

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 poniższym filmie.

Javascript Infinite Scroll Data Fetching Tutorial to Make HTTP Request to JSONPlaceholder API



1. Zapytania AJAX oraz FetchAPI

1. Wykonaj komendy:

```
1. mkdir -p cw6/views
2. cd cw6
3. npm init es6
4. npm install express pug morgan entities
```

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

```

1.  /**
2.   * @author Stanisław Polak <polak@agh.edu.pl>
3.   */
4.
5.  import express from 'express';
6.  import morgan from 'morgan';
7.  import { encodeXML } from 'entities';
8.
9.  const app = express();
10.
11.  app.set('view engine', 'pug');
12.  app.locals.pretty = app.get('env') === 'development';
13.  /* ***** */
14.  morgan.token('accept', function (req, res) { return req.headers['accept'] });
15.  app.use(morgan(':method :url :status\tAccept: :accept'));
16.  app.use(express.urlencoded({ extended: false }));
17.  /* ***** */
18.  app.get('/', function (request, response) {
19.      response.render('index');
20.  });
21.
22.  app.all('/submit', function (req, res) {
23.      let name = ['GET', 'DELETE'].includes(req.method) ? req.query.name: req.body.name;
24.
25.      console.log();
26.      console.log('-----');
27.      console.count('Request');
28.      console.log('-----');
29.      console.group('\x1B[35mreq.query\x1B[0m');
30.      console.table(req.query);
31.      console.groupEnd();
32.      console.group('\x1B[35mreq.body\x1B[0m');
33.      console.table(req.body);
34.      console.groupEnd();
35.      console.group('\x1B[35mname\x1B[0m');
36.      console.log(name);
37.      console.groupEnd();
38.      console.log();
39.
40.      // Return the greeting in the format preferred by the WWW client
41.      switch (req.accepts(['html', 'text', 'json', 'xml'])) {
42.          case 'json':
43.              // Send the JSON greeting
44.              res.type('application/json');
45.              res.json({ welcome: `Hello '${name}'` });
46.              console.log(`The server sent a \x1B[31mJSON\x1B[0m document to the browser
for the request below`);
47.              break;
48.
49.          case 'xml':
50.              // Send the XML greeting
51.              name = name !== undefined ? encodeXML(name) : '';
52.              res.type('application/xml');
53.              res.send(`<welcome>Hello '${name}'</welcome>`);
54.              console.log(`The server sent an \x1B[31mXML\x1B[0m document to the browser
for the request below`);
55.              break;
56.
57.          default:
58.              // Send the text plain greeting
59.              res.type('text/plain');
60.              res.send(`Hello '${name}'`);
61.              console.log(`The server sent a \x1B[31mplain text\x1B[0m document to the
browser for the request below`);

```

```
62.     }
63.   });
64.   /* ***** */
65.   app.listen(8000, function () {
66.     console.log('The server was started on port 8000');
67.     console.log('To stop the server, press "CTRL + C"');
68.   });
```

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.  /* *****
25.  /* Function that retrieves the content of one of the selected text fields of
26.  */
27.
28.  /* *****
29.  function getName(http_method){
30.      let name = '';
31.
32.      // TODO: Here put the code that, depending on the value of the 'http_method'
33.      variable – GET / POST – assigns the 'name' variable to the value of the 'name_GET' / 'name_POST'
34.      form field
35.
36.      return name;
37.  }
38.
39.  /* *****
40.  /* Function that performs (asynchronous) query to the web server using AJAX
41.  */
42.
43.  /* *****
44.  /* http_method ∈ ["GET", "POST"]
45.  */
46.
47.  /* response_type ∈ ["text", "json", "document"]
48.  */
49.
50.  /* name – Contents of the form's text box – data that needs to be sent asynchronously
51.  */
52.
53.  /* *****
54.  function requestAJAX(http_method, response_type, name, show_alert=false) {
55.      //-----
56.      // Create an object representing the request to the web server – see
57.      https://developer.mozilla.org/docs/Web/API/XMLHttpRequest
58.      //-----
59.      const xhr = new XMLHttpRequest();
60.
61.      //-----
62.      // Observers registration
63.      //-----
64.
65.      // If the request was successful

```

```

52.         xhr.addEventListener("load", function (evt) {
53.             if (xhr.status === 200) {
54.                 console.group('AJAX');
55.                 console.log(`HTTP method →\t\t${http_method}\nResponse type
→\t\t${response_type}\nInput data →\t\t${name}`);
56.                 console.log(xhr.response);
57.                 console.groupEnd();
58.                 if(show_alert)
59.                     window.alert(xhr.response);
60.                 else {
61.                     results.set(`ajax ${http_method} ${response_type}`, xhr.respo
62.                     dispatchEvent(received);
63.                 }
64.             }
65.         });
66.
67.         // If the request was failed
68.         xhr.addEventListener("error", function (evt) {
69.             window.alert('There was a problem with this request.');
```

```

110.     }
111.
112.
113.     /* Function that performs (asynchronous) query to the web server usingFetch
114.     */
115.
116.     /* http_method ∈ ["GET", "POST"]
117.     */
118.     /* response_type ∈ ["text", "json", "xml"]
119.     */
120.     /* name - Contents of the form's text box - data that needs to be sent asynch
121.     */
122.
123.     /* *****
124.     function requestFetchAPI(http_method, response_type, name, show_alert=false)
125.     let accept = '*/';
126.
127.     switch(response_type){
128.     case 'json':
129.         accept = 'application/json';
130.         break;
131.     case 'xml':
132.         accept = 'application/xml';
133.         break;
134.     }
135.     //-----
136.     // Configuration and execution of the (asynchronous) query to the web ser
137.     //-----
138.     ((http_method) => {
139.         if(http_method === 'GET')
140.             return fetch('http://localhost:8000/submit', { // TO BE MODIF
141.                 method: 'GET',
142.                 credentials: "include", // Do not modify or remove
143.                 headers: {
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
1000.

```

```

159.
160.         })(http_method) // a promise is returned
161.         .then(function (response) { // if the promise is fulfilled
162.             if (!response.ok)
163.                 throw Error(response.statusText);
164.
165.             console.group('Fetch API');
166.             console.log(`HTTP method →\t\t${http_method}\nResponse type
→\t\t${response_type}\nInput data →\t\t${name}`);
167.             let result;
168.
169.             if (!response.headers.get('content-type')?.includes('application/json'))
170.                 // If the received data is plain text or an XML document
171.                 result = response.text();
172.             }
173.             else {
174.                 //If the received data is a JSON document
175.                 result = response.json();
176.             }
177.             console.log(result);
178.             console.groupEnd();
179.             if(show_alert)
180.                 window.alert(result);
181.             else {
182.                 results.set(`fetch ${http_method} ${response_type}`, result);
183.                 dispatchEvent(received);
184.             }
185.         })
186.         .catch(function (error) { // if the promise is rejected
187.             window.alert(error);
188.         });
189.     }
190.     script(src="https://cdn.jsdelivr.net/npm/mocha/mocha.js")
191.     script(type="module").
192.         import { expect } from 'https://cdn.jsdelivr.net/npm/chai/chai.js'
193.         window.expect = expect
194.     body
195.         script(class="mocha-init").
196.             mocha.setup('bdd');
197.             mocha.checkLeaks();
198.     main
199.         table
200.             tr
201.                 th
202.                 th GET
203.                 th POST
204.             tr
205.                 th(colspan='3' style=' background-color: #04556D;') Without AJAX and
206.             tr
207.                 th HTTP
208.                 td
209.                     form(action="http://localhost:8000/submit" method="GET")
210.                         label(for="name_GET") Your name
211.                         input(type="text" id="name_GET" name="name")
212.                         br
213.                         input(type="submit" value="text")
214.                 td
215.                     form(action="http://localhost:8000/submit" method="POST")
216.                         label(for="name_POST") Your name
217.                         input(type="text" id="name_POST" name="name")
218.                         br
219.                         input(type="submit" value="text")
220.             tr
221.                 th(colspan='3' style=' background-color: #04556D;') Asynchronous req

```

```

222.         tr
223.             th AJAX
224.             each method in ["GET", "POST"]
225.                 td
226.                     each type in ["text", "json", "document"]
227.
228.                         button(onclick=`console.clear() ;
requestAJAX("${method}", "${type}", getName('${method}'), true)` `) #{type}
229.         tr
230.             th Fetch API
231.             each method in ["GET", "POST"]
232.                 td
233.                     each type in ["text", "json", "xml"]
234.                         button(onclick=`console.clear() ;
requestFetchAPI("${method}", "${type}", getName('${method}'), true)` `) #{type}
235.     h1 Unit tests
236.     button(onclick='window.location.reload();') Restart
237.     div(id="mocha")
238.     script.
239.         const name = 'John Doe a/?:@&=+$#';
240.
241.         if(window.location.port == 8000) {
242.             window.addEventListener("load", (event) => {
243.                 for(let method of ["GET", "POST"]){
244.                     for(let type of ["text", "json", "document"]){
245.                         requestAJAX(method, type, name);
246.                     for(let type of ["text", "json", "xml"]){
247.                         requestFetchAPI(method, type, name);
248.                     }
249.                 })
250.             };
251.             script(class="mocha-exec").
252.
253. //*****
254.         // Unit tests
255.
256. //*****
257.         var results = new Map();
258.         var received = new Event('received');
259.         var test_executed = false;
260.
261.         function decodeHtml(html) {
262.             var txt = document.createElement("textarea");
263.             txt.innerHTML = html;
264.
265.             return txt.value;
266.         }
267.
268.         addEventListener('received', (e) => {
269.             if(!test_executed && results.size === 12){
270.                 const parser = new DOMParser();
271.                 const xml_document= parser.parseFromString("<welcome>Hello 'John Doe
a/?:@&=+$#</welcome>", "text/xml");
272.
273.                 describe('AJAX requests', function() {
274.                     it(`Returns "Hello '${name}'" for requestAJAX('GET','text')`, function() {
275.                         expect(results.get('ajax GET text')).toEqual('Hello '${name}')
276.                     });
277.                     it(`Returns "Hello '${name}'" for requestAJAX('GET','json')`, function() {
278.                         expect(results.get('ajax GET json')).toEqual({welcome: `Hello
'${name}'`});
279.                     });
280.                     it(`Returns "Hello '${name}'" for requestAJAX('GET','document')`, function() {
281.                         expect(results.get('ajax GET document')).toEqual({welcome: `Hello
'${name}'`});
282.                     });
283.                 });
284.             }
285.         });
286.     }
287.
288.     function() {

```



```

279.         expect(results.get('ajax GET
document').documentElement.firstChild.data).to.equal(xml_document.documentElement.firstChild
280.     });
281.     it('Returns "Hello `${name}`" for requestAJAX('POST','text')', function() {
282.         expect(results.get('ajax POST text')).to.equal(`Hello `${name}``);
283.     });
284.     it('Returns "Hello `${name}`" for requestAJAX('POST','json')', function() {
285.         expect(results.get('ajax POST json')).to.eql({welcome: `Hello
`${name}``});
286.     });
287.     it('Returns "Hello `${name}`" for requestAJAX('POST','document')', function() {
288.         expect(results.get('ajax POST
document').documentElement.firstChild.data).to.equal(xml_document.documentElement.firstChild
289.     });
290.     });
291.
292.     describe('Fetch API requests', function() {
293.         it('Returns "Hello `${name}`" for requestFetchAPI('GET','text')', function() {
294.             const result = await results.get('fetch GET text')
295.             expect(result).to.equal(`Hello `${name}``);
296.         });
297.         it('Returns "Hello `${name}`" for requestFetchAPI('GET','json')', function() {
298.             const result = await results.get('fetch GET json')
299.             expect(result).to.eql({welcome: `Hello `${name}``});
300.         });
301.         it('Returns "Hello `${name}`" for requestFetchAPI('GET','xml')', function() {
302.             const result = await results.get('fetch GET xml');
303.             expect(decodeHtml(result)).to.equal(`<welcome>Hello
`${name}`</welcome>`);
304.         });
305.         it('Returns "Hello `${name}`" for requestFetchAPI('POST','text')', function() {
306.             const result = await results.get('fetch POST text')
307.             expect(result).to.equal(`Hello `${name}``);
308.         });
309.         it('Returns "Hello `${name}`" for requestFetchAPI('POST','json')', function() {
310.             const result = await results.get('fetch POST json')
311.             expect(result).to.eql({welcome: `Hello `${name}``});
312.         });
313.         it('Returns "Hello `${name}`" for requestFetchAPI('POST','xml')', function() {
314.             const result = await results.get('fetch POST xml');
315.             expect(decodeHtml(result)).to.equal(`<welcome>Hello
`${name}`</welcome>`);
316.         });
317.     });
318.
319.     mocha.run();
320.     test_executed = true;
321. }
322. });

```

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

5. Wykonaj, pojedynczo, poniższe komendy [curl](#)

```
1. # Use of basic HTTP methods
2. curl --include http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
3. curl --include --header "Accept: application/json" http://localhost:8000/submit?
   name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
4. curl --include --header "Accept: application/xml" http://localhost:8000/submit?
   name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
5. curl --include --request POST --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa'
   http://localhost:8000/submit ; echo
6. curl --include --request POST --header "Accept: application/json" --data
   'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
7. curl --include --request POST --header "Accept: application/xml" --data
   'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
8. # Use of other HTTP methods
9. curl --include --request DELETE http://localhost:8000/submit?
   name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
10. curl --include --request DELETE --header "Accept: application/json"
    http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
11. curl --include --request DELETE --header "Accept: application/xml"
    http://localhost:8000/submit?name=Jane%20Doe%20R%C3%B3%C5%BCa ; echo
12. curl --include --request PUT --data 'name=Jane%20Doe%20R%C3%B3%C5%BCa'
    http://localhost:8000/submit ; echo
13. curl --include --request PUT --header "Accept: application/json" --data
    'name=Jane%20Doe%20R%C3%B3%C5%BCa' http://localhost:8000/submit ; echo
14. 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](#):

```
1. curl 'http://localhost:8000/submit?name=Jane&0a=4' ; echo # Użyto znaku przestankowego
   '&'
2. curl 'http://localhost:8000/submit?name=Jane+0a=4' ; echo # Użyto znaku przestankowego
   '+'
3. curl 'http://localhost:8000/submit?name=Jane#0a=4' ; echo # Użyto znaku przestankowego
   '#'
4. curl 'http://localhost:8000/submit?name=Jane%0a=4' ; echo # Użyto znaku przestankowego
   '%'
5. 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+Shift+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 `-r 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 opatrzonej 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 XMLHttpRequest]` lub `[object Promise]`), wyświetlało otrzymany komunikat w czytelnej postaci:
- | Czytelna postać komunikatu | Wartość argumentu <code>response_type</code> |
|--|--|
| Hello 'dane z pola tekstowego formularza' | text |
| {"welcome":"Hello 'dane z pola tekstowego formularza'"} | json |
| <welcome>Hello 'dane z pola tekstowego formularza'</welcome> | <ul style="list-style-type: none">▪ document▪ xml |
3. Przeczytaj [fragment artykułu](#) nt. funkcji `encodeURIComponent()`, a następnie zastosuj 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:

```

1.  /**
2.   * @author Stanisław Polak <polak@agh.edu.pl>
3.   */
4.
5.  import express from 'express';
6.  import morgan from 'morgan';
7.  import { encodeXML } from 'entities';
8.
9.  const app1 = express();
10. const app2 = express();
11.
12. app1.set('view engine', 'pug');
13. app1.locals.pretty = app1.get('env') === 'development';
14. /* ***** */
15. app2.use(morgan('dev'));
16. app2.use(express.urlencoded({ extended: false }));
17. /* ***** */
18. app1.get('/', function (request, response) {
19.     response.render('index');
20. });
21.
22. app2.all('/submit', function (req, res) {
23.     // Return the greeting in the format preferred by the WWW client
24.     let name = ['GET', 'DELETE'].includes(req.method) ? req.query.name: req.body.name;
25.
26.     switch (req.accepts(['html', 'text', 'json', 'xml'])) {
27.         case 'json':
28.             // Send the JSON greeting
29.             res.type('application/json');
30.             res.json({ welcome: `Hello '${name}'` });
31.             console.log(`\x1B[32mThe server sent a JSON document to the browser using
the '${req.method}' method\x1B[0m`);
32.             break;
33.
34.         case 'xml':
35.             // Send the XML greeting
36.             name = name !== undefined ? encodeXML(name) : '';
37.             res.type('application/xml');
38.             res.send(`<welcome>Hello '${name}'</welcome>`);
39.             console.log(`\x1B[32mThe server sent an XML document to the browser using
the '${req.method}' method\x1B[0m`);
40.             break;
41.
42.         default:
43.             // Send the text plain greeting
44.             res.type('text/plain');
45.             res.send(`Hello '${name}'`);
46.             console.log(`\x1B[32mThe server sent a plain text to the browser using the
'${req.method}' method\x1B[0m`);
47.     }
48. });
49. /* ***** */
50. app2.listen(8000, function () {
51.     console.log('The server was started on port 8000');
52.     app1.listen(8001, function () {
53.         console.log('The server was started on port 8001');
54.         console.log('To stop the servers, press "CTRL + C"');
55.     });
56. });

```

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>

- Uruchom aplikację — `node --watch app2 (v18.11.0+) lub npx nodemon app2`
- 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
- Przeczytaj następujące artykuły:
 - [Same-Origin Policy \(SOP\) a bezpieczeństwo www](#)
 - [Cross-Origin Resource Sharing \(CORS\) a bezpieczeństwo www](#)
- Obejrzyj film

[Bydgoszcz JUG #32] Tomasz Domański - CORS czyli co wolno przeglądarce?



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



3. Obietnice

- Przeczytaj rozdziały [Funkcje zwrotne](#), [Obietnice - Promise](#) oraz [Async / await](#) kursu języka JavaScript
- 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`
- 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"
```
- 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