Kurs JavaScript

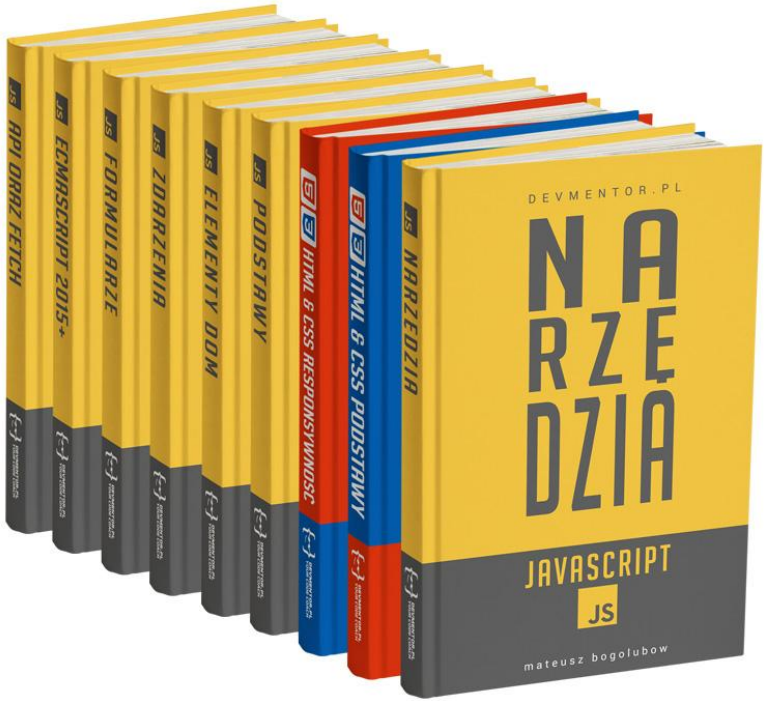
DEMO

W niniejszym materiale znajdziesz wybrane rozdziały z niektórych ebook-ów, które wchodzą w skład kursu JavaScript.

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

Każdy temat oraz ebook to jedna całość, która pozwala wprowadzić czytelnika w tajniki języka JavaScript.

Miłej lektury! - *Mateusz Bogolubow*



#04 Linia poleceń (terminal)

01. Visual Studio Code (IDE)

Wiersz poleceń (w Windows tzw. *cmd*) lub w systemach unix-owych określany jako *terminal* to pewien rodzaj tekstowego interfejsu.

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

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

Proponuję zapoznać się z podstawowymi komendami w systemie windows jak również systemie unix-owym.

Programiści często korzystają z terminala w celu uruchomienia narzędzi, które usprawnią ich pracę np. Webpack-a lub zainstalują przydatne biblioteki np. npm.

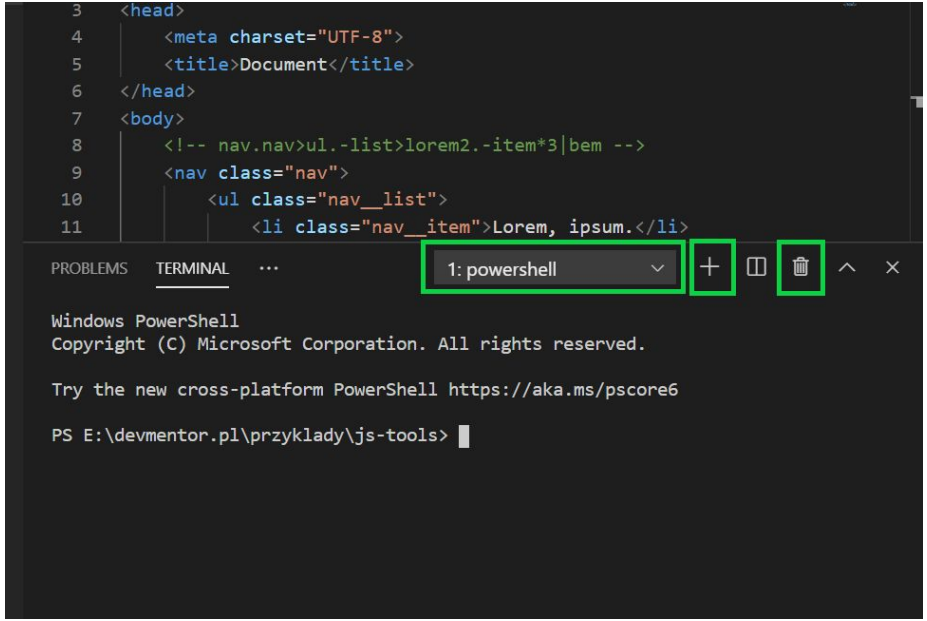
Będziemy te elementy jeszcze omawiać w tym materiale oraz w następnych.

Na chwilę obecną interesuje nas terminal, który jest dostępny w naszym *IDE*.

Aby go uruchomić wystarczy użyć skrótu ctrl+shift+` (backtick).

Można uznać, że ma on tą samą funkcjonalność co ten natywny jednak dużo wygodniej jest korzystać z niego w naszym IDE.

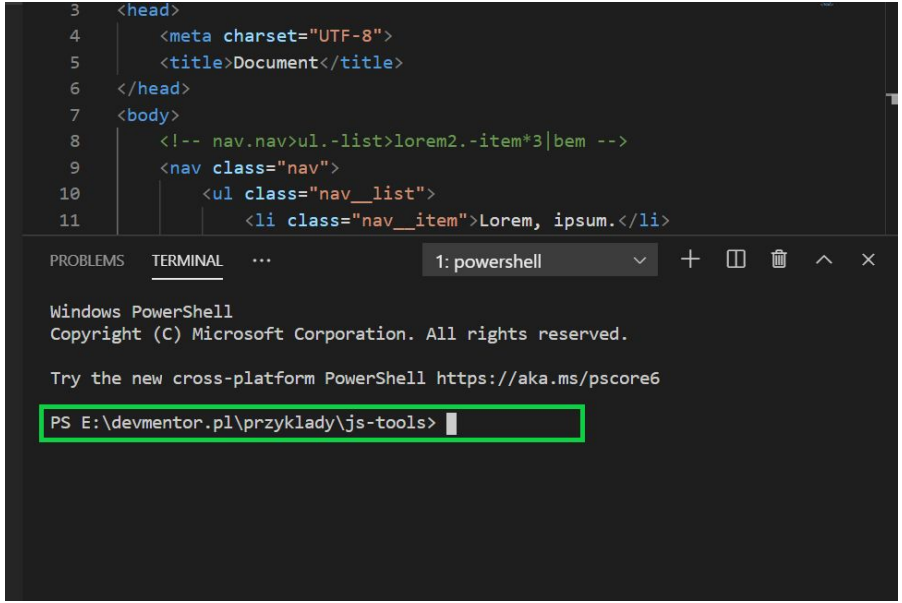
Jednocześnie możemy uruchomić kilka terminali (ikona plusa), zamknąć dowolny (ikona śmietnika) lub wybrać aktywny.



Korzystając 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 to musimy użyć komendy cd, po której następuje wprowadzenie nowej lokalizacji i zatwierdzeniem tej komendy klawiszem Enter.

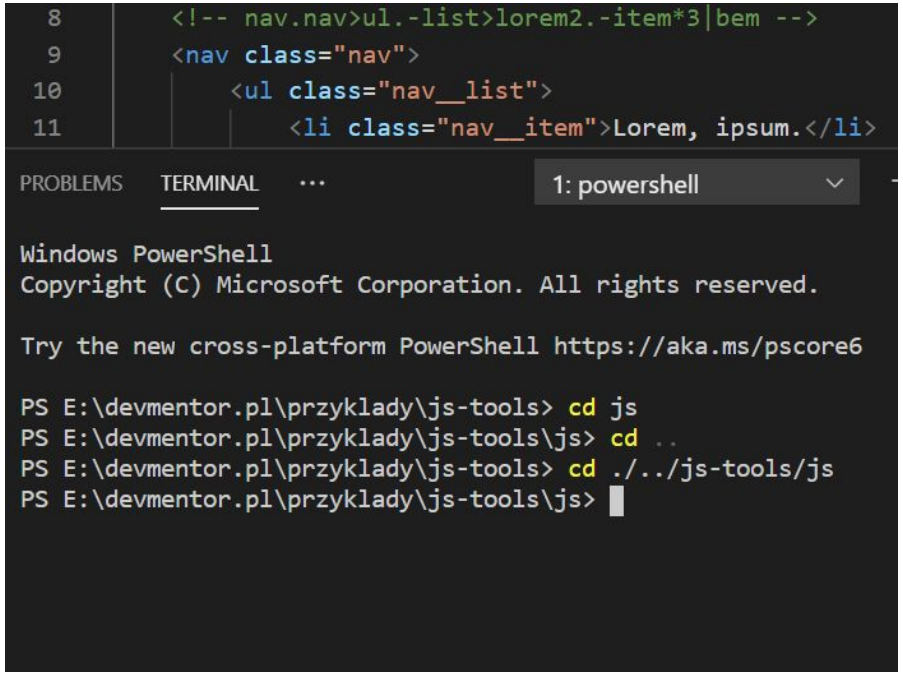


Dla przykładu, jeśli chcemy się przenieść do katalogu `js` to powinniśmy wpisać: `cd js`.

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

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

Jednak kropka oznacza odniesienie względem obecnej lokalizacji.



#03 Renderowanie znaczników

01. HTML

Poznaliśmy już kilkanaście różnych znaczników. Każdy z nich to po prostu tekst, który ma specjalną formę.

Jak uruchomimy plik w przeglądarce, który zawiera listę znaczników w odpowiedniej strukturze to zobaczymy całkiem schludny widok. Dlaczego?

Ponieważ przeglądarka domyślnie ma ustawione style (czyli cechy wyglądu) poszczególnych elementów.

Wspomniane cechy to po prostu określenie jak dany element ma wyglądać. Oczywiście możemy nadpisać te wartości, ale o tym będziemy mówić podczas rozdziału z CSS.

Teraz zajmiemy się jak to wygląda w przypadku wartości domyślnych czyli bez dołączenia do strony stylów CSS.

PS. Każda przeglądarka może posiadać inne domyślne wartości CSS, ale to tym też będziemy wspominać później.

Jeśli uruchomisz poniższy kod w przeglądarce to zobaczysz, że tekst w elemencie `<p>` jest wyświetlany jeden pod drugim.

Natomiast w przypadku znaczników `<span>` tekst jest wyświetlany jeden obok drugiego.

Dzieje się tak ponieważ domyślnie element `<p>` to element blokowy, natomiast `<span>` to element liniowy.

```
<section>
    <h2>Elementy blokowe</h2>
    <p>Lorem, ipsum dolor.</p>
    <p>Lorem ipsum dolor sit.</p>
</section>

<section>
    <h2>Elementy liniowe</h2>
    <span>Lorem, ipsum dolor.</span>
    <span>Lorem ipsum dolor sit.</span>
</section>
```

Elementy blokowe (block) zajmują całą szerokość dostępnej przestrzeni dlatego dwa elementy blokowe są wyświetlane jeden pod drugim, bo w tym samym wierszu nie ma już miejsca.

Elementy liniowe (inline) zajmują tyle przestrzeni ile ich zawartość. W naszym przykładzie to zawartość tekstu.

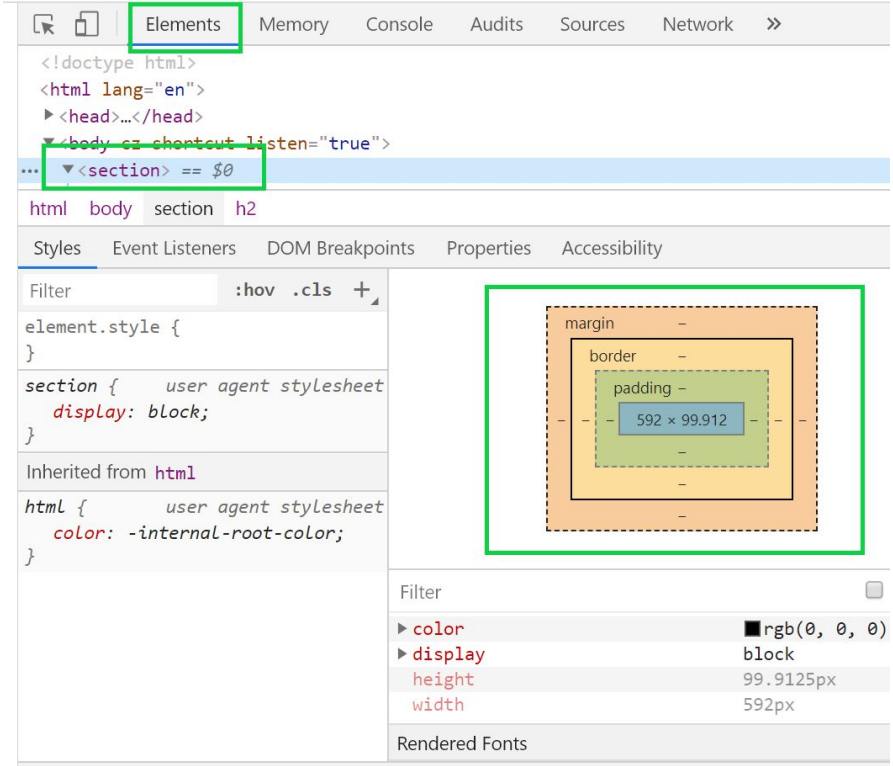
Tutaj warto zwrócić uwagę, że elementom liniowym nie da się ustawić wysokości, ani szerokości w CSS.

```
<section>
    <h2>Elementy blokowe</h2>
    <p>Lorem, ipsum dolor.</p>
    <p>Lorem ipsum dolor sit.</p>
</section>

<section>
    <h2>Elementy liniowe</h2>
    <span>Lorem, ipsum dolor.</span>
    <span>Lorem ipsum dolor sit.</span>
</section>
```

Jeśli spojrzymy w DevTools (skrót F12) do zakładki Element i klikniemy na któryś ze znaczników to będziemy mogli zobaczyć kolorowe pudełko, które prezentuję obok.

Przedstawia ono wymiary danego elementu. Każde ze znaczników posiada taką reprezentację niezależnie od tego jakiego typu jest to element (blokowy, liniowy itp.).

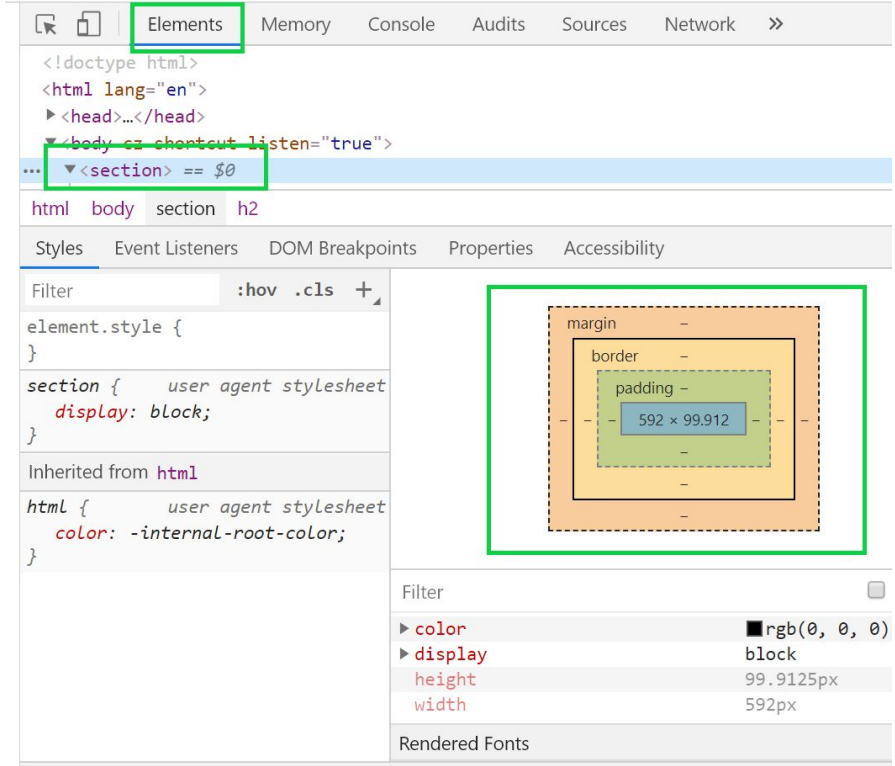


Niebieskie tło to zawartość danego znacznika. Przeglądarka w tym miejscu wyświetla rozmiar tego elementu.

W przedstawionym przykładzie to 592 piksele szerokości oraz 99.912 piksele wysokości.

Zielone pole to odstęp wewnętrzny (padding) czyli taki, który występuje między zawartością znacznika, a jej obramowaniem.

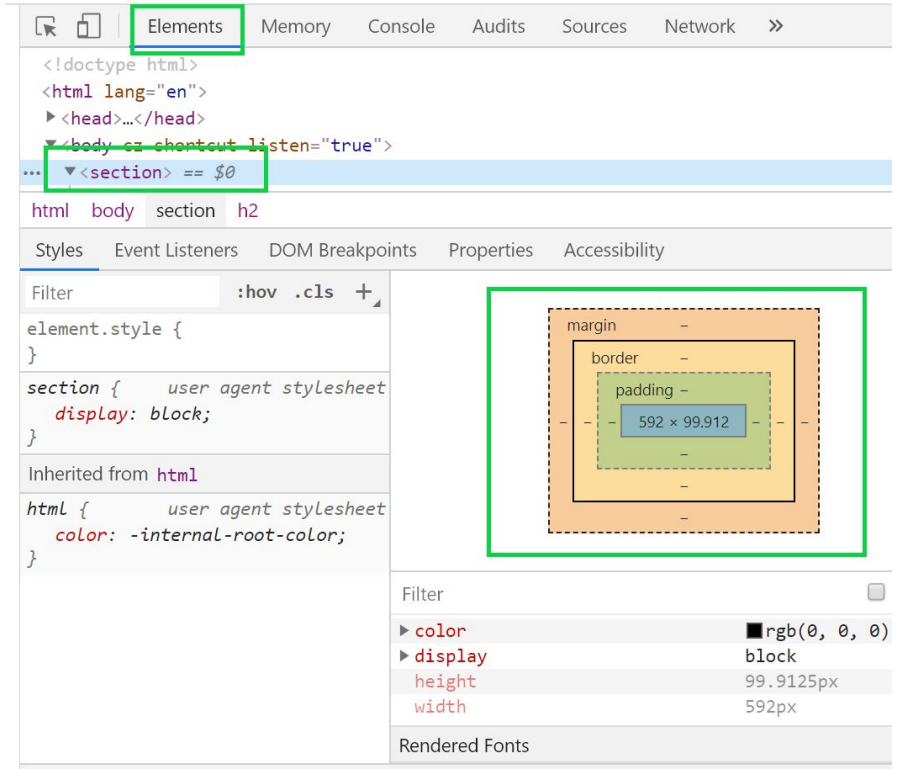
W obecnym przykładzie brak (-) takiego odstępu.



Następnie mamy border tj. obramowanie elementu, które nie występuje, ale z poziomu CSS możemy go modyfikować poprzez grubość, kolor, a nawet styl linii, ale o tym później.

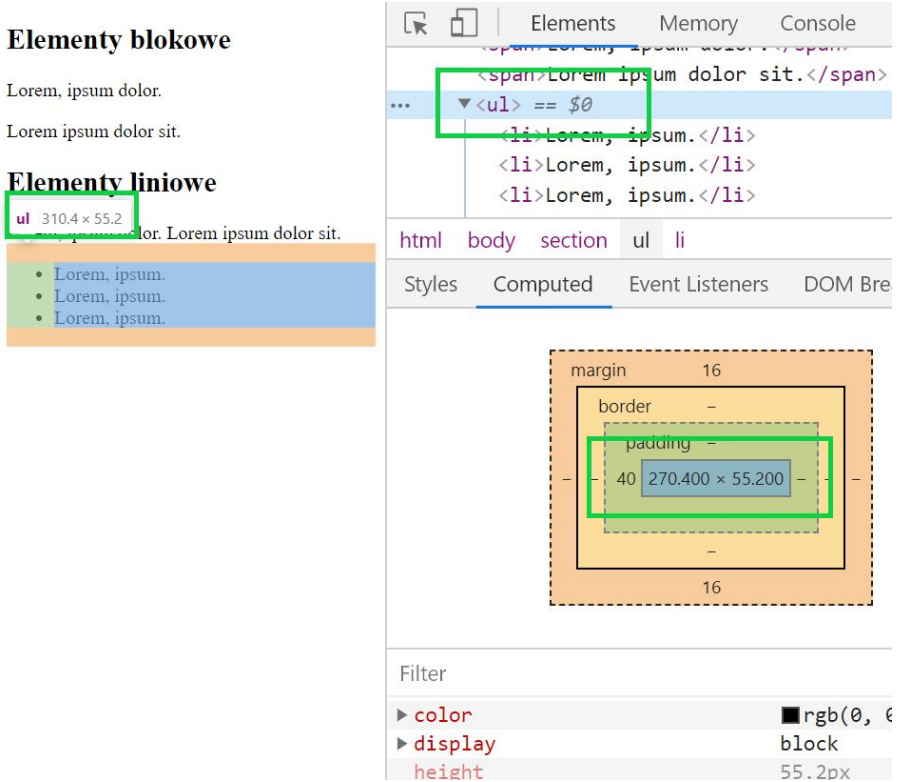
Na końcu mamy odstęp zewnętrzny, który pozwala nam odsuwać elementy od siebie.

Tutaj chciałbym zaznaczyć, że margin, nie jest sumowy dla sąsiednich elementów, a brana jest pod uwagę większa wartość (jeszcze o tym wspomnę).



Skoro jesteśmy już przy wymiarach to domyślnie przeglądarka renderując element określa jego wymiary tj. wysokość i szerokość sumując zawartość elementu, jego odstęp wewnętrzny (padding) oraz obramowanie (border).

Przykład obok prezentuje, że dla elementu <ul> jego zawartość zajmuje *270.4px*, ale posiada jeszcze padding o wartości *40px*, który ostatecznie rozszerza całość do wartości *310.4px* (*270.4+40*).



Nie tylko składowe rozmiaru tj. (wysokość, szerokość, padding, border) są definiowane przez wartości domyślne.

Przykład obok prezentuje znów sekcję, z nagłówkiem, tekstem, listą której jeden z elementów zawiera link tj. znacznik <a>.

Jak będzie to wyglądać w przeglądarce?

```
<section>
    <h2>Elementy blokowe</h2>
    <p>Lorem, ipsum dolor.</p>
    <p>Lorem ipsum dolor sit.</p>
    <ul>
        <li><a href="#">Lorem, ipsum.</a></li>
        <li>Lorem, ipsum.</li>
        <li>Lorem, ipsum.</li>
    </ul>
</section>
```

Mniej więcej tak jak obrazie obok....

Nagłówek jest większy oraz pogrubiony. Posiada również odstęp zewnętrzny tj. `margin` - podobnie jak paragraf.

Natomiast lista zawiera wypunktowanie, a link (`<a>`) w pierwszym elemencie listy (`<li>`) jest niebieski oraz posiada podkreślenie.

Tak przeglądarka domyślnie oznacza odnośniki.

05. Responsywne menu

Menu, które jest responsywne uznajemy za takie, kiedy w łatwy i wygodny sposób możemy przejść do interesującej jest podstrony.

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

Często wspomniane elementy pojawiają się dopiero po kliknięciu w hamburger-a.

Natomiast dla większych rozdzielczości będą to po prostu elementy wyświetlone poziomo - jeden obok drugiego.

Tutaj nasze menu jest widoczne od razu po załadowaniu strony.

Zacznijmy najpierw od struktury HTML...

Zanim przejdziemy do struktury menu zajmijmy się podstawową strukturą HTML.

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

Przyjąłem, że już na wersji tabletowej będę chciał mieć 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>
```

Sama struktura menu będzie wyglądać jak na przykładzie obok.

Całość umieszczam w znaczniku <header>, który zawiera logo w formie tekstu.

Samą nawigację umieszczam zgodnie z z przeznaczneiem w <nav>, a jego strukturę opieram o listę nieuporządkowaną tj. + .

```
<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 resetuję podstawowe ustawienia, aby łatwiej mi było wykonywać dalsze prace.

Najlepszym rozwiązaniem, byłoby wykorzystanie gotowego 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 musiałem robić.

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

Dodatkowo dla elementów `<li>` dodałem 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 chcę ustawić elementy obok siebie dlatego użyłem flex-a.

Najpierw ustawiam go dla `<header>`, aby rozdzielić logo tj. `<h1>` oraz `<nav>`.

Dodatkowo używam `justify-content`, aby przesunąć je do boków oraz przy pomocy `align-items` wyśrodkowuje w pionie.

Flex-a używam również dla ustawienia elementów menu.

```css
/* ./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 hamburger-a.

My będziemy mieli po prostu napis *menu*, ale i tak trzeba to jakoś zrobić.

Zanim do tego przejdziemy dopiszmy do naszej struktury ten napis, używając element `<label>` - zaraz pokażę 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 dopiero po kliknięciu je pokazać.

Wykorzystując JavaScript byłoby to bardzo proste, ale my na razie go nie znamy i wykorzystamy inne rozwiązanie.

Użyjemy tutaj pola `<input>` typu `checkbox`, który możemy zaznaczać i odznaczać.

Jeśli ten element będzie zaznaczony to pokażemy menu, jeśli nie to go ukryjemy.

Na początku osadzę element `<input>` i powiążę z `<label>` dzięki `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 ukrywamy całą nawigację dzięki `display: none`.

Teraz wykorzystując pseudo klasę `:checked` oraz selektor najbliższego rodzeństwa (`+`) możemy pokazać nasze menu tj. `display: block`.

Zwróć uwagę, że teraz nasze menu jest widoczne po zaznaczeniu checkbox-a.

Ten element można zaznaczyć również przez kliknięcie w `<label>`!

```css
/* ./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 ok.

Teraz musimy sprawdzić jak to wygląda dla 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 tutaj jest całkowicie niepotrzebny.

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

PS. Cały kod możesz podejrzeć w repozytorium, natomiast działanie możesz sprawdzić 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;
}
```

#01 Wywoływanie zdarzeń

04. Rozszerzenie

Aby wywołać even z poziomu kodu wystarczy wywołać metodą na elemencie o nazwie eventu np. `.click()`.

Dzięki właściwości `.isTrusted` możemy sprawdzić czy został uruchomiony przez użytkownika czy z poziomu kodu JS.

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

```js
const btnElement = document.querySelector('button');
const handleClick = function(e) {
    console.log('button was clicked');
    console.log(e.isTrusted);
    // zwraca [true] jeśli uruchomione przez użytkownika
    // zwraca [false] jeśli z poziomu kodu JS
}

btnElement.addEventListener('click', handleClick);
btnElement.click();
```

Przedstawione wcześniej rozwiązanie nie pozwala nam definiować dodatkowych opcji dlatego zdecydowanie bardziej polecam użycie .dispatchEvent().

To metoda, która wywołuje przekazany przez parametr *event* wraz z ustawieniami, o których dowiesz się na następnym slajdzie.

Do utworzenia zdarzenia możemy użyć obiektu Event, co prezentuje przykład obok.

```
const btnElement = document.querySelector('button');
const handleClick = function(e) {
    console.log('button was clicked');
    console.log(e.isTrusted);
    // zwraca [true] jeśli uruchomione przez użytkownika
    // zwraca [false] jeśli z poziomu kodu JS
}

btnElement.addEventListener('click', handleClick);

const eventClick = new Event('click');
// tworzę event typu [click]

btnElement.dispatchEvent(eventClick);
// wywołuję event na elemencie [btnElement]
```

Obiekt Event może przyjmować jako drugi parametr dodatkowe ustawienia

- *bubbles*
- *cancelable*

To dzięki nim możemy określać zachowanie naszego zdarzenia.

```
const btnElement = document.querySelector('button');
const handleClick = function(e) {
    console.log('button was clicked');
    console.log(e.isTrusted);
    // pobieram przekazane dane przy wywołaniu
}
btnElement.addEventListener('click', handleClick);
const eventClick = new Event('click', {
    'bubbles': true,
    // czy wykorzystujemy fazę bubbling przy propagacji
    'cancelable': true,
    // czy można zatrzymać event
    // za pomocą .preventDefault()
});
btnElement.dispatchEvent(eventClick);
```

Powinniśmy mieć również świadomość, że każdy event przynależy do grupy zdarzeń, które mają swoje szczególne cechy, np:

- *Event*
- *MouseEvent*
- *KeyboardEvent*
- *DragEvent*
- ...

Dlatego lepszym rozwiązaniem będzie określenie naszego event-u z przykładu jako MouseEvent.

```
const btnElement = document.querySelector('button');
const handleClick = function(e) {
    console.log('button was clicked');
    console.log(e.isTrusted);
    // pobieram przekazane dane przy wywołaniu
}
btnElement.addEventListener('click', handleClick);
const eventClick = new MouseEvent('click', {
    'bubbles': true,
    // czy wykorzystujemy fazę bubbling przy propagacji
    'cancelable': true,
    // czy można zatrzymać event
    // za pomocą .preventDefault()
});
btnElement.dispatchEvent(eventClick);
```

#01 Wprowadzenie

04. JSON Server

JSON Server to rozwiązanie, które pozwala nam uruchomić *API* na naszym komputerze dzięki czemu będziemy mieli nad nim całkowitą kontrolę.

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

Aby to zrobić w pierwszej kolejności musisz zainstalować *Node.js* na swoim sprzęcie 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 instalowałeś Node-a to wystarczy sprawdzić czy Twoja wersja jest **nowsza lub równa wersji 12**.

Wystarczy, że w terminalu wpiszesz: `node -v`

Jeśli instalacja przebiegła pomyślnie to również powyższa komenda powinna wyświetlić Ci informacje o wersji.

Teraz możemy zabrać się już za instalację 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 globalnie w naszym systemie i możemy go uruchamiać z dowolnego miejsca w terminalu.

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

Teraz w dowolnej lokalizacji tworzymy katalog `json-server-helloworld`.

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

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

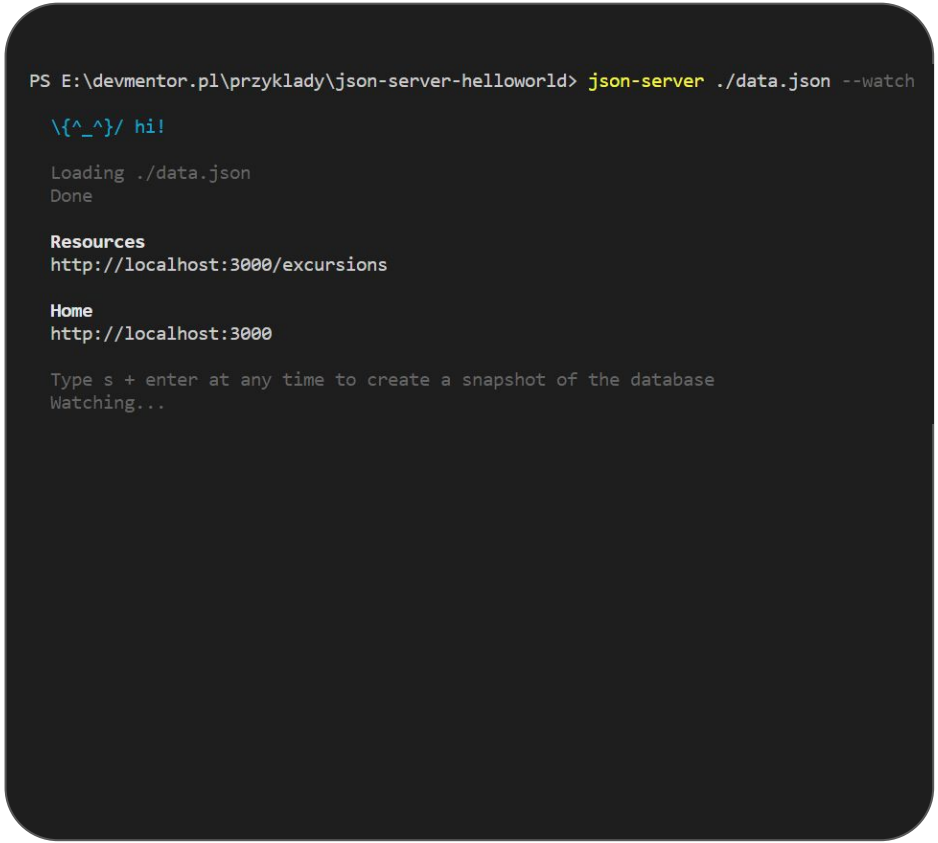
```
{
    "excursions": [
        {
            "id": 1,
            "name": "Zwiedzanie Krakowa",
            "price": 120
        },
        {
            "id": 2,
            "name": "Zwiedzanie Warszawy",
            "price": 120
        }
    ]
}
```

W konsoli powinieneś mieć napisane pod jakim adresem dostępne są zasoby.

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

U Ciebie za pewnie również jeśli wszystko wykonałeś poprawnie i nie masz zablokowanego portu *3000*, na którym nasłuchuje nasze *API*.

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



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

Jeśli zamkniesz terminal musisz ponownie go odpalić (tak jak wcześniej), aby *API* znów działało.

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

Jeśli będziesz potrzebować odpalić inną aplikację w terminalu np. *Webpack-a* to musisz uruchomić osobny terminal.

Jeśli używasz terminala z VS Code to wystarczy, że klikniesz plus (`+`) w prawym górnym rogu jego sekcji.