Kurs JavaScript

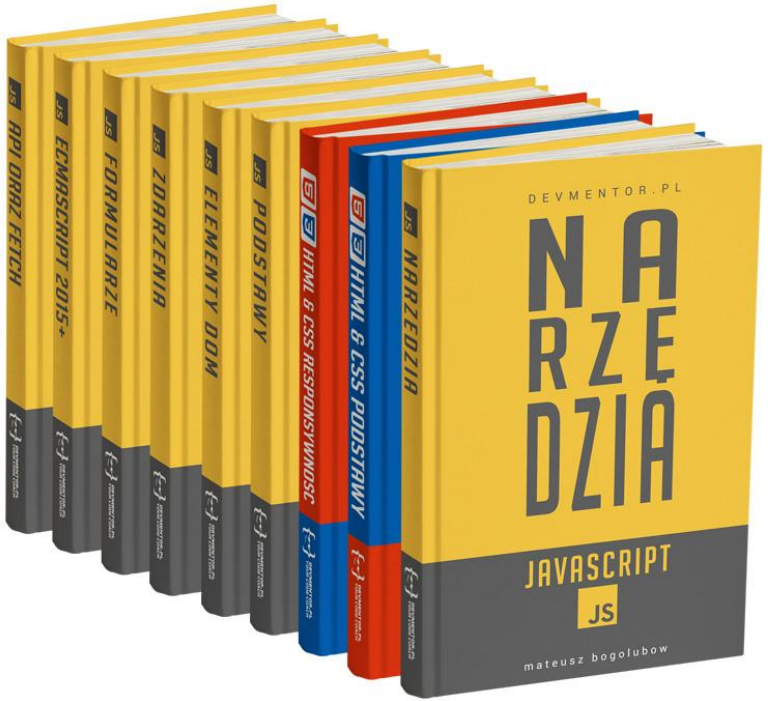
DEMO

Znajdziesz tu kilka przykładowych rozdziałów z ebooków. Wchodzą one w skład materiałów, które otrzymujesz w ramach mentoringu lub kursu JavaScript.

Pamiętaj, że są to fragmenty wyciągnięte z kontekstu, co wpływa na ich wartość merytoryczną.

Miłej lektury!

- Mateusz Bogolubow, Mentor Programowania



#04 Linia poleceń (terminal)

01. Visual Studio Code (IDE)

Wiersz poleceń (w Windows to tzw. *cmd*, a w systemach unixowych *terminal*) to pewien rodzaj tekstowego interfejsu.

Możemy za jego pomocą wykonywać działania, np. utworzenie pliku czy uruchomienie skryptu.

Często jest to wygodniejszy i szybszy sposób wykonywania operacji na komputerze, jednak trzeba się do niego przyzwyczaić.

Proponuję zapoznać się z podstawowymi komendami w systemie Windows jak również systemie unixowym (macOS/Linux).

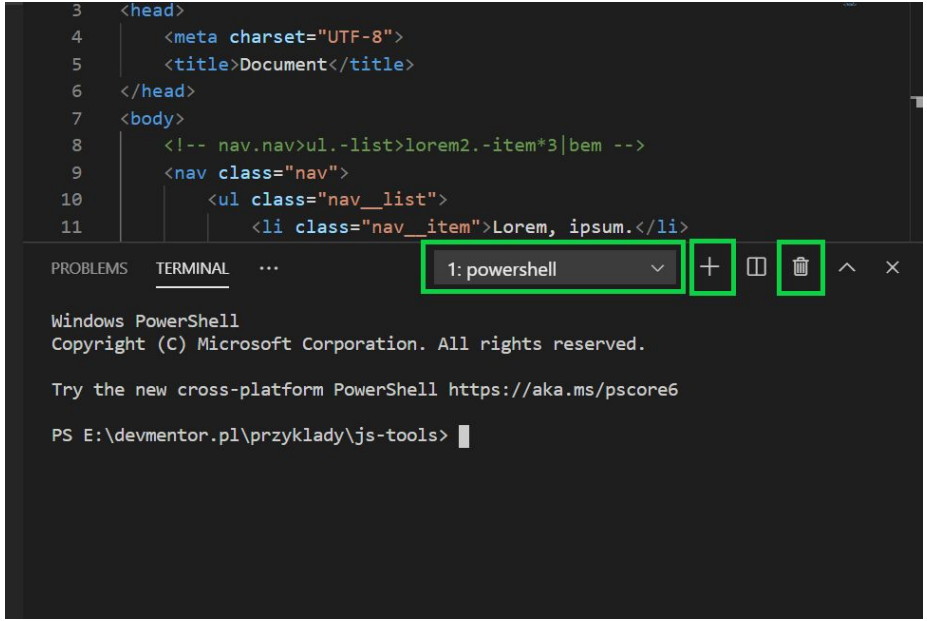
Programiści często korzystają z terminala w celu uruchomienia narzędzi, które usprawnią ich pracę (np. webpacka) lub zainstalują przydatne biblioteki (np. npm).

Będziemy te elementy omawiać jeszcze w tym materiale oraz w następnych.

Na chwilę obecną interesuje nas terminal, który jest dostępny w naszym *IDE*. Aby go uruchomić, wystarczy użyć skrótu `ctrl+`` (backtick).

Można uznać, że terminal w *IDE* ma tą samą funkcjonalność, co ten natywny, jednak dużo wygodniej jest korzystać z niego w edytorze kodu.

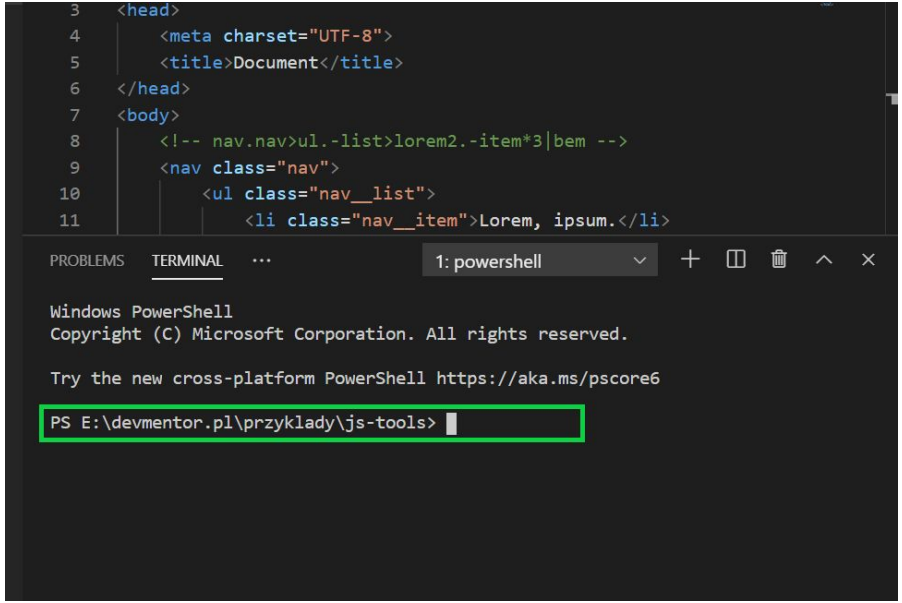
Jesteśmy w stanie uruchomić naraz kilka terminali (ikona plusa lub skrót `ctr+shift+``), zamknąć dowolny z nich (ikona śmietnika) lub wybrać aktywny.



Podczas korzystania z terminala bardzo ważnym elementem jest lokalizacja, w której uruchamiamy komendy.

Zazwyczaj terminal jest uruchamiany (odpalany) w lokalizacji, w której znajduje się nasz projekt.

Jednak jeśli potrzebujemy wprowadzić inną lokalizację, musimy użyć komendy cd - po której następuje wprowadzenie nowej lokalizacji - i zatwierdzić tę komendę klawiszem Enter.

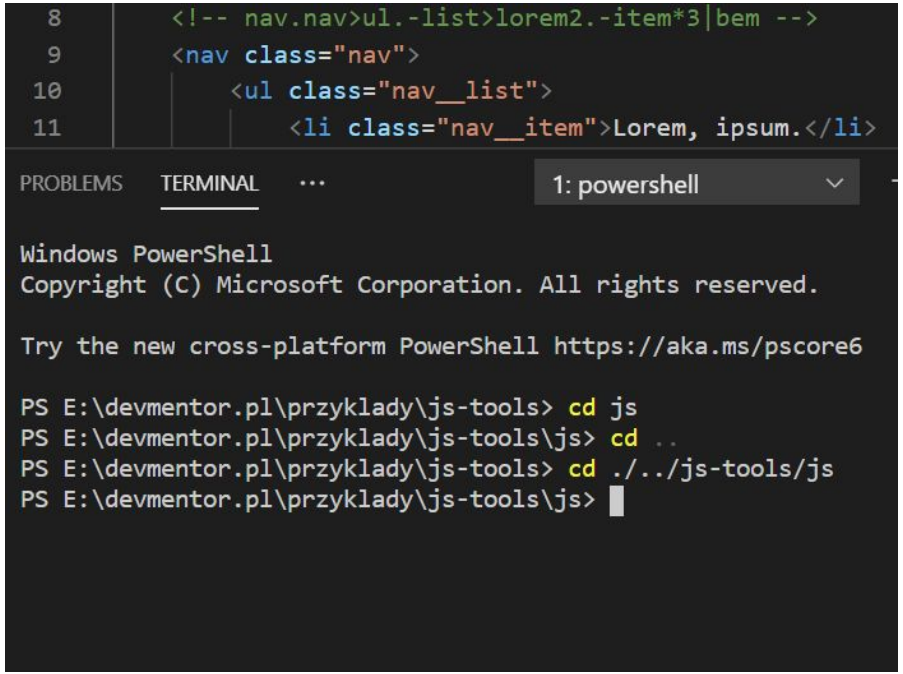


Dla przykładu, jeśli chcemy się przenieść do katalogu `js`, powinniśmy wpisać: `cd js`.

Jeśli chcielibyśmy przejść do katalogu znajdującego się wyżej w strukturze katalogów, to wykonamy polecenie `cd ..` (dwie kropki).

Możemy również wpisać rozbudowaną ścieżkę, w której katalogi są rozdzielone ukośnikiem: `cd ./../js-tools/js`.

Jedna kropka oznacza odniesienie względem obecnej lokalizacji.



#06 Semantyka

01. HTML

Semantyczny kod HTML to taki, w którym wykorzystujemy elementy zgodnie z ich przeznaczeniem.

Potrzebną dokumentację na ten temat wraz z przykładami znajdziesz m. in. na stronach w3.org lub whatwg.org.

Umiejętność jej czytania i odnajdywania tam potrzebnych informacji jest bardzo ważną częścią pracy programisty, dlatego już teraz warto zacząć się z nią oswajać.

Oczywiście nadal możesz wspomagać się wygodniejszą, ale mniej szczegółową informacją ze strony w3schools.com.

Zacznijmy może od przykładu z elementami liniowymi `<i>` oraz `<em>`, które w przeglądarce wyglądają identycznie, tzn. są napisane kursywą (CSS → `font-style: italic`).

Użycie każdego z nich powinno być zgodne z jego przeznaczeniem.

Element `<i>` stosuje się w przypadku, gdy część tekstu jest napisana w szczególnej formie, np. jest to termin techniczny, inny język, myśl itp.

Co ciekawe, powinien być on wykorzystywany tylko wtedy, gdy nie ma lepszego semantycznego dopasowania, jakim może być `<em>`.

Element `<em>` określa emfazę, czyli podkreślenie, uwypuklenie emocjonalne danego sformułowania.

Przykład:
Wstań z łóżka `<em>natychmiast</em>`!

Spojrzałem i pomyślałem: `<i>To nie może być prawdziwe!</i>`

Podobne rozróżnienie występuje w przypadku użycia elementów liniowych `<b>` oraz `<strong>`, które przeglądarka wyświetla jako pogrubione (CSS → `font-weight: bold`).

Znacznik `<strong>` określa ważny tekst, który może być np. ostrzeżeniem.

Natomiast znacznik `<b>` jedynie wyróżnia określony tekst i powinien być używany tylko wtedy, gdy reszta znaczników semantycznych nie pasuje w do danego kontekstu.

Przykład:

Przechodząc przez drogę, `<strong>zawsze sprawdzaj, czy nic nie jedzie<strong>`.

Dziś w szkole Jaś miał do zrobienia zadania z `<b>j. polskiego</b>` oraz `<b>chemii</b>`.

Nie wszystkie elementy HTML mają dokładnie określone przeznaczenie.

Przykładem takiego elementu jest `<div>` (blokowy) oraz `<span>` (liniowy).

Wykorzystujemy je do budowania struktury, która nie niesie za sobą dodatkowego znaczenia.

UWAGA. *Widoczny na przykładzie atrybut `style` pozwala nam zdefiniować liniowo style CSS, jednakże jest to uznawane za złą praktykę.*

```
<section>
    <div class="content">
        <!-- potrzebujemy wyśrodkować zawartość -->

        <p>
            Chciałbym pokazać jak wygląda kolor<br>
            <span style="color:burlywood">
                burlywood
            </span>!
        </p>
    </div>
</section>
```

Zobaczmy teraz jak mógłby wyglądać nagłówek naszej strony, czyli jej górna część zawierająca logo oraz menu.

Do określenia nagłówka użyliśmy elementu `<header>` - co było dość łatwe do przewidzenia.

Do deklaracji menu na naszej stronie zastosowaliśmy natomiast `<nav>` oraz listę nieuporządkowaną `<ul>`.

```
<header>
    <a href="/">
        <img src="./logo.png" alt="">
    </a>
    <nav>
        <ul>
            <li>
                <a href="/">home</a>
            </li>
            <li>
                <a href="/contact">kontakt</a>
            </li>
        </ul>
    </nav>
</header>
```

Główną zawartością strony, identyfikowaną przez znacznik <main>, jest lista odnośników do artykułów, które zawierają ich tytuł oraz krótki opis.

Pod treść w <h2> moglibyśmy podpiąć linki (czyli wykorzystać <a>), przekierowujące do stron konkretnych artykułów. Byłoby to dobrym rozwiązaniem pod względem użyteczności.

```
<main>
    <article>
        <h2>Lorem, ipsum dolor.</h2>
        <p>
            Lorem ipsum dolor sit amet consectetur.
            <a href="./art1.html">czytaj więcej</a>
        </p>
    </article>
    <article>
        <h2>Lorem, ipsum dolor.</h2>
        <p>
            Lorem ipsum dolor sit amet consectetur.
            <a href="./art2.html">czytaj więcej</a>
        </p>
    </article>
</main>
```

Stopka, `<footer>`, to w naszym przykładzie informacja o prawach autorskich oraz ponowne umieszczenie nawigacji.

Chciałbym jeszcze zwrócić Twoją uwagę na sposób zapisu znaku praw autorskich, tj. `&copy;`.

Użyłem do tego specjalnego zapisu, tzw. encji, który pozwala na umieszczenie na stronie znaku niedostępnego z poziomu klawiatury lub takiego, który mógłby zostać błędnie zinterpretowany przez przeglądarkę (np. znak większości lub mniejszości).

```
<footer>
    <div>&copy; 2020 devmentor.pl</div>
    <nav>
        <ul>
            <li>
                <a href="/">home</a>
            </li>
            <li>
                <a href="/contact">kontakt</a>
            </li>
        </ul>
    </nav>
</footer>
```

Na wcześniejszych slajdach zaprezentowałem Ci sposób zbudowania struktury strony zgodnie z semantyką.

Warto wiedzieć, że temat semantyki pojawił się wraz z HTML5, który wprowadził dodatkowe elementy, takie jak <header>, <footer>, <aside>, <section>, <main> itp.

Dlatego zapraszam Cię do zapoznania się z artykułem Semantyczny blog w HTML.

Znajdziesz tam dogłębną analizę przemiany HTML4 w HTML5 zgodnie z najnowszymi trendami.

Polecam również obejrzeć nagranie na temat pisania poprawnego HTML-a na YouTube.

Oba materiały mogą być dość trudne do zrozumienia na początku nauki, dlatego sugeruję wrócić do nich po zrealizowaniu zagadnień z HTML-a i CSS-a.

05. Responsywne menu

Menu uznajemy za responsywne, gdy w łatwy i wygodny sposób możemy przejść do interesującej nas podstrony.

W przypadku wersji mobilnej będzie to menu, którego elementy występują jeden pod drugim (pionowo), ponieważ potrzebujemy więcej miejsca na kliknięcie danego elementu.

Często menu pojawia się dopiero po kliknięciu w ikonę hamburgera.

W większych rozdzielczościach elementy menu zostaną po prostu wyświetlone poziomo - jeden obok drugiego - i będą widoczne od razu po załadowaniu strony.

Stworzymy teraz własne menu.

Najpierw opracujemy podstawową strukturę HTML.

Dla tego przykładu będziemy potrzebować kilku plików css, które dotyczą poszczególnych punktów granicznych.

Przyjmijmy, że już na wersji tabletowej będziemy chcieli uzyskać menu poziome.

Pamiętamy również o tagu `<meta>` z informacją o viewport!

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0"
    >
    <link rel="stylesheet" href="./css/global.css">
    <link rel="stylesheet" href="./css/mobile.css">
    <link rel="stylesheet" media="(min-width: 761px)"
        href="./css/tablet.css"
    >
<body>

</body>
</html>
```

Struktura HTML dla menu będzie wyglądać jak na przykładzie obok.

Całość umieszczamy w znaczniku `<header>`, który zawiera też logo w formie tekstu.

Sama nawigacja zgodnie z przeznaczeniem znajduje się w `<nav>`, a jej strukturę opieramy o listę nieuporządkowaną, czyli `<ul>`.

```
<header>
    <h1>Logo</h1>
    <nav>
        <ul>
            <li>
                <a href="#">start</a>
            </li>
            <li>
                <a href="#">portfolio</a>
            </li>
            <li>
                <a href="#">contact</a>
            </li>
        </ul>
    </nav>
</header>
```

W pliku global.css resetujemy podstawowe ustawienia, aby łatwiej nam było wykonywać dalsze prace.

Najlepszym rozwiązaniem byłoby wykorzystanie gotowego pliku reset.css lub normalize.css, ale o tym już wspominałem w poprzednich materiałach.

Tutaj skupiamy się na samym menu.

```css
/* ./css/global.css */

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

ul {
    list-style: none;
}

nav a {
    color: #336699;
    text-decoration: none;
}
```

Ponieważ nasze elementy `<li>` są blokowe, to automatycznie ułożyły się jeden pod drugim - nic nie musieliśmy robić.

Dla lepszej prezentacji dodajemy wyśrodkowanie tekstu dla `<h1>` oraz `<li>`.

Dodatkowo elementy `<li>` otrzymują odstępy oraz obramowanie.

```css
/* ./css/mobile.css */

header h1 {
    text-align: center;
}

header li {
    border: 1px solid #bbb;
    margin: 5px;
    padding: 5px 10px;
    text-align: center;
}
```

Dla wersji tabletowej elementy menu ustawiamy obok siebie, korzystając z flexboxa.

Najpierw ustawiamy go dla `<header>`, aby wypozycjonować logo, tj. `<h1>`, oraz `<nav>`.

Za pomocą `justify-content` przesuwamy dzieci `<header>` do boków, a dzięki `align-items` wyśrodkowujemy je w pionie.

Flexboxa używamy również dla ustawienia elementów menu.

```
/* ./css/tablet.css */

header {
    display: flex;
    justify-content: space-between;
    align-items: center;
}

header ul {
    display: flex;
}
```

Teoretycznie mamy już wszystko ustawione, jednak powiedzieliśmy sobie na początku, że menu na mobile powinno być widoczne po kliknięciu w hamburgera.

W przykładzie zamiast hamburgera będziemy mieli napis “menu”.

Jak sprawić, by nasze menu otwierało się i zamykało po kliknięciu na ten napis?

Najpierw dopiszemy go do naszej struktury, używając elementu `<label>` - zaraz dowiesz się dlaczego.

```
<header>
    <h1>Logo</h1>
    <nav>
        <label>menu</label>
        <ul>
            <li>
                <a href="#">start</a>
            </li>
            <li>
                <a href="#">portfolio</a>
            </li>
            <li>
                <a href="#">contact</a>
            </li>
        </ul>
    </nav>
</header>
```

Teraz należałoby ukryć nasze elementy menu i pokazać je dopiero po kliknięciu.

Z JavaScript byłoby to bardzo proste, ale na razie go nie znamy. Wykorzystamy więc inne rozwiązanie.

Użyjemy elementu `<input>` typu `checkbox`, który możemy zaznaczać i odznaczać. Jeśli element ten będzie zaznaczony, to pokażemy menu, jeśli nie - to je ukryjemy.

Na początku osadzimy element `<input>` w kodzie HTML i powiążemy go z `<label>` za pomocą `for`.

```
<header>
    <h1>Logo</h1>
    <nav>
        <label for="menu-switcher">menu</label>
        <input type="checkbox" id="menu-switcher">
        <ul>
            <li>
                <a href="#">start</a>
            </li>
            <li>
                <a href="#">portfolio</a>
            </li>
            <li>
                <a href="#">contact</a>
            </li>
        </ul>
    </nav>
</header>
```

Domyślnie ukryjemy całą nawigację dzięki `display: none`.

Wykorzystując pseudo klasę `:checked` oraz selektor najbliższego rodzeństwa (`+`) będziemy mogli pokazać nasze menu (`display: block`) po kliknięciu.

Zwróć uwagę, że teraz nasze menu staje się widoczne po zaznaczeniu checkboxa.

Checkbox można zaznaczyć również przez kliknięcie w `<label>`!

```
/* ./css/mobile.css */

/* ... */
header ul {
    display: none;
}

header input:checked + ul {
    display: block;
}
```

Skoro wystarczy kliknięcie w `<label>` , to ukrywamy `<input>` oraz ustawiamy w odpowiednim miejscu nasz napis “menu”.

Na wersji mobilnej wszystko wygląda dobrze.

Teraz musimy sprawdzić, jak menu prezentuje się w wersji tabletowej.

```css
/* ./css/mobile.css */

/* ... */
header {
    position: relative;
}

header label {
    position: absolute;
    right: 10px;
    top: 10px;
}

header input {
    display: none;
}
```

Dla rozdzielczości większej bądź równej `761px` musimy jedynie ukryć nasz napis “menu”, który jest tutaj całkowicie niepotrzebny.

To wszystko! Nasze menu jest użyteczne i funkcjonalne dla mniejszych i większych rozdzielczości!

UWAGA. *Cały kod możesz podejrzeć w repozytorium, a jego działanie sprawdzisz na tej stronie.*

```css
/* ./css/tablet.css */

header {
    display: flex;
    justify-content: space-between;
    align-items: center;
}

header ul {
    display: flex;
}

header label {
    display: none;
}
```

02. Propagacja zdarzeń

Propagacja to rozchodzenie się zdarzeń na kolejne elementy.

Po kliknięciu w `<button>` zdarzenie zostanie wywołane również na `<section>`.

```
<section>
    <button>click me</button>
</section>
```

```
const btnEl = document.querySelector('button');
const sectionEl = document.querySelector('section');
if(btnEl && sectionEl) {
    btnEl.addEventListener('click', function() {
        console.log('button was clicked');
    });

    sectionEl.addEventListener('click', function() {
        console.log('section was clicked too!');
        // zauważ, że klikając w <button />
        // równocześnie klikasz też w <section/>
        // która zawiera w sobie ten przycisk
    });
}
```

Propagację można przyrównać np. do chodzenia po dywanie.

Jeśli chodzisz po dywanie, to oznacza, że chodzisz również po podłodze, prawda?

Każdy nasz krok oddziałuje przez dywan na podłogę (i na odwrót). Podobnie dzieje się przy propagacji!

Klikając w przycisk jednocześnie klikamy w inne elementy, które znajdują się pod nim (są jego przodkami).

```
const btnEl = document.querySelector('button');
const sectionEl = document.querySelector('section');
if(btnEl && sectionEl) {
    btnEl.addEventListener('click', function() {
        console.log('button was clicked');
    });

    sectionEl.addEventListener('click', function() {
        console.log('div was clicked too!');
        // zauważ, że klikając w <button />
        // równocześnie klikasz też w <div/>
        // który zawiera w sobie ten przycisk
    });
}
```

Zastanówmy się jeszcze, w jaki sposób moglibyśmy wywołać na obu elementach tą samą funkcję, która doda do kliknięte go elementu klasę, będącą nazwą znacznika.

Z pomocą przyjdzie nam tutaj specjalna zmienna this, której wartość zależy od kontekstu, tj. miejsca wywołania funkcji.

```
const btnEl = document.querySelector('button');
const sectionEl = document.querySelector('section');

const addClassToElement = function() {
    const className = this.tagName.toLowerCase();
    // pobieram nazwę klikniętego tagu
    // oraz zamieniam wielkie litery na małe
    this.classList.add(className);
    console.log(className + ' was clicked');
}

if(btnEl && sectionEl) {
    btnEl.addEventListener('click', addClassToElement);
    // this wskazuje na [btnEl]
    sectionEl.addEventListener('click', addClassToElement);
    // this wskazuje na [sectionEl]
}
```

Sama zmienna this nie jest ściśle powiązana z efektem propagacji.

Często this jest wykorzystywana przy rozprzestrzenianiu się zdarzeń, jednak ma też inne zastosowanie.

Dzięki tej zmiennej możemy odwołać się do konkretnego elementu, na którym zostało wywołane zdarzenie (i tylko do niego!), mimo że nasłuchujemy tego samego eventu na wielu elementach i po jego wystąpieniu wywołujemy tę samą funkcję.

```
<section>
   <button>1</button><button>2</button><button>3</button>
</section>
```

```
const btnsList = document.querySelectorAll('button');
const changeText = function() {
    this.innerText = 'clicked';
    // zmień tekst na kliknięty elemencie
}
btnsList.forEach(function(btnEl) {
    // do każdego przycisku
    // który jest dostępny pod zmienną [btnEl]
    btnEl.addEventListener('click', changeText);
    // dopisz nasłuchiwanie na event [click]
    // [this] będzie wskazywał na [btnEl]
});
```

Już wiesz, że propagacja to rozprzestrzenianie się zdarzenia po elementach będących w relacji rodzic - dziecko.

Propagacja posiada 2 fazy. Pierwsza polega na przemieszczaniu się zdarzenia od korzenia do elementu, na którym został wywołany event.

Druga faza to powrót - zaczynamy od elementu, na którym wywołano zdarzenie i kończymy na korzeniu.

```
const btnEl = document.querySelector('button');
const sectionEl = document.querySelector('section');
if(btnEl && sectionEl) {
    btnEl.addEventListener('click', function() {
        console.log('button was clicked');
    });

    sectionEl.addEventListener('click', function() {
        console.log('section was clicked too!');
        // zauważ, że klikając w <button />
        // równocześnie klikasz też w <div/>
        // który zawiera w sobie ten przycisk
    });
}
```

#01 Wprowadzenie

04. JSON Server

JSON Server to rozwiązanie, które pozwala na uruchomienie *API* na naszym komputerze, dzięki czemu mamy nad nim całkowitą kontrolę.

Nie musimy martwić się o limity czy inne ograniczenia (np. brak Internetu), aby testować nasze rozwiązanie.

Aby skorzystać z *JSON Server,* musisz zainstalować *Node.js*, ponieważ *JSON Server* działa w oparciu o to środowisko.

W tym celu wystarczy odwiedzić stronę środowiska Node.js, pobrać instalkę dla odpowiedniego systemu operacyjnego i podążać za instrukcjami.

Jeśli wcześniej już instalowałeś *Node.js*, to sprawdź, czy Twoja wersja jest **nowsza lub równa wersji 12**. Wystarczy, że w terminalu wpiszesz: `node -v`.

Jest to również dobry sposób na sprawdzenie, czy instalacja przebiegła pomyślnie. Powyższa komenda powinna wówczas wyświetlić najnowszą wersję *Node.js*.

Teraz możemy przystąpić do instalacji naszego lokalnego *API*.

Aby to zrobić, należy wprowadzić poniższą komendę do terminala:

```
npm install json-server -g
```

Uwaga! *Osoby używające systemu operacyjnego innego niż Windows powinny poprzedzić tą komendę słowem* `sudo`*, aby wykonać ją z uprawnieniami administratora.*

Od teraz *JSON Server* jest zainstalowany w naszym systemie globalnie i możemy uruchamiać go w terminalu z dowolnego miejsca.

JSON Server wszystkie dane przechowuje w pliku w formacie JSON (oczywiste), dlatego zaraz taki plik utworzymy i uruchomimy nasze *API*.

W dowolnej lokalizacji tworzymy katalog `json-server-helloworld`.

W nim tworzymy plik `data.json` i wprowadzamy do niego kod, który widzisz obok.

Następnie odpalamy terminal w lokalizacji, w której znajduje się nasz plik w formacie *JSON* i wprowadzamy poniższą komendę: `json-server ./data.json --watch`.

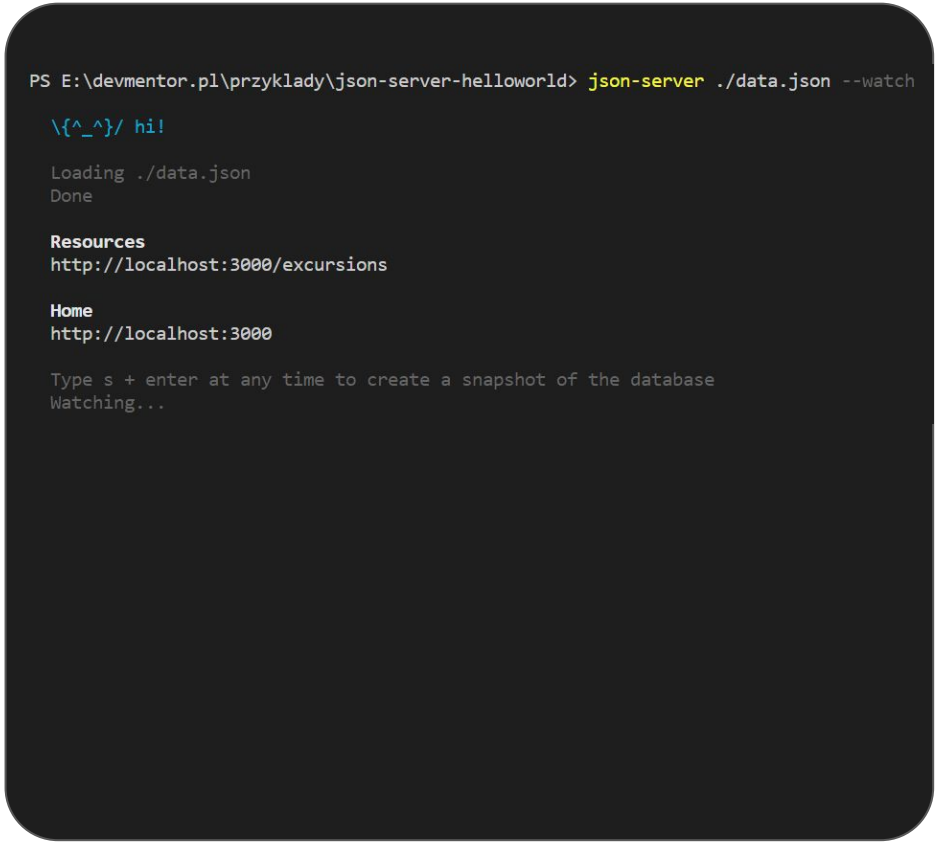
```
{
    "excursions": [
        {
            "id": 1,
            "name": "Zwiedzanie Krakowa",
            "price": 120
        },
        {
            "id": 2,
            "name": "Zwiedzanie Warszawy",
            "price": 120
        }
    ]
}
```

W konsoli powinien wyświetlić się adres, pod którym dostępne są zasoby.

W moim przypadku jest to adres: http://localhost:3000/excursions

U Ciebie, jeśli wszystko wykonałeś poprawnie i nie masz zablokowanego portu *3000,* adres, na którym nasłuchuje nasze *API*, najpewniej będzie ten sam.

Teraz, wprowadzając ten adres do przeglądarki, możesz zobaczyć zawartość pliku data.json.



Od teraz Twoje lokalne *API* działa! Jeśli chcesz je zatrzymać, używasz skrótu klawiszowego `ctrl+c`.

Jeśli zamkniesz terminal, musisz ponownie uruchomić *JSON Server* (tak jak wcześniej), aby *API* znów działało.

Wszystkie dane będą od teraz zapisywane w pliku `data.json`.

Jeśli chcesz uruchomić inną aplikację w terminalu, np. *webpacka,* to musisz włączyć osobny terminal.

W terminalu w programie VS Code wystarczy kliknąć plus (`+`) w prawym górnym rogu jego sekcji.

Przykładowe projekty*

Wykonane przez uczestników mentoringu

* Nie każdy projekt jest dopracowany graficznie. Uczestnik ma możliwość wyboru, czy skupi się jedynie na stronie technicznej, czy także wizualnej.

LANDING PAGE	SLIDER	REZERWACJA WYCIECZEK strona użytkownika; strona admina
• strona jest wykonana zgodnie z zasadami **RWD** (Responsive Web Design) • wykorzystuje jedynie HTML i CSS • menu działa bez JavaScriptu i prezentuje się dobrze w każdej rozdzielczości • po wykonaniu takiej strony jest się gotowym do wykonywania pierwszych **projektów komercyjnych**	• projekt kładzie nacisk na umiejętność pracy z **nie swoim kodem** (należy dokończyć programowanie slidera, stosując się do zastanej konwencji) • slider wykorzystuje, często pomijane w procesie nauki, zdarzenia niestandardowe (tzw. **custom events**) • projekt działa na zasadzie pokazu slajdów i zatrzymuje się po najechaniu na przycisk przełączenia obrazu	• projekt wymaga wykonania panelu klienta i panelu administratora • wykorzystuje **lokalne API**, które pozwala na komunikację między panelami • panel klienta posiada **koszyk** (produkty można dodawać i usuwać) • informacje o zamówieniu są przekazywane do API dopiero po wprowadzeniu danych klienta (np. adresu e-mail)

TABLICA KANBAN	FORMULARZ NEUMORPHISM	BLOG Z HEADLESS CMS
• projekt wykorzystuje bibliotekę React.js • dane wprowadzane przez użytkownika zapisywane są w **Local Storage**, dzięki czemu nie tracimy ich po zakończeniu sesji • poszczególne zadania możemy przenosić między kolumnami • jest to okazja do zapoznania się z **tablicą kanban** często wykorzystywaną do pracy nad projektami	• projekt wykorzystuje bibliotekę React.js i Styled Components • styl wykonania formularza i poszczególne jego elementy są z góry narzucone (trzeba więc **dostosować się do wymagań** jak w klasycznym projekcie) • formularz jest **walidowany** po stronie front endu, czyli dane wprowadzane przez użytkownika są weryfikowane przed ewentualnym wysłaniem ich do bazy danych	• projekt wykorzystuje bibliotekę React.js, React Router DOM oraz headless CSM prismic.io • prismic.io umożliwia wprowadzanie i przechowywanie contentu strony, dzięki czemu niepotrzebna jest **baza danych** • dane pobieramy z prismic.io przez **API** • React Router DOM pozwala na swobodną nawigację po stronie oraz renderowanie zawartości zależnie od **adresu URL**

* ten zakres materiału jest realizowany jedynie w trakcie mentoringu