) lub `npx nodemon app2`

3. Wpisz w przeglądarce adres http://localhost:8001/ i sprawdź, czy operacje asynchroniczne wykonują się poprawnie — naciśnij dowolny z przycisków sekcji "Asynchronous requests" i zobacz, co wyświetlają: okno alertu oraz konsola przeglądarki
4. Przeczytaj następujące artykuły:
   - Same-Origin Policy (SOP) a bezpieczeństwo www
   - Cross-Origin Resource Sharing (CORS) a bezpieczeństwo www
5. Obejrzyj film



6. Korzystając z informacji zawartych w tych artykułach oraz filmie, popraw zawartość pliku *app2.js* tak, aby asynchroniczne zapytania działały poprawnie
7. Dodaj, w szablonie, element *div*
8. Zmień sposób wyświetlania odebranych danych — zamiast w oknie alertu dane mają być wyświetlane w treści strony WWW — **w elemencie *div***

🏠 3. Obietnice

1. Przeczytaj rozdziały Funkcje zwrotne, Obietnice - Promise oraz Async / await kursu języka JavaScript
2. Zmodyfikuj treść funkcji `requestFetchAPI()` — zamiast metod `then()` oraz `catch()` ma używać deklaracji `await`, a obsługa błędów ma się odbywać w oparciu o blok `try` / `catch`
3. Korzystając z przykładu omówionego na wykładzie, zdefiniuj funkcję `getTime(europe_city)` tworzącą, a następnie zwracającą obiekt *Promise*:
   - Wykonawca obietnicy łączy się (AJAX lub Fetch API) z usługą sieciową World Time API i pobiera aktualny czas dla podanego miasta europejskiego
   - Jeżeli status odpowiedzi to 200, obietnica jest uważana za spełnioną — należy zwrócić odebrany (z serwera czasu) dokument JSON
   - Jeżeli status odpowiedzi to 404, obietnica jest niespełniona — należy zwrócić, otrzymaną z serwera czasu, treść komunikatu o błędzie

   > - Funkcja `getTime(europe_city)` **ma tworzyć odrębny obiekt *Promise*, a następnie go zwracać**; a nie, bezpośrednio zwracać obiekt *Promise* będący wynikiem zapytania asynchronicznego do usługi sieciowej "World Time API"
   > - Serwis "World Time API" obsługuje CORS — możesz wyświetlić nagłówki odpowiedzi za pomocą komendy `curl --head "http://worldtimeapi.org/api/timezone/Europe"`

4. Zmodyfikuj formularz HTML w szablonie:
   - Ma zawierać pole tekstowe 'city' oraz przycisk typu *button*
   - Naciśnięcie przycisku ma powodować wywołanie funkcji `getTime(europe_city)`, gdzie *europe_city* to nazwa miasta znajdująca się w polu tekstowym 'city'
   - Wynik wywołania funkcji (aktualny czas lub komunikat o błędzie) mają być widoczne w ww. elemencie *div* — patrz zadanie 2

🧑‍🏫 4. Zadanie

- Zmodyfikuj aplikację z poprzednich ćwiczeń — szczegóły zostaną określone **na początku zajęć**
- Założenia — aplikacja ma pobierać dane z wykorzystaniem komunikacji asynchronicznej

Edytuj zadanie | Usuń zadanie

Status przesłanego zadania

Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Nieocenione
Ostatnio modyfikowane	środa, 5 czerwca 2024, 19:34
Przesyłane pliki	lab6.zip 5 czerwca 2024, 19:34
Komentarz do przesłanego zadania	Komentarze (0)

Informacja zwrotna

Ocena	4,50 / 5,00
Ocenione dnia	środa, 22 maja 2024, 15:15