

✓ Wykonano

W ramach zadań przygotowawczych do poprzedniego ćwiczenia stworzyliśmy prostą aplikację internetową w oparciu o "Vanilla Node.js", czyli bez użycia dodatkowych bibliotek czy frameworków. Dzisiaj stworzymy ją jeszcze raz, ale tym razem, przy użyciu specjalizowanego frameworka.

1. Prosta aplikacja

1. Wykonaj komendy:

```
1. mkdir -p cw5/test
2. cd cw5
3. ln ../cw4/test/server_script1.test.js test/test.js
4. npm init es6
5. npm install express morgan
6. npm install supertest --save-dev
```

Skrypty NPM

Wygenerowany plik `package.json` zawiera m.in. następujący fragment:

```
1. "scripts": {
2.   "dev": "",
3.   "test": ""
4. }
```

Sekcja "scripts" definiuje dwa **skrypty NPM** — "dev" oraz "test" — a więc są dostępne dwa polecenia: `npm run dev` oraz `npm run test`. Ponieważ treść obydwu skryptów jest pusta, więc wykonanie powyższych poleceń nie przyniesie żadnego efektu; ich treść zostanie wyspecyfikowana za chwilę

- Przeglądnij [listę wytyczek ExpressJS](#) dla *Visual Studio Code*, a następnie zainstaluj tę, która Twoim zadaniem będzie najbardziej przydatna
- Utwórz skrypt `app1.js` o poniższej zawartości — implementuje on tę samą funkcjonalność co skrypt `cw4/src/server_script1.js` — patrz konspekt poprzednich ćwiczeń

```

1.  /**
2.   * @author Stanisław Polak <polak@agh.edu.pl>
3.   */
4.
5.  import express from 'express';
6.  import morgan from 'morgan';
7.
8.  const app = express();
9.
10. /* ***** */
11. /* Determining the contents of the middleware stack */
12. /* ***** */
13.
14. app.use(morgan('dev')); // Place an HTTP request recorder on the stack – each request
    will be logged in the console in 'dev' format
15.
16. /* ***** */
17. /* "Routes" */
18. /* ***** */
19.
20. /* ----- */
21. /* Route 'GET /' */
22. /* ----- */
23. app.get('/', (request, response) => {
24.     // Generating the form if the relative URL is '/', and the GET method was used to
    send data to the server'
25.     response.send(`
26.         <!DOCTYPE html>
27.         <html lang="en">
28.             <head>
29.                 <meta charset="utf-8">
30.                 <meta name="viewport" content="width=device-width, initial-scale=1">
31.                 <title>First Express application</title>
32.             </head>
33.             <body>
34.                 <main>
35.                     <h1>First Express application</h1>
36.                     <form method="GET" action="/submit">
37.                         <label for="name">Give your name</label>
38.                         <input name="name">
39.                         <br>
40.                         <input type="submit">
41.                         <input type="reset">
42.                     </form>
43.                 </main>
44.             </body>
45.         </html>
46.     `); // Send the response to the browser
47. });
48.
49. /* ----- */
50. /* Route 'GET /submit' */
51. /* ----- */
52. app.get('/submit', (request, response) => {
53.     // Processing the form content, if the relative URL is '/submit', and the GET method
    was used to send data to the server'
54.     /* ***** */
55.     // Setting an answer header – we inform the browser that the returned data is plain
    text
56.     response.set('Content-Type', 'text/plain')
57.     /* ***** */
58.     // Place given data (here: 'Hello <name>') in the body of the answer
59.     response.send(`Hello ${request.query.name}`); // Send a response to the browser
60. });

```

```

61.
62.  /* ***** */
63.  // The application is to listen on port number 8000
64.  app.listen(8000, () => {
65.      console.log('The server was started on port 8000');
66.      console.log('To stop the server, press "CTRL + C"');
67.  });

```

4. Uruchom aplikację — `node --watch app1` (v18.11.0+) lub `npx nodemon app1`
5. Sprawdź jej działanie dla adresów wymienionych w pkt. 2.5 konspektu poprzedniego ćwiczenia
6. Przeczytaj "[Serving static files in Express](#)", a następnie dopisz do `app1.js` linię, która spowoduje, że będzie możliwe korzystanie z plików statycznych
7. Utwórz katalog dla plików statycznych, a w nim, podkatalog o nazwie `images`
8. Umieść, w podkatalogu, dowolny plik graficzny
9. Spowoduj, aby poniżej formularza wyświetlał się plik graficzny

2. System szablonów "Pug"

1. Wykonaj komendę `npm install pug`
2. Utwórz plik główny aplikacji — 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.
8.  /* ***** */
9.  /* Configuring the application */
10. /* ***** */
11. const app = express();
12.
13. app.locals.pretty = app.get('env') === 'development'; // The resulting HTML code will
    be indented in the development environment
14.
15. /* ***** */
16.
17. app.use(morgan('dev'));
18.
19. /* ***** */
20. /* "Routes" */
21. /* ***** */
22.
23. /* ----- */
24. /* Route 'GET /' */
25. /* ----- */
26. app.get('/', (request, response) => {
27.     response.render('index'); // Render the 'index' view
28. });
29.
30. /* ***** */
31.
32. app.listen(8000, () => {
33.     console.log('The server was started on port 8000');
34.     console.log('To stop the server, press "CTRL + C"');
35. });

```

3. Przeczytaj "[Using template engines with Express](#)", a następnie dopisz do `app2.js` fragment kodu, który spowoduje, że aplikacja będzie używać system szablonów "Pug"
4. Utwórz katalog dla szablonów, a w nim, [szablon Pug](#) o odpowiedniej nazwie (patrz nazwa widoku w pliku `app2.js`) i następującej zawartości (z zachowaniem wcięć):

```

1.  //- @author Stanisław Polak <polak@agh.edu.pl>
2.
3.  doctype html
4.  html(lang='en')
5.    head
6.      meta(charset='utf-8')
7.      meta(name='viewport' content='width=device-width, initial-scale=1')
8.      title Second Express application
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.     body
23.         main
24.             h1 Second Express application
25.             table
26.                 tr
27.                     th GET
28.                     th POST
29.                 tr
30.                     td
31.                         form(method='GET' action='/submit')
32.                             label(for='name') Give your name
33.                             input(name='name')
34.                             br
35.                             input(type='submit')
36.                             input(type='reset')
37.                     td
38.                         form(method='POST' action='/')
39.                             label(for='name') Give your name
40.                             input(name='name')
41.                             br
42.                             input(type='submit')
43.                             input(type='reset')

```

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

Powyższe komendy uruchamiają aplikację w trybie deweloperskim. Aby uruchomić ją w trybie **produkcyjnym**, należy powyższe wywołania poprzedzić ciągiem znaków `NODE_ENV=production` — przykład — `NODE_ENV=production node --watch app2`

6. Wykonaj komendę `npm pkg set 'scripts.test'='npx jest'` co spowoduje zdefiniowanie treści skryptu NPM "test" — linia "test":
`"",` pliku `package.json`, zostanie zastąpiona linią `"test": "npx jest"`
7. Ponieważ podczas tworzenia struktury katalogów — patrz punkt 1 — podlinowałeś/aś plik `../cw4/test/server_script1.test.js`, który został utworzony na poprzednich ćwiczeniach, uruchom testy — wykonaj komendę `npm run test`, `npm test` lub `npm t`
8. Zmodyfikuj kod źródłowy aplikacji `app2.js` tak, aby testy kończyły się powodzeniem — skorzystaj z: kodu źródłowego `app1.js` oraz przykładu "Obsługa formularzy" przedstawionego na [wykładzie](#)
9. Spowoduj, aby aplikacja działała tak jak aplikacja z sekcji 1, tzn. ma wyświetlać, poniżej formularza, plik graficzny
10. Utwórz, w pliku głównym, tablicę [obiektów](#) o nazwie `students`

```

1. let students = [
2.     {
3.         fname: 'Jan',
4.         lname: 'Kowalski'
5.     },
6.     {
7.         fname: 'Anna',
8.         lname: 'Nowak'
9.     },
10. ];

```

11. [Przełącz tę tablicę do widoku](#), a następnie spowoduj, aby widok wyświetlał zawartość odebranej tablicy w postaci poniższej tabeli HTML:

Nazwisko	Imię
Kowalski	Jan
Nowak	Anna

Wskazówka: użyj w szablonie jednej z **metod iteracji**

3. Baza danych "MongoDB"

Na zajęciach "Podstawy JavaScript" poznaliśmy — wersja trudniejsza zadania ćwiczeniowego — przeglądarkową bazę danych *IndexedDB*. Ponieważ teraz działamy po stronie serwera, a NodeJS nie oferuje wbudowanej bazy danych, potrzebujemy odrębnej (lokalnej lub zdalnej) bazy danych — **najczęściej używaną** jest **MongoDB**.

Opcje instancji MongoDB

Aby zainstalować instancję MongoDB, można użyć dowolnej z następujących metod:

- Zainstaluj produkt Community Edition na tym samym serwerze co aplikacja lub jako oddzielony serwer bazy danych.
- Zainstaluj produkt Enterprise Edition na tym samym serwerze, na którym znajduje się aplikacja lub jako oddzielony serwer bazy danych.
- Zasubskrybuj usługę Atlas MongoDB w chmurze.

— Cytat ze strony <https://www.ibm.com/docs/pl/process-mining/1.14.4?topic=integration-database>

1. [Zainstaluj](#) wtyczkę "[MongoDB for VS Code](#)"
2. Skorzystaj z chmurowej bazy danych [MongoDB Atlas](#) lub postaw lokalny serwer bazy danych — zainstaluj [MongoDB Community Edition](#)

MongoDB Atlas

Intro to MongoDB Atlas in 10 mins | Jumpstart



MongoDB Community Edition

Mongo tuts #7 Instalacja i podstawy MongoDB



3. Obejrzyj jeden z poniższych filmów

How to Use Visual Studio Code as Your MongoDB IDE



MongoDB for VS Code - MongoDB Developer Tools



4. Z poziomu IDE "Visual Studio Code":

1. Połącz się z serwerem MongoDB
2. Utwórz bazę danych o nazwie *AGH*
3. Utwórz kolekcję *students*
4. Korzystając z "Playground":
 1. Utwórz kilka dokumentów / rekordów — pojedynczy dokument ma przechowywać: imię, nazwisko studenta oraz akronim nazwy wydziału, np. WI, WIET, WMS, ...
 2. Wyszukaj studentów określonego wydziału

5. Wykonaj komendy

1. `cp app2.js app3.js`
2. `npm install mongodb`

6. Wykonaj komendę `npm pkg set 'scripts.dev'='node --watch app3'` — linia `"dev": ""`, pliku *package.json*, zostanie zastąpiona linią `"dev": "node --watch app3"`

7. Wyświetl listę dostępnych skryptów npm — `npm run`

8. Uruchom skrypt "dev" — `npm run dev` — spowoduje to uruchomienie Twojej aplikacji w trybie śledzenia zmian kodu źródłowego

9. Na podstawie [przykładu omówionego na wykładzie](#) zmodyfikuj treść skryptu *app3.js* oraz szablonu *index.pug* — aplikacja ma:

1. Pobierać dane (dokumenty) z bazy danych *AGH*, a wynik zapytania umieszczać w tablicy *students*. Tak więc, w porównaniu do kodu z sekcji 2, w tym zadaniu, zawartość tablicy *students* nie jest już zahardkodowana

2. Wyświetlać tabelę HTML z informacjami o studentach podanego wydziału — akronim podajemy w URL — przykładowy URL: <http://localhost:8000/WI>:

Nazwisko	Imię	Wydział
Kowalski	Jan	WI
Nowak	Anna	WI
...

1. Należy przyjąć, że akronimy nie są z góry znane, a więc nie mogą być zahardkodowane w skrypcie — użyj mechanizmu "Parametry trasy" omówionego na [wykładzie](#)
2. Ponieważ szablon Pug jest wspólny dla obydwu aplikacji (*app2* oraz *app3*), a kolumna *Wydział* występuje tylko w *app3*, będziesz musiał(a) zastosować [metodę warunkową](#).

Dla ciekawskich

1. W przypadku większych aplikacji, definiując trasy, używa się [obiektu 'Router'](#), a nie, pokazanego w powyższych przykładach, obiektu 'Express'

Trasowanie przy użyciu obiektu 'Express'

```
1. import express from 'express';
2.
3. const app = express();
4. ...
5.
6. app.get('/', (req, res) => {
7.   ...
8. })
9.
10. ...
```

Trasowanie przy użyciu obiektu 'Router'

```
1. import express from 'express';
2.
3. const app = express();
4. const router1 = express.Router();
5. const router2 = express.Router();
6. ...
7.
8. router1.get('/', (req, res) => {
9.   ...
10. })
11.
12. ...
13.
14. router2.get('/', (req, res) => {
15.   ...
16. })
17.
18. ...
19.
20. app.use(prefix1, router1);
21. app.use(prefix2, router2);
```

2. Istnieje kilka konkurencyjnych frameworków, takich jak np. [Fastify](#).
3. [Aplikacja Vanilla Node.js](#) bazująca na rozwiązaniach Express-a — middleware, trasy

4. W przykładzie pokazanym na wykładzie do łączenia z bazą danych użyto sterownika **MongoDB**. W przypadku bardziej złożonych projektów używa się **Mongoose**
5. **Trzy domyślne bazy danych w MongoDB**
6. **Schematy MongoDB**
7. Ponieważ korzystamy z bazy danych "MongoDB", aplikacja może być podatna na atak **NoSQL injection** — badaniem bezpieczeństwa oraz zabezpieczaniem aplikacji tworzonej podczas ćwiczeń laboratoryjnych zajmiemy się na ostatnich zajęciach
 - Narzędzia do badania podatności na atak: **NoSQLi**, **NoSQLMap**
 - **Pakiety zabezpieczające**
8. Kurs MongoDB

MongoDB w godzinę



9. Testy automatyczne stron internetowych (UI)

Playwright - Twój pierwszy test automatyczny #01 JS/TS (Sekcja 01, Lekcja 01)




4. Zadanie

- Zmodyfikuj aplikację z poprzednich ćwiczeń — szczegóły zostaną określone **na początku zajęć**
- Założenia:
 - Aplikacja ma być zbudowana w oparciu o pakiet *Express*
 - Dane mają być przechowywane w bazie danych *MongoDB*

Edytuj zadanie

Usuń zadanie

Status przesłanego zadania

Status przesłanego zadania	Przesłane do oceny
Stan oceniania	Nieocenione
Ostatnio modyfikowane	środa, 5 czerwca 2024, 19:33
Przesyłane pliki	<div><div> lab5.zip</div><div>5 czerwca 2024, 19:33</div></div>
Komentarz do przesłanego zadania	<div><div></div><div>▶ Komentarze (0)</div></div>

Informacja zwrotna

Ocena	4,20 / 5,00
Ocenił dnia	środa, 8 maja 2024, 15:13



Platforma obsługiwana przez:
[Centrum e-Learningu i Innowacyjnej Dydaktyki AGH](#)
[Centrum Rozwiązań Informatycznych AGH](#)

Pobierz aplikację mobilną 

Wybierz język 