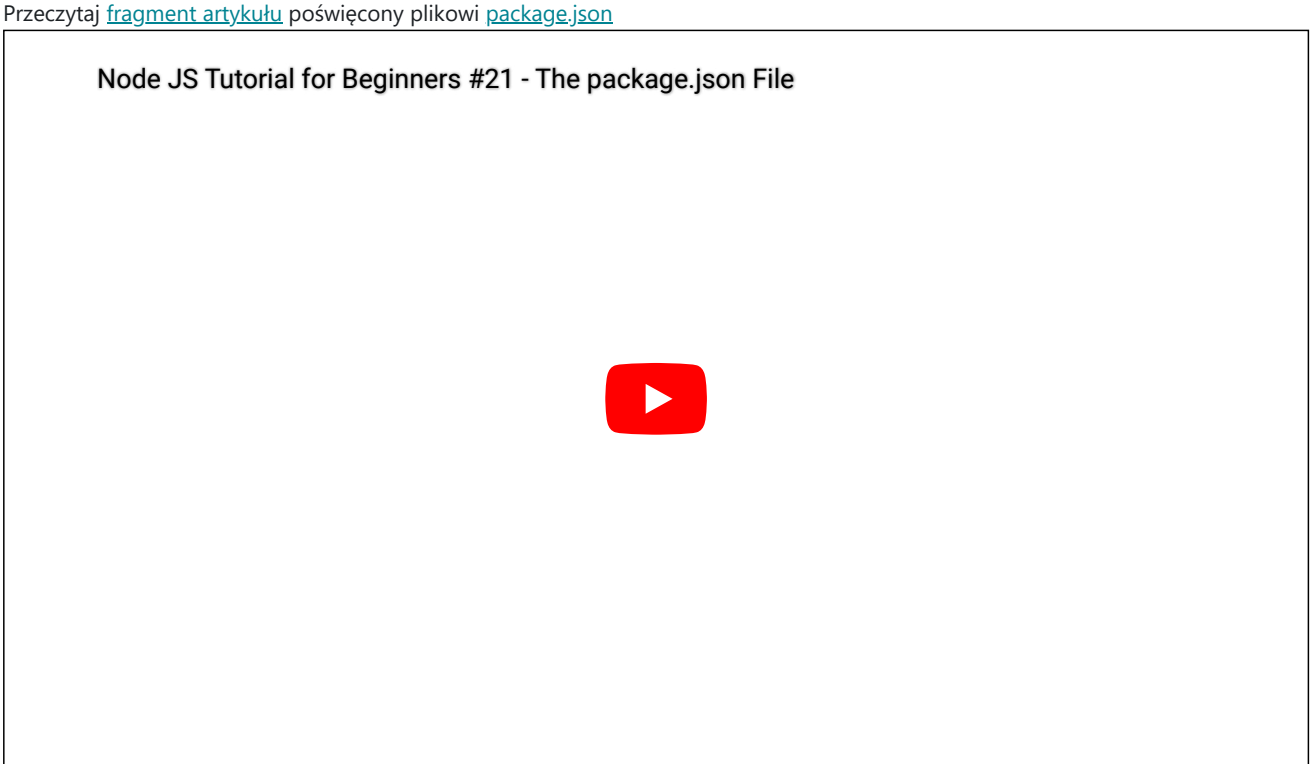
✓ Wykonano

🏠 1. Skrypt konsolowy

1. Wykonaj komendy:

```
mkdir cw4 # Utworzenie katalogu głównego projektu
cd cw4
mkdir src test
npm init --yes # Utworzenie pliku 'package.json'
```

2. Uruchom program "Visual Studio Code" — `code .`
3. Utwórz profil o nazwie *JavaScript* bazujący na szablonie "Node.js" — zostaną zainstalowane podstawowe rozszerzenia VSC dla tego języka — zaznajom się z ich skróconym opisem
4. Przeczytaj artykuł nt. rekomendowanej struktury katalogów dla projektów NodeJS
5. Przeczytaj fragment artykułu poświęcony plikowi package.json



6. Obejrzyj zawartość wygenerowanego pliku *package.json*
7. Utwórz plik *src/console_script1.js* o następującej zawartości:

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

 /* ********* */
 /* CommonsJS */
 /* ********* */
 // const fs = require('fs-extra')
 const fs = require('node:fs')
 const { argv } = require('node:process');

 /* **** */
 /* ES6  */
 /* **** */
 // import fs from 'fs-extra';
 // import fs from 'node:fs';
 // import { argv } from 'node:process';

 /************************ */
 function read_async() {
     console.log(`1.\t\x1B[33mWykonano pierwszą linię funkcji "read_async()"\x1B[0m`);
     console.log('2.\t\x1B[33mWywołano funkcję \'readFile()\'\x1B[0m');
     console.time('\tCzas wykonania "readFile()"');
     fs.readFile(argv[1], (err, data) => {
         if (err) throw err;
         console.log('3.\t\x1B[33mWczytana zawartość pliku jest dostępna w zmiennej \'data\'\x1B[0m');
     });
     console.timeEnd('\tCzas wykonania "readFile()"');
     console.log(`4.\t\x1B[33mWykonano ostatnią linię funkcji "read_async()"\x1B[0m`);
 }

 function read_sync() {
     console.log(`5.\t\x1B[32mWykonano pierwszą linię funkcji "read_sync()"\x1B[0m`);
     console.log('6.\t\x1B[32mWywołano funkcję \'readFileSync()\'\x1B[0m');
     console.time('\tCzas wykonania "readFileSync()"');
     let data = fs.readFileSync(argv[1]);
     console.timeEnd('\tCzas wykonania "readFileSync()"');
     console.log('7.\t\x1B[32mWczytana zawartość pliku jest dostępna w zmiennej \'data\'\x1B[0m');
     console.log(`8.\t\x1B[32mWykonano ostatnią linię funkcji "read_sync()"\x1B[0m`);
 }

 /************************ */

 console.clear()
 console.log(`\x1B[31mAsynchroniczny odczyt pliku "${argv[1]}"\x1B[0m`);
 read_async();
 console.log('------------------');
 console.log(`\x1B[31mSynchroniczny odczyt pliku "${argv[1]}"\x1B[0m`);
 read_sync();
 console.log('------------------');
 console.log('9.\t\x1B[34mWykonano ostatnią linię skryptu\x1B[0m');
```

- Funkcja `read_async()` asynchronicznie odczytuje zawartość pliku; komunikaty diagnostyczne są w kolorze żółtym
- Funkcja `read_sync()` synchronicznie odczytuje zawartość pliku; komunikaty diagnostyczne są w kolorze zielonym
- Wbudowany moduł **fs** nie oferuje funkcji do obsługi plików JSON — taką funkcjonalność oferuje, miedzy innymi, pakiet **fs-extra** — **przykład**.
- Instalacja — `npm install fs-extra`
- W naszym przykładowym kodzie, do wypisywania komunikatów diagnostycznych użyto `console.log()` — lepszym rozwiązaniem jest użycie modułu **debug**

8. Uruchom skrypt — za pomocą sekwencji klawiszy `Ctrl+`` otwórz panel terminala, a następnie wpisz w nim polecenie `node src/console_script1.js` .
   Zaobserwuj, czy wyniki wykonania funkcji `read_async()` oraz `read_sync()` są jednakowe? Jak myślisz, dlaczego komunikat diagnostyczny nr 3

   ```
   3.      Wczytana zawartość pliku jest dostępna w zmiennej 'data'
   ```

   pojawia się na samym końcu?
9. Przeczytaj artykuł poświęcony modułom
10. Zakomentuj linie 9-10 a odkomentuj linie 16-17 skryptu *console_script1.js*
11. Ponownie uruchom skrypt — jak można zauważyć, uruchomienie kończy się niepowodzeniem — rozwiąż problem
12. Napisz skrypt "Licznik uruchomień":
    - Wartość licznika jest przechowywana w pliku
    - Jeżeli skrypt wywołano z opcją `--sync`, to odczyt oraz zapis są wykonywane za pomocą funkcji: fs.readFileSync(...) oraz fs.writeFileSync(...)
    - Jeżeli skrypt wywołano z opcją `--async`, to odczyt oraz zapis są wykonywane za pomocą funkcji korzystających z callbacków: fs.readFile(...,callback) oraz fs.writeFile(...,callback) — **proszę nie korzystać z wersji 'Promise-based'** — mechanizm obietnic będzie używany na szóstych ćwiczeniach

      > Jeżeli jest wymagana **maksymalna wydajność** (zarówno pod względem czasu wykonania, jak i alokacji pamięci), to należy używać wersji 'Callback-based', a nie 'Promise-based'.
      > — Cytat ze strony ***File system*** *(dokumentacja Node.js)*

    - W przypadku gdy skrypt został wywołany bez argumentu, to wykonuje komendy systemowe wprowadzone poprzez standardowe wejście

Przykład

```
$ node src/console_script2.js --async
  Liczba uruchomień: 1
$ node src/console_script2.js --async
  Liczba uruchomień: 2
$ node src/console_script2.js --sync
  Liczba uruchomień: 3
$ node src/console_script2.js
  Wprowadź komendy – naciśnięcie Ctrl+D kończy wprowadzanie danych
ls
  file.js package.json package-lock.json
date
  pią, 31 mar 2023, 12:32:12 CEST
 Ctrl+D
$
```

Dla ambitnych — zadanie nieobowiązkowe

Korzystając z programu **Apache Bench** zbadaj wydajność dwóch aplikacji internetowych. Pierwsza z nich czyta zawartość pliku tekstowego, korzystając z funkcji modułu 'fs' w wersji synchronicznej. Druga, używa wersji asynchronicznej tych funkcji.

- **Pobierz kod źródłowy** ww. aplikacji z **repozytorium**
- Wykonaj komendę `perl -pi -e 's/Buffer/Buffer.alloc/g;' *.js`
- Uruchom wersję synchroniczną — wykonaj komendę `node sync.js`
- Za pomocą polecenia `ab -n 1000 -c 1000 -vhr http://localhost:8081/` zbadaj szybkość działania wersji synchronicznej
- Uruchom wersję asynchroniczną — wykonaj komendę `node async.js`
- Za pomocą polecenia `ab -n 1000 -c 1000 -vhr http://localhost:8080/` zbadaj szybkość działania wersji asynchronicznej
- Porównaj swoje wyniki z wynikami przedstawionymi w artykule "**Async Vs. Sync I/O Benchmark In NodeJs**"

Warto wiedzieć

1. **Why using sync versions of async functions is bad**
2. Obsługa plików, a wątki:

> Oto interfejsy API modułu Node.js korzystające z puli wątków roboczych:
>
> - ...
> - System plików: Wszystkie interfejsy API systemu plików z wyjątkiem `fs.FSWatcher()` i tych, które są jawnie synchroniczne, korzystają z puli wątków *libuv*.
>
> ...
>
> Kilka podstawowych modułów Node.js ma synchronicznie drogie interfejsy API, w tym:
>
> - ...
> - System plików
>
> Te interfejsy API są drogie, ponieważ wymagają znacznych obliczeń (szyfrowanie, kompresja), wymagają operacji we/wy (we/wy pliku) lub potencjalnie obu (proces potomny). Te interfejsy API są przeznaczone dla wygody tworzenia skryptów, ale nie są przeznaczone do użytku w kontekście serwera. Jeśli wykonasz je w pętli zdarzeń, ich wykonanie zajmie znacznie więcej czasu niż typowa instrukcja JavaScript, blokując pętlę zdarzeń.
>
> — Cytat ze strony ***Don't Block the Event Loop (or the Worker Pool)***

Dla ciekawskich

- Ryan Dahl — twórca Node.js — stworzył alternatywny projekt o nazwie **Deno**
- Porównanie obydwu projektów: **1**, **2**
- Na ostatnich ćwiczeniach będziemy korzystać z Deno
- Poniżej znajduje się wersja dla Deno skryptu *src/console_script1.js* — uruchamianie komendą `deno run --allow-read=. nazwaPliku.js nazwaPliku.js`

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

function read_sync() {
    console.log(`1.\t\x1B[32mWykonano pierwszą linię funkcji "read_sync()"\x1B[0m`);
    console.log('2.\t\x1B[33mWywołano funkcję \'readTextFileSync()\'\x1B[0m');
    const data = Deno.readTextFileSync(Deno.args[0]);
    console.log('3.\t\x1B[33mWczytano zawartość pliku – jest ona dostępna w zmiennej \'data\'\x1B[0m');
    console.log(`4.\t\x1B[32mWykonano ostatnią linię funkcji "read_sync()"\x1B[0m`);
}

function read_async() {
    console.log(`1.\t\x1B[32mWykonano pierwszą linię funkcji "read_async()"\x1B[0m`);
    console.log('2.\t\x1B[33mWywołano funkcję \'readTextFile()\'\x1B[0m');
    Deno.readTextFile(Deno.args[0]).then((data) => {
        console.log('3.\t\x1B[33mWczytano zawartość pliku – jest ona dostępna w zmiennej \'data\'\x1B[0m');
    })
    console.log(`4.\t\x1B[32mWykonano ostatnią linię funkcji "read_async()"\x1B[0m`);
}
/************************* */

if (Deno.args.length === 1) {
    console.clear()
    console.log(`\x1B[31mSynchroniczny odczyt pliku "${Deno.args[0]}"\x1B[0m`);
    read_sync();
    console.log('--------------------');
    console.log(`\x1B[31mAsynchroniczny odczyt pliku "${Deno.args[0]}"\x1B[0m`);
    read_async();
    console.log('\t\x1B[34mWykonano ostatnią linię skryptu\x1B[0m');
}
else {
    console.log('Podaj nazwę pliku');
}
```

🏠 2. Skrypt serwerowy

1. Przeczytaj artykuł "Anatomy of an HTTP Transaction"
2. Utwórz plik *src/server_script1.js* o poniższej zawartości:

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

// const http = require('node:http');
// const { URL } = require('node:url');
import http from 'node:http';
import { URL } from 'node:url';


/**
      * Handles incoming requests.
      *
      * @param {IncomingMessage} request - Input stream – contains data received from the browser, e.g,. encoded contents of HTML form fields.
      * @param {ServerResponse} response - Output stream – put in it data that you want to send back to the browser.
      * The answer sent by this stream must consist of two parts: the header and the body.
      * <ul>
      *  <li>The header contains, among others, information about the type (MIME) of data contained in the body.
      *  <li>The body contains the correct data, e.g. a form definition.
      * </ul>
      * @author Stanisław Polak <polak@agh.edu.pl>
 */

function requestListener(request, response) {
    console.log('---------------------------------------');
    console.log(`The relative URL of the current request: ${request.url}`);
    console.log(`Access method: ${request.method}`);
    console.log('---------------------------------------');
    // Create the URL object
    const url = new URL(request.url, `http://${request.headers.host}`);
    /* ************************************************ */
    // if (!request.headers['user-agent'])
    if (url.pathname !== '/favicon.ico')
        // View detailed URL information
        console.log(url);

    /* *************** */
    /* "Routes" / APIs */
    /* *************** */

    switch ([request.method, url.pathname].join(' ')) {
        /*
          ----------------------------------------------------
          Generating the form if
             the GET method was used to send data to the server
          and
             the relative URL is '/',
          ----------------------------------------------------
        */
        case 'GET /':
            /* ************************************************ */
            // Creating an answer header – we inform the browser that the returned data is HTML
            response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
            /* ************************************************ */
            // Setting a response body
            response.write(`
<!DOCTYPE html>
```

```
58.  <html lang="en">
59.    <head>
60.      <meta charset="utf-8">
61.      <meta name="viewport" content="width=device-width, initial-scale=1">
62.      <title>Vanilla Node.js application</title>
63.    </head>
64.    <body>
65.      <main>
66.        <h1>Vanilla Node.js application</h1>
67.        <form method="GET" action="/submit">
68.          <label for="name">Give your name</label>
69.          <input name="name">
70.          <br>
71.          <input type="submit">
72.          <input type="reset">
73.        </form>
74.      </main>
75.    </body>
76.  </html>`);
77.              /* ************************************************ */
78.              response.end(); // The end of the response - send it to the browser
79.              break;
80.
81.          /*
82.            ----------------------------------------------------
83.            Processing the form content, if
84.                the GET method was used to send data to the server
85.            and
86.                the relative URL is '/submit',
87.            ----------------------------------------------------
88.          */
89.          case 'GET /submit':
90.              /* ************************************************ */
91.              // Creating an answer header - we inform the browser that the returned data
    is plain text
92.              response.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
93.              /* ************************************************ */
94.              // Place given data (here: 'Hello <name>') in the body of the answer
95.              response.write(`Hello ${url.searchParams.get('name')}`); //
    "url.searchParams.get('name')" contains the contents of the field (form) named 'name'
96.              /* ************************************************ */
97.              response.end(); // The end of the response - send it to the browser
98.              break;
99.
100.         /*
101.           ----------------------
102.           If no route is matched
103.           ----------------------
104.         */
105.         default:
106.             response.writeHead(501, { 'Content-Type': 'text/plain; charset=utf-8' });
107.             response.write('Error 501: Not implemented');
108.             response.end();
109.     }
110.  }
111.
112.  /* ************************************************ */
113.  /* Main block
114.  /* ************************************************ */
115.  const server = http.createServer(requestListener); // The 'requestListener' function is
    defined above
116.  server.listen(8000);
```

```
117. console.log('The server was started on port 8000');
118. console.log('To stop the server, press "CTRL + C"');
```

3. Uruchom aplikację za pomocą komendy `node src/server_script1`

   Jeżeli chcesz, aby po zmodyfikowaniu zawartości pliku źródłowego, aplikacja samodzielnie się restartowała, to uruchom ją następująco: `node --watch src/server_script1` (v18.11.0+) lub `npx nodemon src/server_script1`

4. Wpisz w przeglądarce adres http://localhost:8000/, podaj, w formularzu HTML, swoje imię i obejrzyj wyniki generowane przez skrypt
5. Sprawdź, jak działa skrypt dla poniższych adresów — co wyświetla przeglądarka i jak wygląda zawartość obiektu URL w konsoli serwera?
   - http://localhost:8000/submit?name=Jan
   - http://localhost:8000/submit?name=Stanis%C5%82aw+Polak%28AGH%29
   - http://localhost:8000/submit?nazwisko=Kowalski
   - http://localhost:8000/submit?name=Jan&nazwisko=Kowalski

   Zamiast adresu symbolicznego `localhost`, np. **http://localhost:8000/**, możesz użyć adresu **<nazwa>.lvh.me** (przykład **http://www.lvh.me:8000/**)

6. Zbadaj działanie skryptu dla poniższych adresów — sprawdź, czy działa on jak typowy serwer WWW — czy zwraca zawartość pliku podanego w URL lub status *404*, jeżeli plik nie istnieje?
   - http://localhost:8000/src/server_script1.js
   - http://localhost:8000/index.html
   - http://localhost:8000/favicon.ico
7. Utwórz aplikację "Księga gości" — plik *src/server_script2.js*:
   1. Pod adresem http://localhost:8000/ ma być dostępna strona WWW zawierająca:
      1. Poprzednie wpisy
      2. Formularz, w skład którego wchodzą:
         - Pole tekstowe 'imię i nazwisko
         - Obszar tekstowy 'treść wpisu'
         - Przycisk typu *submit*
   2. Po zatwierdzeniu formularza (po naciśnięciu przycisku), dane (imię i nazwisko oraz treść wpisu) są przesyłane do skryptu serwerowego *src/server_script2.js;* ten, na ich podstawie, dodaje nowy wpis do księgi (pliku tekstowego)

Przykładowy wygląd księgi gości

Adam Wolski

Odwiedzam Państwa stronę dosyć często. Jej wygląd jest prawie 👍 — rozmiar czcionek trochę zbyt mały.

Jan Kowalski

Pozdrawiam studentów rocznika 2023.

Anna Nowak

Studiowałam w latach 1988-1993.

Nowy wpis:

Twoje imię i nazwisko

Jerzy Wiśniewski

Treść wpisu

Proszę o kontakt osoby, które ze mną studiowały — tel. 12 345 67 89

Dodaj wpis

Aplikację należy napisać w "czystym" NodeJS, tj. bez użycia frameworka *Express* lub podobnego — wspomniany framework będzie tematem kolejnych zajęć

🏠 3.Narzędzia

1. Zainstaluj rozszerzenie "Code Runner" — `code --install-extension formulahendry.code-runner` , a następnie sprawdź na stronie rozszerzenia, w jaki sposób możesz, od teraz, uruchamiać skrypty JS
2. Wykonaj komendę `npm init @eslint/config # Konfiguracja lintera 'eslint'`

   Wybierz następujące opcje:
   1. To check syntax and find problems
   2. JavaScript modules (import/export)
   3. None of these
   4. No
   5. Node
   6. JavaScript
   7. Yes
   8. npm

3. Wyedytuj, za pomocą *Visual Studio Code*, dowolny z utworzonych plików źródłowych i zbadaj, czy linter znalazł jakieś nieprawidłowości — jeżeli edytor podkreślił jakiś fragment kodu, to najedź kursorem myszy na ten fragment i zobacz, co się wyświetla w "dymku"
4. Uruchom, w terminalu, komendę `npx eslint --fix 'src/*.js'` i sprawdź, czy treść kodu źródłowego uległa zmianie?
5. Wygeneruj dokumentację techniczną korzystając z komendy `npx jsdoc src --verbose` , a następnie obejrzyj ją (plik wynikowy *out/index.html*) za pomocą przeglądarki WWW
6. Wzbogać kod źródłowy, utworzonej przez Ciebie, aplikacji konsolowej lub serwerowej o komentarze dokumentacyjne programu JSDoc
7. Ponownie wygeneruj dokumentację techniczną
8. Zainstaluj pakiet SuperTest oraz Jest — `npm install supertest jest --save-dev`
9. Utwórz plik *test/server_script1.test.js* o następującej zawartości:

```
/* eslint-disable indent */
// Source:  https://codeforgeek.com/unit-testing-nodejs-application-using-mocha/
// Modified by Stanisław Polak <polak@agh.edu.pl>

const supertest = require('supertest');
// import supertest from 'supertest'; // "Jest" doesn't work well with ES6 modules – https://jestjs.io/docs/ecmascript-modules

// This agent refers to PORT where program is runninng.
const server = supertest.agent('http://localhost:8000');

// UNIT test begin
describe('GET /', () => {
    it('responds with "HTML form"', (done) => {
        server
            .get('/')
            .expect('Content-Type', /html/)
            .expect(200, /form/)
            .end((err, res) => {
                if (err) {
                    return done(err);
                }
                return done();
            });
    });
});

describe('GET /submit', () => {
    it('responds with welcome', (done) => {
        server
            .get('/submit')
            .query({ name: 'róża' })
            .expect(200, 'Hello róża')
            .end((err, res) => {
                if (err) return done(err);
                return done();
            });
    });
});

describe('POST /', () => {
    it('responds with welcome', (done) => {
        server
            .post('/')
            .type('form')
            .send({ name: 'róża' })
            .expect(200, 'Hello róża')
            .end((err, res) => {
                if (err) return done(err);
                return done();
            });
    });
});
// UNIT test end
```

10. Wykonaj komendy:

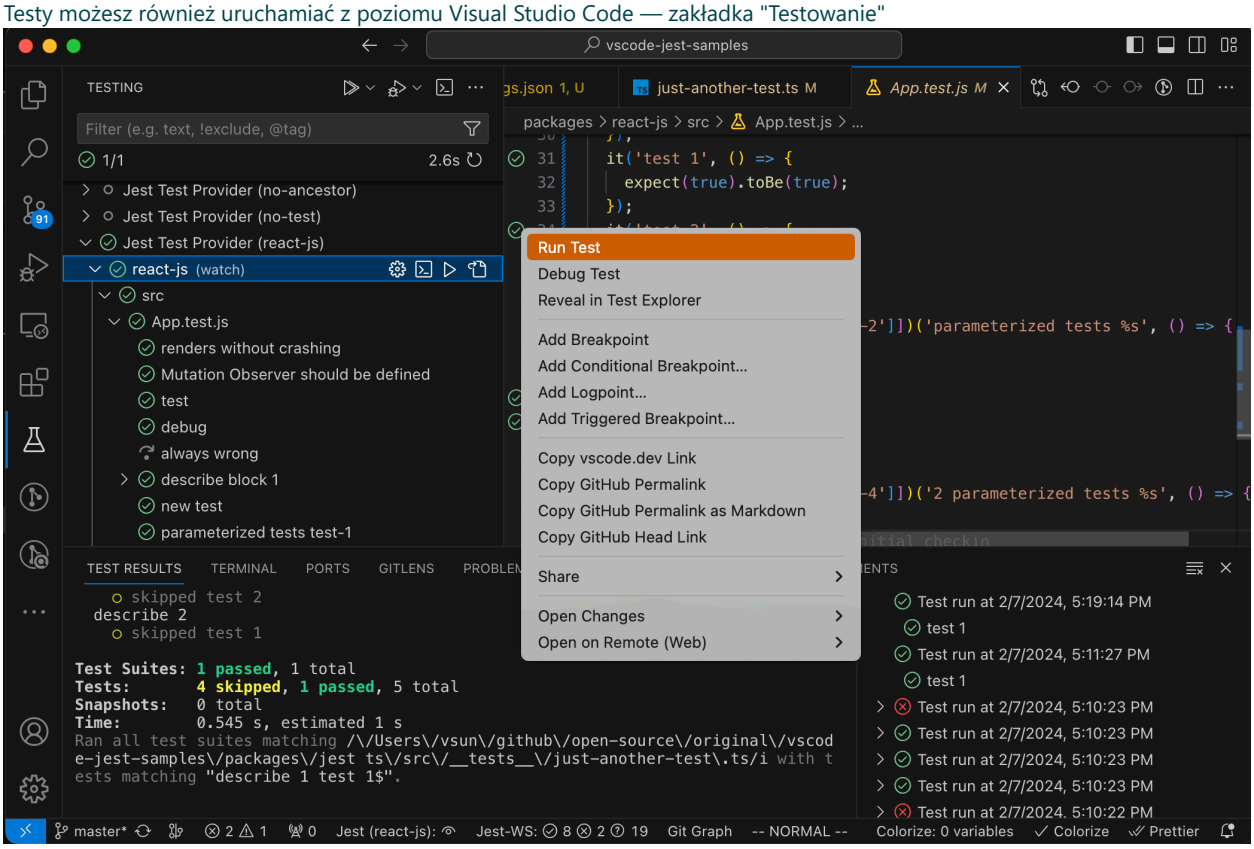
Terminal 1

```
node --watch src/server_script1
```

Terminal 2

```
npx jest # Uruchomienie testów
```

Testy możesz również uruchamiać z poziomu Visual Studio Code — zakładka "Testowanie"



11. Zmodyfikuj kod źródłowy aplikacji *src/server_script1.js* tak, aby test kończył się powodzeniem

    W oryginalnym przykładzie dane (z formularza) są przekazywane metodą GET (`<form method="GET" ...>`). W przypadku przesyłania danych poufnych należy użyć POST — patrz **GET a POST**. Aby skrypt *src/server_script1.js* obsługiwał tę metodę przesyłu, należy go uzupełnić o **dodatkowy fragment kodu**

12. Przeczytaj jak korzystać z debuggera

Dla ciekawskich

Tworząc kody źródłowe korzystaliśmy z **IDE** *Visual Studio Code*. W przypadku developmentu przydają się **task runnery oraz bundlery**.

- Obejrzyj film poświęcony systemom automatyzacji pracy

- W powyższym filmie pojawia się informacja, że zamiast systemów automatyzacji pracy, coraz częściej, używa się skryptów NPM — oglądnij film poświęcony tym skryptom

- W filmie nr 1 pojawia się wzmianka o narzędziu *Webpack* — obejrzyj film poświęcony temu programowi

- Jeżeli zamierzasz pracować jako Back-end Developer, to warto zaznajomić się z narzędziem **Postman** — na naszych zajęciach, raczej 🤔, Ci się nie przyda (ewentualnie, może być przydatny na szóstych zajęciach), gdyż aplikacja, którą będziemy tworzyć, będzie oferować dosyć ubogie 🙄 **REST API**

👷 4. Zadanie

- Zmodyfikuj aplikację z poprzednich ćwiczeń — szczegóły zostaną określone **na początku zajęć**
- Założenia:
  - Aplikacja ma działać w oparciu o model "klient-server"
  - Przetwarzanie danych, tylko, po stronie serwera; nie używamy JavaScript po stronie przeglądarki, ani do przetwarzania, ani do przechowywania danych
  - Dane mają być przechowywane w pliku tekstowym — format własny lub JSON

Edytuj zadanie | Usuń zadanie