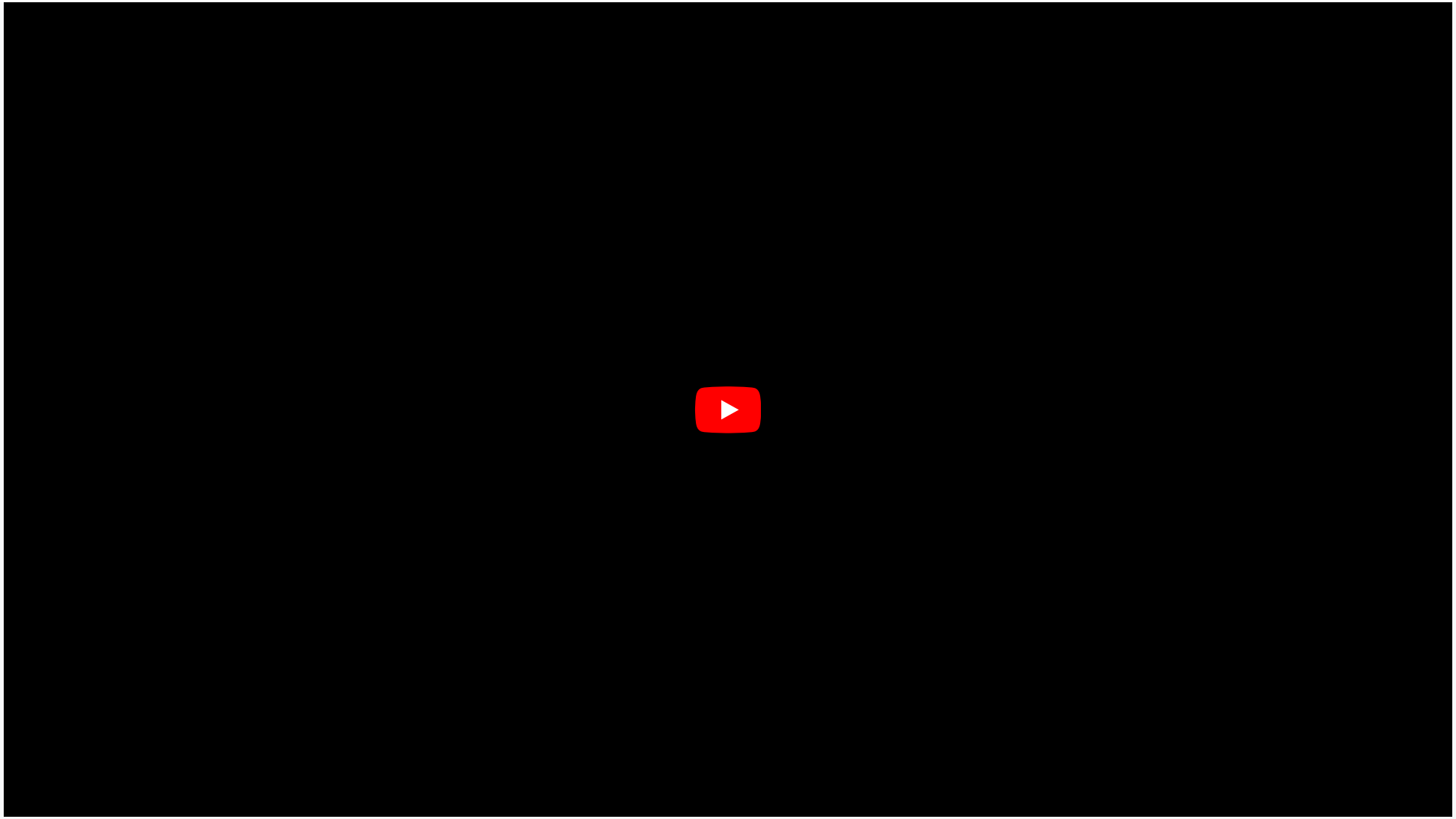
Framework Express

Express.js lub po prostu *Express* to framework dla Node.js, wzorowany na bibliotece Ruby o nazwie *Sinatra*, umożliwiający tworzenie aplikacji internetowych. W oparciu o *Express* stworzono, m.in., framework MVC o nazwie Sails.js — odpowiednik *Ruby on Rails*

1. Tworzenie prostej aplikacji

1. Utwórz katalog 'helloWorld'
2. Wejdź do utworzonego katalogu
3. Przeczytaj fragment artykułu poświęcony plikowi package.json



4. Za pomocą komendy `npm init` utwórz ten plik, a następnie obejrzyj jego zawartość — przykład generowania:

```
$ npm init

This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
```

```
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (helloworld)
version: (1.0.0)
description: The "Hello World" application
entry point: (index.js)
test command: npx mocha
git repository:
keywords:
author: Stanisław Polak <polak@agh.edu.pl> (https://www.icsr.agh.edu.pl/~polak/)
license: (ISC)
About to write to /home/polak/helloWorld/package.json:

{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "The \"Hello World\" application",
  "main": "index.js",
  "scripts": {
    "test": "npx mocha"
  },
  "author": "Stanisław Polak <polak@agh.edu.pl> (https://www.icsr.agh.edu.pl/~polak/)",
  "license": "ISC"
}

Is this OK? (yes)
```

5. Wykonaj następujące komendy

```
npm install express # ⇔ npm install express --save
npm install pug     # ⇔ npm install pug --save
npm install morgan  # ⇔ npm install morgan --save
```

6. Utwórz plik główny pierwszej aplikacji — 'app1.js':

```
// No use of any template system
var express = require('express'),
    logger = require('morgan');
var app = express();
var x = 1;
var y = 2;

// Determining the contents of the middleware stack
app.use(logger('dev'));                         // Place an HTTP request recorder on the stack – each request will be logged in the console in 'dev' format
// app.use(express.static(__dirname + '/public')); // Place the built-in middleware 'express.static' – static content (files .css, .js, .jpg, etc.) will be provided from the 'pu

// Route definitions
app.get('/', function (req, res) {     // The first route
    res.send('<h1>Hello World!</h1>'); // Send a response to the browser
});
```

```
// The application is to listen on port number 3000
app.listen(3000, function () {
    console.log('The application is available on port 3000');
});
```

> W przypadku większych aplikacji, definiując trasy, używa się obiektu 'Router' (`express.Router().<metodaHTTP>(...){...}`), a nie, pokazanego w powyższym przykładzie, obiektu 'Express' (`app.<metodaHTTP>(...){...}`)

7. Uruchom pierwszą aplikację: `node app1`
8. Wpisz w przeglądarce adres http://localhost:3000/
9. Przerwij wykonywanie aplikacji
10. Utwórz plik główny drugiej aplikacji — 'app2.js':

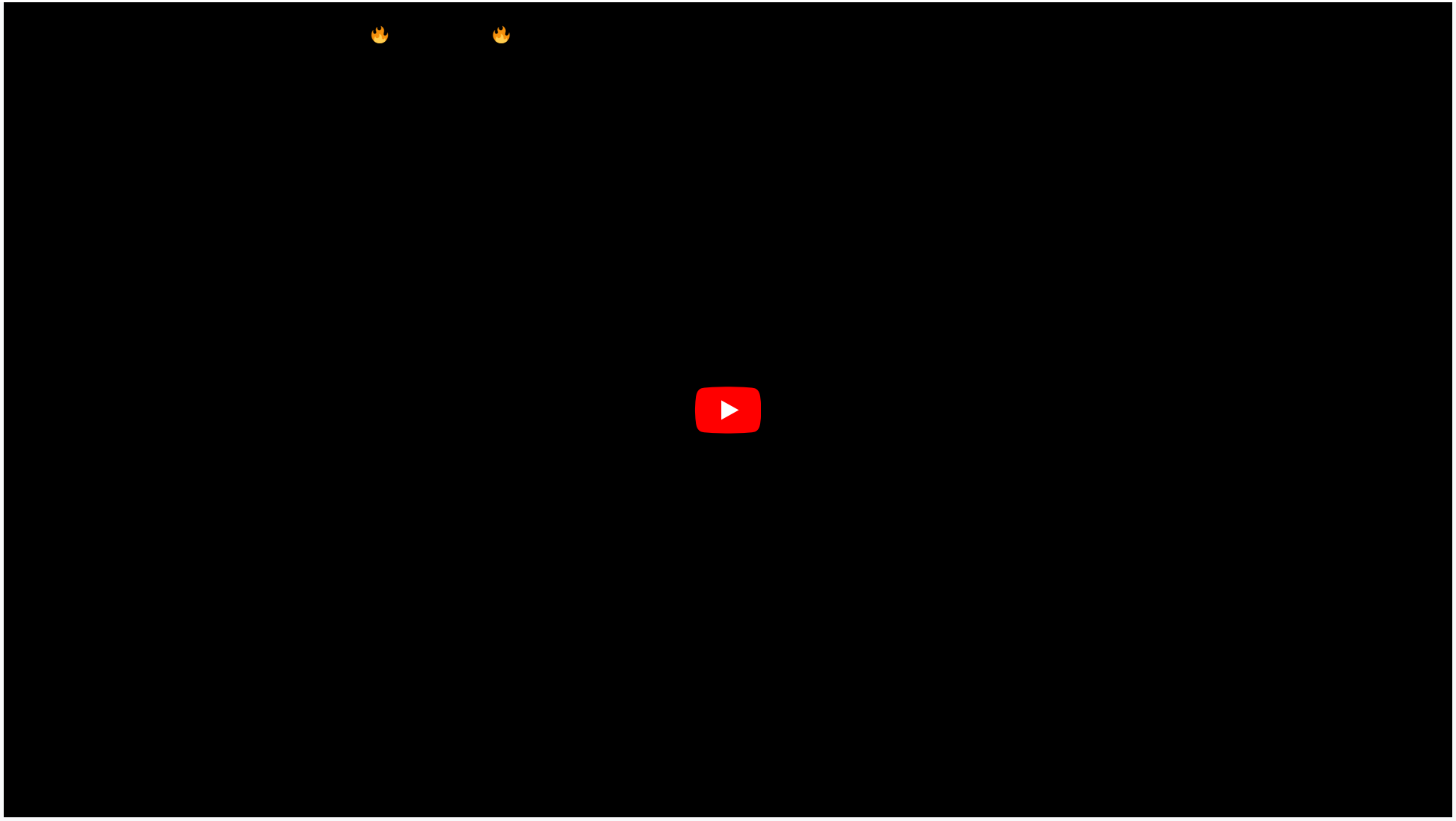
```
// Application using the 'Pug' template system
var express = require('express'),
    logger = require('morgan');
var app = express();
var x = 1;
var y = 2;

// Configuring the application
app.set('views', __dirname + '/views'); // Files with views can be found in the 'views' directory
app.set('view engine', 'pug');          // Use the 'Pug' template system

// Determining the contents of the middleware stack
app.use(logger('dev'));                       // Add an HTTP request recorder to the stack – every request will be logged in the console in the 'dev' format
// app.use(express.static(__dirname + '/public')); // Place the built-in middleware 'express.static' – static content (files .css, .js, .jpg, etc.) will be provided from the 'pu

// Route definitions
app.get('/', function (req, res) {      // The first route
    res.render('index', {pretty:true}); // Render the 'index' view in 'pretty' mode – the resulting HTML code will be indented – the 'pretty' option has the 'deprecated' status
    //res.render('index '); // Render the 'index' view; because the 'pretty' mode is, by default, turned off so the resulting HTML will be without indentation
});

// The application is to listen on port number 3000
app.listen(3000, function () {
    console.log('The application is available on port 3000');
});
```

11. Utwórz podkatalog 'views'
12. Utwórz (w tym katalogu), **zachowując wcięcia**, plik (szablon) 'index.pug' o następującej zawartości:

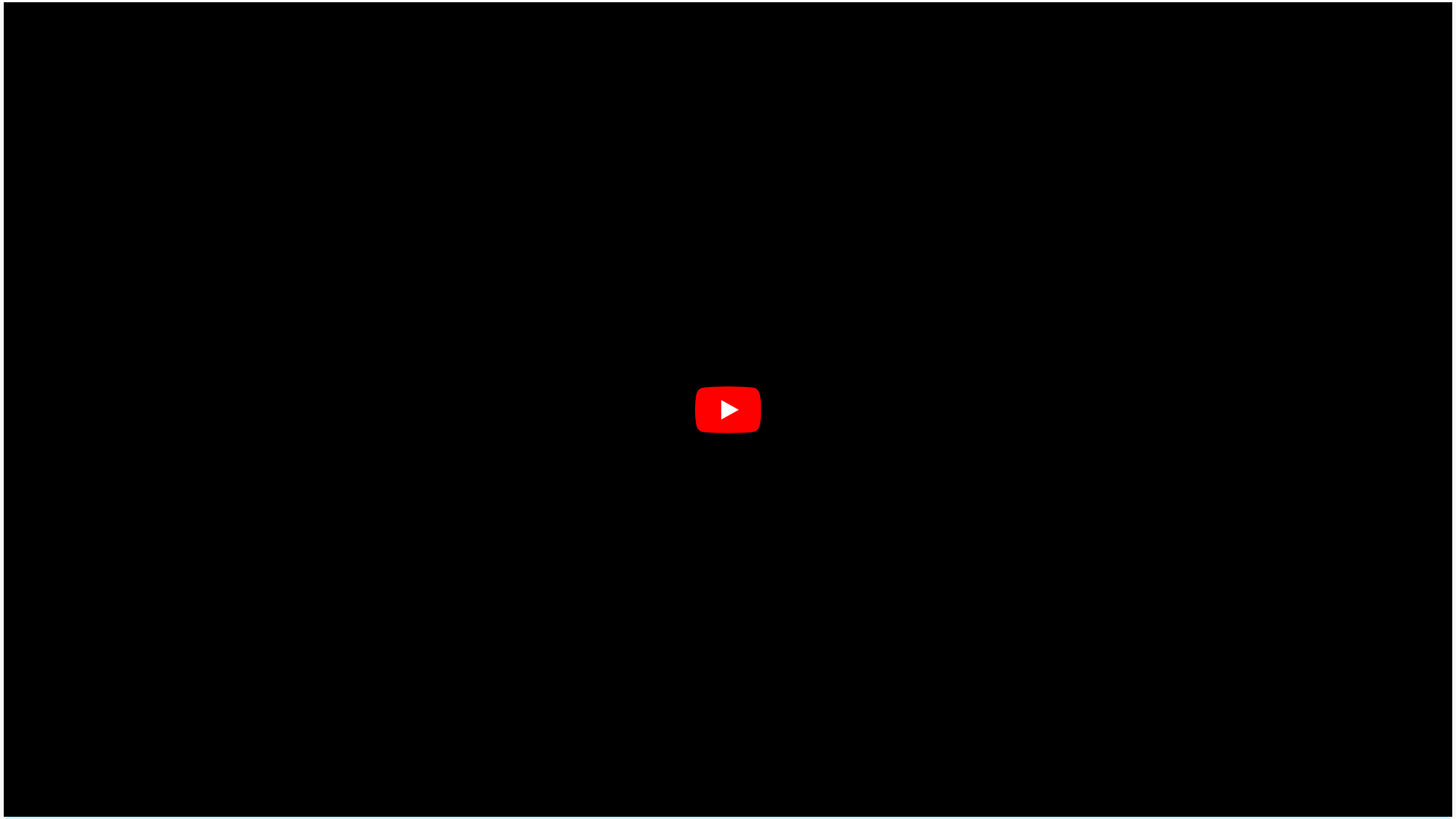
```
doctype html
html
    head
        meta(charset='utf-8')
        meta(name='viewport', content='width=device-width, initial-scale=1')
        link(href='https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css', rel='stylesheet')
        title Your first page
    body
```

```
    h1 Hello World
    script(src='https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js')
```

13. Uruchom drugą aplikację: `node app2`
14. Wykonaj komendę `curl http://localhost:3000` i zobacz jaki kod HTML wygenerował silnik 'pug' dla powyższego szablonu
15. Wpisz w przeglądarce adres http://localhost:3000/
16. Zainstaluj wtyczkę "ExpressJs 4 Snippets" dla *Visual Studio Code* — wykonaj komendę: `code --install-extension gurayyarar.expressjs-4-snippets`
17. Obejrzyj jej demo, a następnie wypróbuj niektóre z oferowanych, przez nią, snippetów
18. Zmodyfikuj kod **obydwu** aplikacji: mają obliczać, a następnie wypisywać, wartość `x+y`, gdzie *x* oraz *y* to zmienne, których wartość jest określona (zahardkodowana) w pliku 'app?.js'. Postać danych wynikowych: `<pierwszaLiczba> + <drugaLiczba> = <wynik>`
19. Zmodyfikuj zawartość pliku 'package.json' tak, aby po wykonaniu polecenia npm start uruchamiała się druga aplikacja
20. Usuń zawartość katalogu 'node_modules' (`rm -rf node_modules`), a następnie, za pomocą pojedynczego wywołania odpowiedniej komendy, przywróć możliwość uruchamiania / działania **obydwu** aplikacji
21. Alternatywą dla 'npm' jest 'yarn' — zaznajom się z opisem tego ostatniego
22. Ponownie usuń zawartość katalogu 'node_modules' oraz, dodatkowo, plik 'package-lock.json'. Przeczytaj artykuł, w którym porównano składnię 'npm' ze składnią 'yarn', a następnie zainstaluj pakiety niezbędne do działania aplikacji, korzystając z polecenia 'yarn, a nie 'npm'
23. Obejrzyj film poświęcony systemom automatyzacji pracy

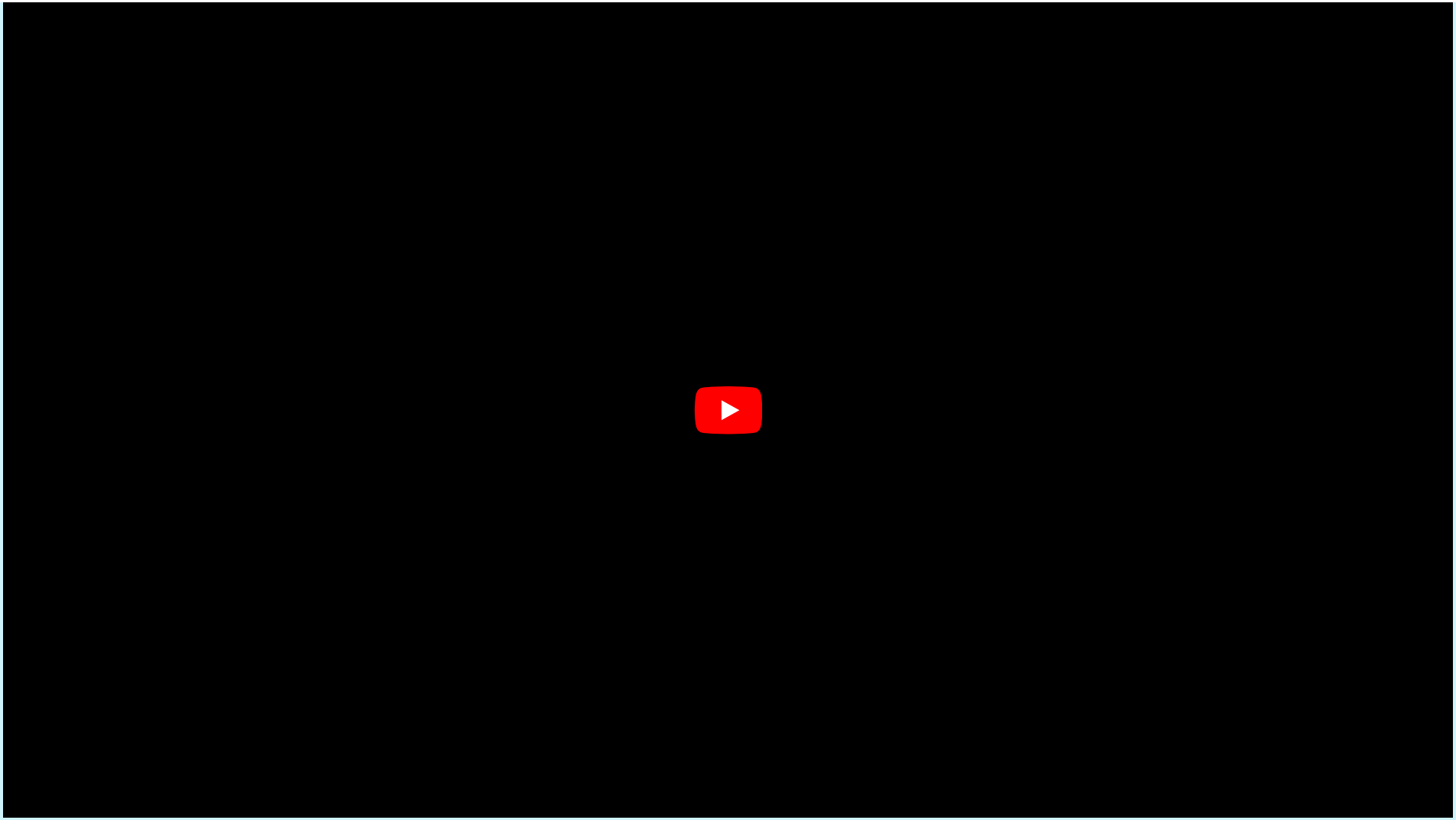


24. W powyższym filmie pojawia się informacja, że zamiast systemów automatyzacji pracy, coraz częściej, używa się skryptów NPM — oglądnij film poświęcony tym skryptom



Dla zainteresowanych

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



25. Stwórz dwa skrypty NPM:

    app1

    1. Wypisuje komunikat "`Uruchamiam aplikację nr 1`"
    2. Uruchamia aplikację nr 1

    app2

    1. Wypisuje komunikat "`Uruchamiam aplikację nr 2`"
    2. Uruchamia aplikację nr 2

Dla ambitnych

Napisz skrypt, który wykonuje, w podanej kolejności, następujące kroki:

1. Tworzy archiwum 'zip' zawierające całą zawartość katalogu 'helloWorld' (wraz z podkatalogami)
2. Usuwa wszystkie pliki '*.js' oraz '*.pug' zlokalizowane w katalogu bieżącym lub jego podkatalogach — w tym przypadku przydatny może być pakiet ShellJS

2. Obsługa danych złożonych w formacie JSON

1. Zainstaluj pakiet SuperTest — `npm install supertest --save-dev`
2. Utwórz katalog 'test', a w nim plik 'test.js' o następującej zawartości:

```
//Source:  https://codeforgeek.com/unit-testing-nodejs-application-using-mocha/
var supertest = require("supertest");

// This agent refers to PORT where program is runninng.
var server = supertest.agent("http://localhost:3000");

// UNIT test begin
describe('GET /', function() {
      it('respond with html', function(done) {
         server
         .get('/')
         .expect('Content-Type', /html/)
         .expect(200, done);
      });
});
```

3. Uruchom test — patrz poprzednie zajęcia — sprawdź, czy **obydwie** aplikacje (z pkt. 1) działają zgodnie z powyższym testem
4. Rozbuduj (**obydwie**) aplikacje o obsługę nowej trasy `app.get('/json/:name)` — aplikacja, w przypadku tej trasy:
   1. Wczytuje zawartość pliku JSON o podanej (w URL) nazwie — plik ma zawierać informacje o operacjach arytmetycznych oraz ich argumentach, tzn.: *operacja*, *x*, oraz *y*, gdzie:
      operacja
      Operacja arytmetyczna: dodawanie, odejmowanie, mnożenie lub dzielenie

      x
      Pierwszy argument operacji arytmetycznej

      y
      Drugi argument operacji arytmetycznej

   2. Oblicza wynik wykonania każdej z operacji
   3. Wyświetla ten wynik w postaci tabeli

x	Operation	y	Result
x_1	op_1	y_1	res_1
x_2	op_2	y_2	res_2

x	Operation	y	Result
...	...	...	...

- Struktura danych JSON — własna
- Należy przyjąć, że **w pliku można przechowywać wiele operacji**, a nie tylko jedną

Dla ciekawskich

'Express' potrafi odbierać oraz wysyłać dane JSON

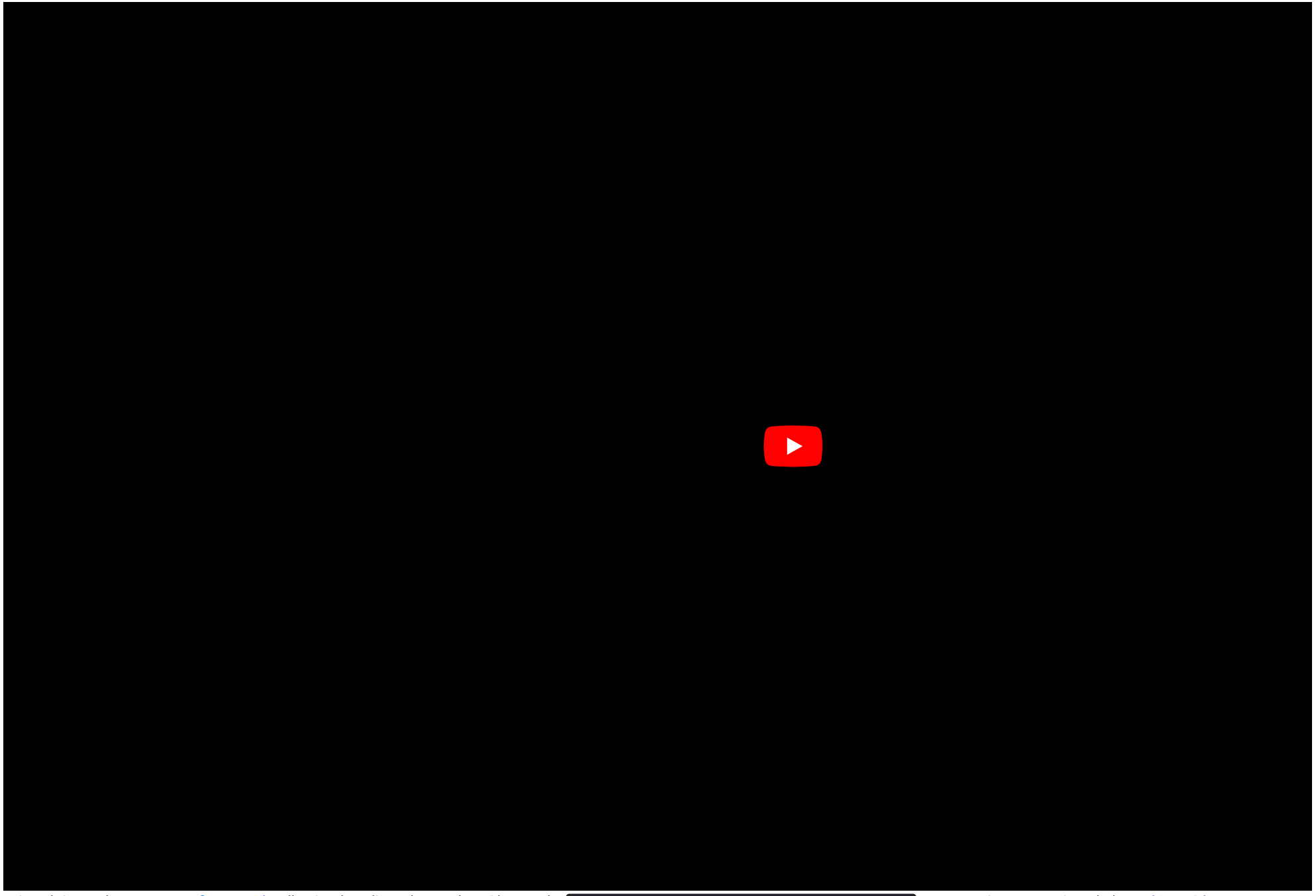
5. Zainstaluj bibliotekę Chai oraz wtyczkę Chai-json:

```
npm install chai --save-dev
npm install chai-json --save-dev
```

6. Napisz testy sprawdzające:
   - Poprawność generowanych wyników dla aplikacji dodających liczby (aplikacji z pkt. 1) — trasa `app.get('/')`
   - W przypadku trasy `app.get('/json/:name')`:
     1. Test 'supertest' — czy wygenerowany dokument HTML zawiera właściwe wyniki obliczeń arytmetycznych
     2. Test 'chai-json' — czy podany plik jest plikiem JSON i zawiera obiekt o podanych właściwościach — użyj `jsonObj()` oraz `jsonWithProps()`

3. Obsługa bazy danych MongoDB

1. Zainstaluj darmowy pakiet MongoDB Community Edition, który zawiera, między innymi, lokalny serwer bazy danych MongoDB albo skorzystaj z chmurowej bazy danych MongoDB Atlas



2. Zainstaluj wtyczkę "MongoDB for VS Code" dla *Visual Studio Code* — wykonaj komendę: `code --install-extension mongodb.mongodb-vscode`, a następnie przeczytaj artykuł "Working with MongoDB"

3. Rozbuduj **obydwie** aplikacje o obsługę nowej trasy `app.get('/calculate/:operation/:x/:y')`, dzięki czemu będzie można przekazywać, do aplikacji z pkt. 1, nie tylko wartości zmiennych *x* oraz *y*, ale równ
4. Procedura obsługi tej trasy, oprócz wypisywania danych wynikowych postaci: `<pierwszaLiczba> <operacja> <drugaLiczba> = <wynik>`, ma wstawiać do bazy danych *MongoDB* trzy wartości: *opera*

   Integracja z bazą danych MongoDB — dowolny ODM lub sterownik, np. MongoDB, Monk, Mongoose, ...

5. Dodaj nową trasę `app.get('/results')` — ma wypisywać zawartość bazy danych w postaci tabeli — patrz opis tabeli dla trasy `app.get('/json/:name')`
6. Zmodyfikuj zawartość pliku 'test.js' — dodaj test sprawdzający poprawność generowanych wyników dla tras:
   - `app.get('/calculate/:operation/:x/:y')`
   - `app.get('/results')`

4. Zadania

- Przeczytaj informacje na temat sposobów nawiązywania połączeń HTTP
- Przeczytaj informacje o sposobach dostępu do danych z formularza
- Napisz skrypt o podanej przez prowadzącego funkcjonalności, który w oparciu o system szablonów 'Pug', przetwarza dane JSON pochodzące z zewnętrznego źródła (API) lub z bazy danych MongoDB