✓ 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:

```
mkdir -p cw5/test
cd cw5
ln ../cw4/test/server_script1.test.js test/test.js
npm init es6
npm install express morgan
npm install supertest --save-dev
```

Skrypty NPM

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

```
"scripts": {
    "dev": "",
    "test": ""
}
```

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

2. Przeglądnij listę wtyczek ExpressJS dla *Visual Studio Code*, a następnie zainstaluj tę, która Twoim zadaniem będzie najbardziej przydatna
3. Utwórz skrypt *app1.js* o poniższej zawartości — implementuje on tę samą funkcjonalność co skrypt *cw4/src/server_script1.js* — patrz konspekt poprzednich ćwiczeń

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

import express from 'express';
import morgan from 'morgan';

const app = express();

/* ********************************************** */
/* Determining the contents of the middleware stack *
/* ********************************************** */

app.use(morgan('dev')); // Place an HTTP request recorder on the stack – each request will be logged in the console in 'dev' format

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

/* ------------- */
/* Route 'GET /' */
/* ------------- */
app.get('/', (request, response) => {
    // Generating the form if the relative URL is '/', and the GET method was used to send data to the server'
    response.send(`
    <!DOCTYPE html>
    <html lang="en">
       <head>
          <meta charset="utf-8">
          <meta name="viewport" content="width=device-width, initial-scale=1">
          <title>First Express application</title>
       </head>
       <body>
          <main>
             <h1>First Express application</h1>
             <form method="GET" action="/submit">
                <label for="name">Give your name</label>
                <input name="name">
                <br>
                <input type="submit">
                <input type="reset">
             </form>
          </main>
       </body>
    </html>
          `); // Send the response to the browser
});

/* ------------------- */
/* Route 'GET /submit' */
/* ------------------- */
app.get('/submit', (request, response) => {
    // Processing the form content, if the relative URL is '/submit', and the GET method was used to send data to the server'
    /* ********************************************** */
    // Setting an answer header – we inform the browser that the returned data is plain text
    response.set('Content-Type', 'text/plain')
    /* ********************************************** */
    // Place given data (here: 'Hello <name>') in the body of the answer
    response.send(`Hello ${request.query.name}`); // Send a response to the browser
```

```
});

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

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

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

import express from 'express';
import morgan from 'morgan';

/* ************************** */
/* Configuring the application */
/* ************************** */
const app = express();

app.locals.pretty = app.get('env') === 'development'; // The resulting HTML code will be indented in the development environment

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

app.use(morgan('dev'));

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

/* ------------- */
/* Route 'GET /' */
/* ------------- */
app.get('/', (request, response) => {
    response.render('index'); // Render the 'index' view
});

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

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

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**):

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

doctype html
html(lang='en')
    head
        meta(charset='utf-8')
        meta(name='viewport' content='width=device-width, initial-scale=1')
        title Second Express application
        style.
            table {
                width: 100%;
            }
            td {
                border: 1px solid #000;
                padding: 15px;
                text-align: left;
            }
            th {
                background-color: #04AA6D;
                color: white;
            }
    body
        main
            h1 Second Express application
            table
                tr
                    th GET
                    th POST
                tr
                    td
                        form(method='GET' action='/submit')
                            label(for='name') Give your name
                            input(name='name')
                            br
                            input(type='submit')
                            input(type='reset')
                    td
                        form(method='POST' action='/')
                            label(for='name') Give your name
                            input(name='name')
                            br
                            input(type='submit')
                            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*

```
let students = [
      {
            fname: 'Jan',
            lname: 'Kowalski'
      },
      {
            fname: 'Anna',
            lname: 'Nowak'
      },
];
```

11. Przekaż 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**.

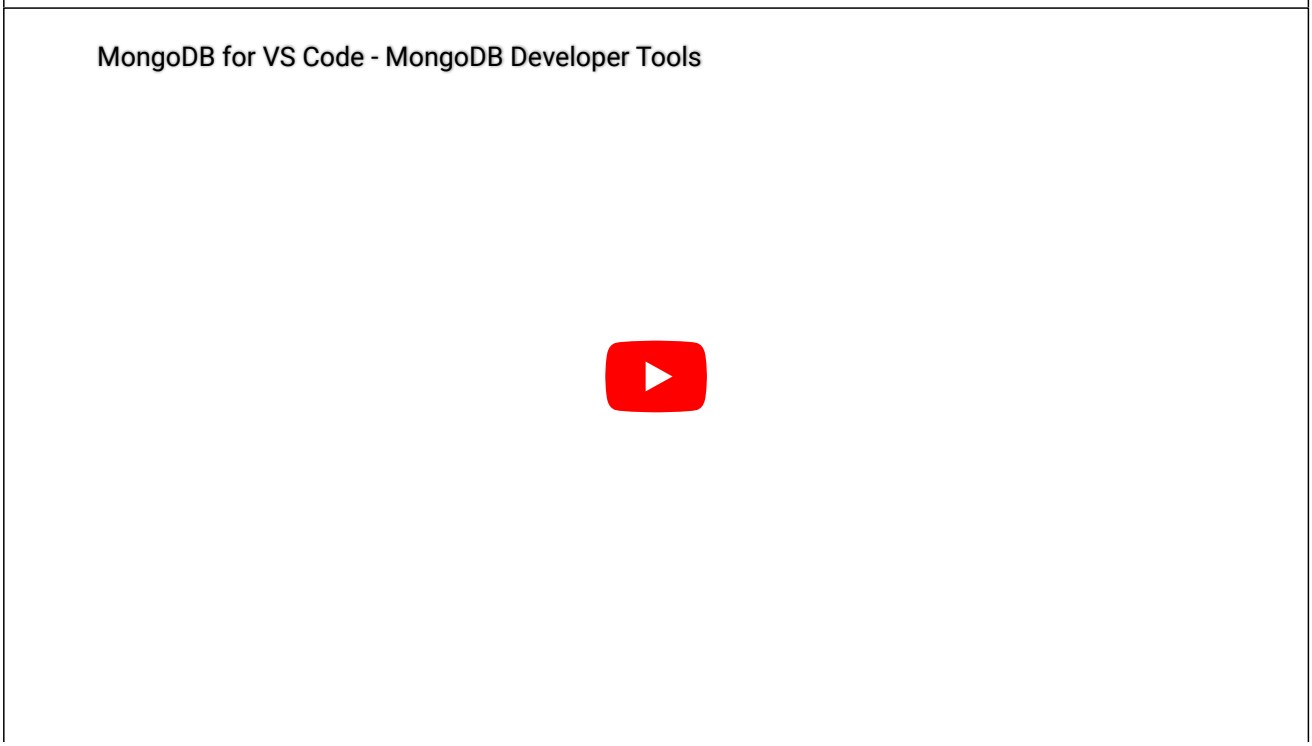
> ## Opcje instancji MongoDB
>
> Aby zainstalować instancję MongoDB, można użyć dowolnej z następujących metod:
>
> - Zainstaluj produkt Community Edition na tym samym serwerze co aplikacja lub jako oddzielony serwer bazy danych.
> - Zainstaluj produkt Enterprise Edition na tym samym serwerze, na którym 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

MongoDB Community Edition

3. Obejrzyj jeden z poniższych filmów



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

```
cp app2.js app3.js
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'

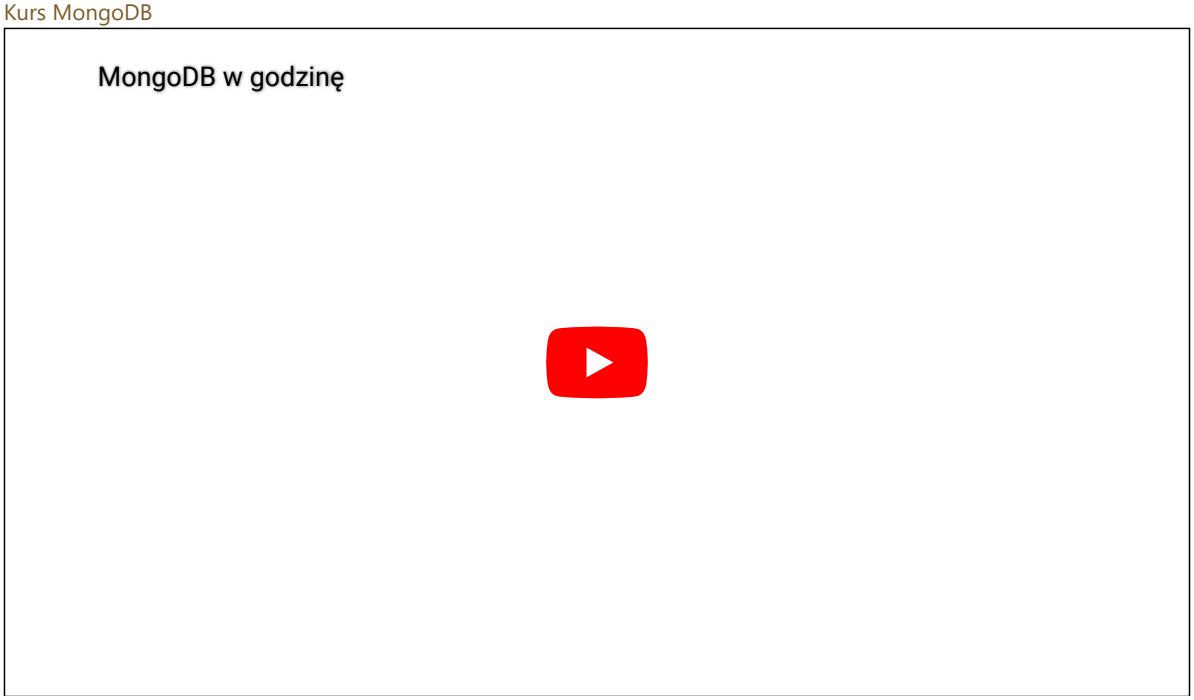
```
import express from 'express';

const app = express();
...

app.get('/', (req, res) => {
    ...
}

...
```

Trasowanie przy użyciu obiektu 'Router'

```
import express from 'express';

const app = express();
const router1 = express.Router();
const router2 = express.Router();
...

router1.get('/', (req, res) => {
    ...
}

...

router2.get('/', (req, res) => {
    ...
}

...

app.use(prefix1, router1);
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



9. Testy automatyczne stron internetowych (UI)

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