**12. Projekt indywidualny - tydzień 4 z 4**

**Wyzwania:**

Nadszedł ostatni tydzień Twojego pierwszego samodzielnego projektu. Pokazaliśmy Ci więcej praktycznych rzeczy związanych z inicjowaniem projektu, dobrymi praktykami RWD oraz budową formularzy.

W tym tygodniu uzupełnimy tę wiedzę o:

* budowę i przygotowanie modali (pop-upów)
* sposoby szukania i implementowania pluginów JavaScript na przykładzie wykresów
* sposoby budowania paczek i publikacji projektu

**12.1. Modale**

Modale/pop-upy pojawiają się niemalże na każdej stronie. Czasem mają funkcję użytkową (np. galeria, czat), kiedy nie chcemy, aby użytkownik opuszczał obecną stronę, a czasem funkcję skupiającą uwagę, kiedy chcemy pokazać ofertę specjalną.

**Założenia**

Po raz kolejny może się wydawać, że to nic trudnego — w końcu modal to nic innego, jak kontener na ciemnym tle z przyciskiem zamykającym w prawym górnym rogu. Oczywiście jest to prawda, ale nie całkowita, ponieważ musimy wziąć pod uwagę UX (User Experience), czyli dobre doświadczenia użytkowników z modalami. Będą one bazować na zachowaniach związanych z JavaScriptem oraz responsywnością modala.

Dlatego projektując modale musimy zadać sobie pytania o następujące kwestie:

* Responsywność modala
  + Jak modal ma się zachowywać na różnych szerokościach ekranu?
  + W jaki sposób tło powinno być wypozycjonowane względem całej strony?
  + W jaki sposób modal powinien być wypozycjonowany względem tła?
  + W jaki sposób modal ma się zachować, jeśli jego zawartość wykracza poza wysokość ekranu?
* Działania i kontrola modala
  + Jakie akcje powinny umożliwić zamknięcie modala?
    - kliknięcie w ikonę "X"
    - kliknięcie w tło
    - kliknięcie w przycisk "ESC" na klawiaturze

**Budowa HTMLa**

Zacznijmy od przygotowania kodu HTML. Zaczniemy od tzw. overlaya/backdropa, czyli tego ciemnego tła, które często towarzyszy modalom. Poza klasą dodamy mu ID, aby łatwiej go identyfikować w JavaScripcie.

<div class="overlay" id="overlay">

<!-- Tutaj będą nasze modale -->

</div>

Drugim krokiem będzie dodanie kodu modala. Zauważ, że dodaliśmy dodatkową klasę js--close-modal przyciskowi X (ktory znajdzie się w prawym górnym rogu), oraz przyciskom na końcu modala. Ta klasa w późniejszym etapie posłuży nam do identyfikacji akcji zamykania modala.

<div class="overlay" id="overlay">

<div class="modal" id="myModal">

<div class="btn-close js--close-modal">x</div>

<h4>Witaj w modalu!</h4>

<p>Lorem ipsum dolor sit amet...</p>

<button class="js--close-modal">Okay</button>

</div>

</div>

**Stylowanie**

Zaczniemy od upewnienia się, że wszystkie elementy posiadają box-sizing: border-box;

\* {

box-sizing: border-box;

}

Następnie przechodzimy do pozycjonowania overlaya. Dodajmy mu specjalnie wysoki z-index, aby zabezpieczyć się przed sytuacją, że jakiś element będzie wyżej niż modale.

.overlay {

position: fixed;

top: 0;

left: 0;

z-index: 9999;

}

Później uzupełniamy kod o wymiary oraz kolor wypełnienia tła.

.overlay {

position: fixed;

top: 0;

left: 0;

z-index: 999;

width: 100%;

height: 100%;

background-color: rgba(0,0,0,0.5);

}

Teraz przyszedł czas na odpowiedzenie sobie na kilka z wcześniej zadanych pytań:

* W jaki sposób modal powinien być wypozycjonowany względem tła?
* W jaki sposób modal ma zachować się, jeśli jego zawartość wykracza poza wysokość ekranu?

Chcemy, aby nasz modal pojawiał się ogólnie na środu strony (flexbox). Jednak jeśli jego zawartość wykroczy poza wysokość ekranu, to chcemy mieć możliwość scrollowania go (overflow: auto). Do tego damy mu przestrzeń (padding), aby modal nigdy nie stykał się z krawędzią przeglądarki.

Domyślnie nasz overlay będzie ukryty (display:none).

.overlay {

position: fixed;

top: 0;

left: 0;

z-index: 999;

width: 100%;

height: 100%;

background-color: rgba(0,0,0,0.5);

justify-content: center;

align-items: center;

overflow: auto;

padding: 30px;

display: none;

}

Utwórzymy teraz regułę, która pokaże nam overlaya, jeśli będzie posiadać on klasę show:

.overlay.show {

display: flex;

}

Opcjonalnie możemy ukryć scrollbar, pozostawiając funkcję scrollowania.

.overlay::webkit-scrollbar {

display: none;

}

Teraz możemy ostylować właściwy modal.

**UWAGA** margin musi mieć wartość auto, aby go nie obcinało, jeśli ekran byłby za niski na jego zawartość!

.modal {

max-width: 600px;

margin: auto;

background-color: #ffffff;

position: relative;

padding: 30px;

display: none;

}

Jemu również dodajemy klasę, która umożliwi jego wyświetlenie.

.modal.show {

display: block;

}

Dodajemy przycisk X do zamykania, który będzie znajdował się w prawym górnym rogu.

.modal .btn-close {

position: absolute;

top: 0;

right: 0;

padding: 5px 10px;

}

W ten sposób mamy gotowy i ostylowany modal.

**Podczepiamy JavaScript**

Czas na dodanie kontroli do naszego komponentu. Pisząc skrypty powinniśmy uwzględnić następujące założenia:

1. Możemy mieć wiele modali, więc otwieranie konkretnego powinno zabezpieczać nas przed wyświetlaniem innych
2. Wyłączanie modala powinno odbywać się na 3 sposoby:
   * wyłączanie na X
   * wyłączenie na overlay
   * wyłączanie na klawisz ESC

**Zamykanie**

Zaczniemy od napisania prostej funkcji, która zamknie modal poprzez usunięcie klasy showz overlaya.

function closeModal() {

document.getElementById('overlay').classList.remove('show')

}

Następnie podpinamy ją pod przyciski zamykające z klasą js--close-modal:

document.querySelectorAll('#overlay .js--close-modal').forEach(function(btn) {

btn.addEventListener('click', function(e) {

e.preventDefault()

closeModal()

})

})

Później dodamy możliwość zamknięcia modala po kliknięciu w tło overlaya.

document.querySelector('#overlay').addEventListener('click', function(e) {

if(e.target === this) {

closeModal()

}

})

A ostatnią metodą będzie zamykanie poprzez kliknięcie w przycisk Escape na klawiaturze.

document.addEventListener('keyup', function(e) {

if(e.keyCode === 27) {

closeModal()

}

})

**Otwieranie**

Otwieranie modala będzie polegać na zamknięciu wszystkich modali, które są wewnątrz overlay, a następnie otwarciu wskazanego modala oraz pokazania overlay.

function openModal(modal) {

document.querySelectorAll('#overlay > \*').forEach(function(modal) {

modal.classList.remove('show')

})

document.querySelector('#overlay').classList.add('show')

document.querySelector(modal).classList.add('show')

}

Powyższy kod możemy wywołać, kiedy tylko będziemy potrzebować pokazać konkretny modal, np. openModal('#myModal').

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**12.2. Tworzymy wykresy**

Kolejną rzeczą, którą możemy dodatkowo zaimplementować do projektu, będzie wtyczka, która umożliwi nam stworzenie interaktywnego wykresu. Wybór narzędzia nie jest wcale taki łatwy, ponieważ kiedy wpiszemy do wyszukiwarki js charts, to wyskoczą nam najróżniejsze rozwiązania. Dlatego potrzebujemy kilku kryteriów, aby móc przefiltrować dostępne biblioteki.

**Kryteria doboru biblioteki**

1. Responsywność — naturalną rzeczą jest to, że chcemy, żeby nasza strona była responsywna, więc wykres też taki powinien być
2. Typ wykresu — czy dana biblioteka obsługuje typ wykresu, który nas interesuje
3. Możliwości dostosowania — np. kolory, układy, podpisy osi itp.
4. Prostota użycia

Przeglądając listy bibliotek dokonujemy selekcji. W przypadku naszego projektu zdecydowaliśmy się na zastosowanie [Chart.js](http://www.chartjs.org/), które spełnia wszystkie podane przez nas kryteria, a dodatkowo w prosty sposób umożliwi nam zrobienie podobnego wykresu do tego, który był załączony w projekcie.

**Implementacja**

Zaczynamy od umieszczenia linku do biblioteki tuż przed naszym głównym plikiem z JavaScriptem.

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.4.0/Chart.min.js"></script>

Później należy dodać poniższy fragment w wybranym miejscu w kodzie HTML wykresu:

<canvas id="myChart"></canvas>

Kiedy popatrzymy sobie teraz naszą stronę, to naturalnie niczego nowego nie zauważymy, ponieważ wykres musi zostać zainicjoway i skonfigurowany.

Otwórz swój główny plik ze skryptami. Zaczniemy od wskazania elementu canvas z HTML i pobrania jego kontekstu (*context*), który umożliwi nam pracę na tym elemencie. Więcej na temat tego elementu oraz jego możliwości znajdziesz w [oficjalnej dokumentacji](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API).

var ctx = document.getElementById('myChart').getContext('2d');

Drugim etapem będzie skonfigurowanie wykresów zgodnie z dokumentacją.

var chart = new Chart(ctx, {

// 1

type: 'bar',

data: {

// 2

labels: ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10"],

// 3

datasets: [{

// 4

label: "Signups",

// 5

backgroundColor: '#8DBEC8',

borderColor: '#8DBEC8',

// 6

data: [ 52, 51, 41, 94, 26, 6, 72, 9, 21, 88 ],

},

{

label: "FTD",

backgroundColor: '#F29E4E',

borderColor: '#F29E4E',

data: [ 6, 72, 1, 0, 47, 11, 50, 44, 63, 76 ],

},

{

label: "Earned",

backgroundColor: '#71B374',

borderColor: '#71B374',

data: [ 59, 49, 68, 90, 67, 41, 13, 38, 48, 48 ],

// 7

hidden: true,

}]

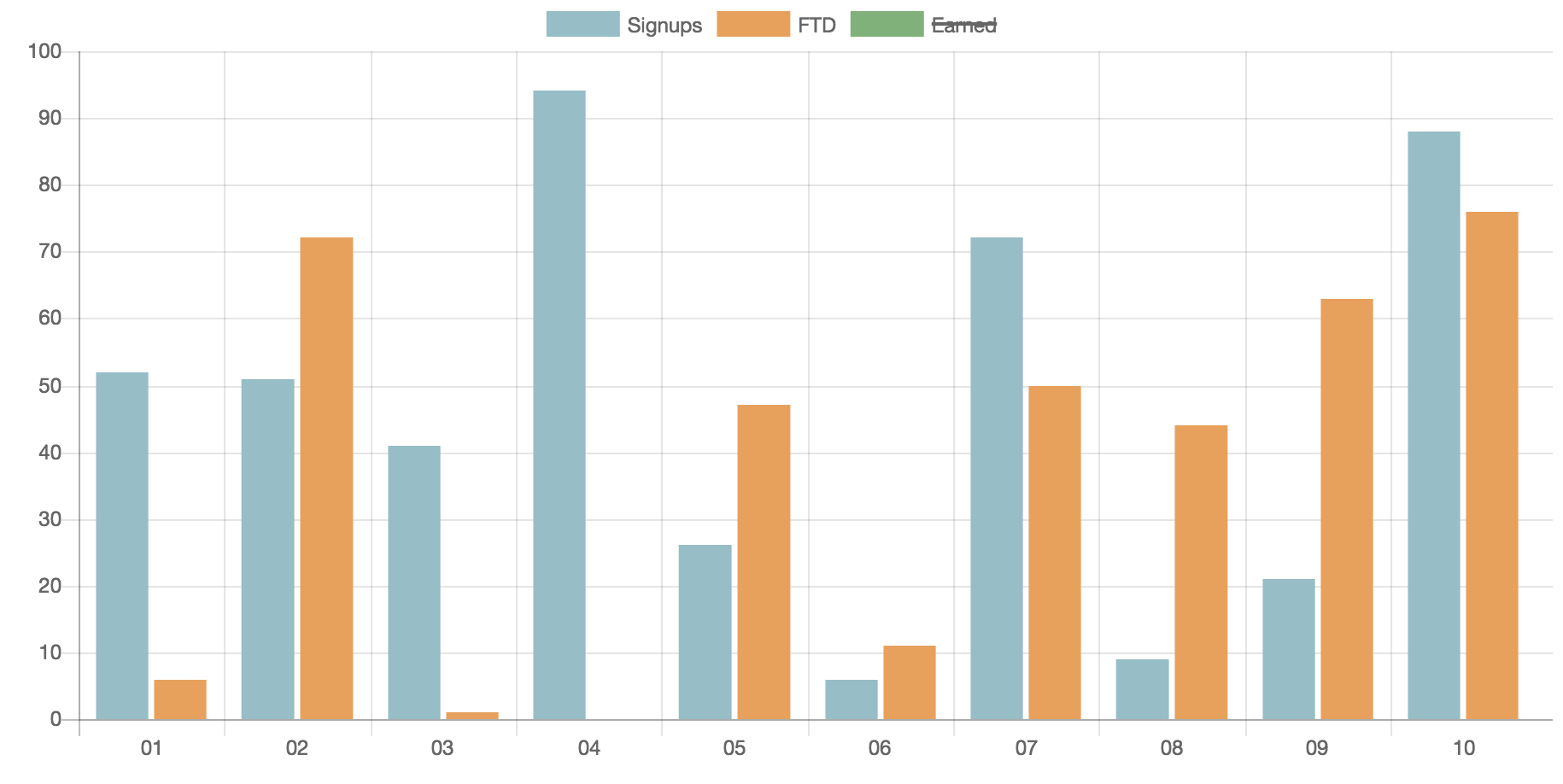
},

});

Zaczynamy od wybrania typu wykresu (**1**) — na naszym projekcie mamy typ barowy (kolumnowy). Następnie dodajemy etykiety dla osi X (**2**). Kolejnym krokiem jest ustawienie serii danych (**3**) poprzez nadanie im nazwy (**4**), wybór koloru serii (**5**) oraz uzupełnienie przykładowymi danymi (**6**).

Na naszym projekcie ostatnia seria jest ukryta, dlatego musimy uwzględnić to w konfiguracji (**7**).

Teraz możemy sprawdzić, jak finalnie wygląda nasz wykres, i w jakim stopniu pasuje do tego, który był załączony w projekcie.

[](https://uploads.kodilla.com/bootcamp/kodilla-bootcamp-24-07-2018-1532422718-8323.png)

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**12.3. Budujemy paczkę**

Wszystkie podstrony pocięte? Są responsywne i funkconalne? Świetnie! Teraz czas przeprowadzić ostatnie testy naszego projektu i będziemy mogli szykować się do zbudowania i opulikowania paczki.

Co należy sprawdzić przed publikacją strony?

* czy wszystkie podstrony i akcje są podlinkowane wzajemnie — przeklikaj możliwość przejścia na każdą podstronę,
* czy wszystkie aktywne elementy typu linki i buttony mają hovery,
* czy pliki .html nie zawierają błędów,
* czy pliki .css są zminimalizowane i skompresowane.

Pierwszy i drugi punkt musimy sprawdzić manualnie, ale w testowaniu dwóch pozostałych pomogą nam task runnery.

**Task runnery**

Podobnie jak na początku tego projektu, zasugerujemy Ci pewną strukturę zadań, lecz znalezienie odpowiednich paczek i konfigurację skryptów pozostawimy całkowicie Tobie.

Zacznijmy od stworzenia ogólnego skryptu build, który uruchomi nam wszystkie skrypty zawierające prefix build:.

Następnie stwórzmy zadanie, które sprawdzi nam poprawność plików HTML. Tego typu zadanie tworzyliśmy już w module poświęconym NPM, dlatego jeśli go nie pamiętasz, to wróć do niego.

Kolejnym skryptem będzie kompilacja SASS na skompresowany CSS bez sourcemap.

I ostatnim zadaniem będzie skrypt, który umożliwi nam zmniejszenie wagi obrazków poprzez kompresję.

Efektem końcowym ma być zadanie uruchamiane za pomocą npm run build, które sprawdzi poprawność plików oraz zbuduje kompletną paczkę gotową do publikacji.

**Publikacja na GitHub Pages**

Kiedy wszystko będzie już na GitHubie, a Twój główny plik HTML będzie miał nazwę index.html, to możemy przygotować naszą stronę do publikacji na GitHub Pages.

Jeśli nie pamiętasz, w jaki sposób to zrobić, to zachęcamy do powrotu do modułu, gdzie omawialiśmy sobie temat Gita.

**Zadania dla chętnych**

Dla chętnych mamy wyzwanie. GitHub Pages umożliwia nie tylko publikację głównego katalogu repozytorium, ale też poprzez publikację katalogu docs.

[Instrukcja do konfiguracji GitHub Pages](https://help.github.com/articles/configuring-a-publishing-source-for-github-pages/#publishing-your-github-pages-site-from-a-docs-folder-on-your-master-branch)

Twoim wyzwaniem będzie stworzenie takich zadań, aby uruchamiając komendę npm run build usuwał się obecny folder docs, tworzył się na nowo, a w nim zostały umieszczone tylko niezbędne pliki do uruchomienia naszej strony, czyli HTML, CSS, obrazki, czcionki, skrypty JavaScript. Wszystkie pliki mają być możliwie skompresowane (w tym HTML). Tak powstała paczka będzie całkowicie wyseparowane od tej, nad którą pracujemy.

**Zadanie: Przesłanie zadania do Mentora**

Czas na przesłanie finalnej wersji projektu do Mentora. Upewnij się, że repozytorium na GitHubie jest aktualne, strona jest opublikowana na GitHub Pages, a link do opublikowanej strony znajduje się w pliku README.md. Następnie, wyślij do Mentora link do repozytorium.

[**Podgląd zadania**](https://kodilla.com/pl/bootcamp-task/200/322200)

Początek formularza



 Wyślij link

Dół formularza

**12.4. Podsumowanie projektu**

W pierwszych tygodniach kursu pokazywaliśmy Ci, w jaki sposób działa HTML i CSS. Rozwijaliśmy te umiejętności robiąc kolejne, coraz bardziej rozbudowane przykłady. Wprowadziliśmy również pewne wzorce, takie jak gridy czy praca z flexboksem. Dodaliśmy do tego trochę narzędzi typu Git, Sass i NPM, aby pokazać Ci, z czym pracują na co dzień developerzy.

Przez ostatnie 4 tygodnie daliśmy Ci do zrobienia nieco większy i bardziej skomplikowany projekt, a co tydzień pokazywaliśmy Ci kolejne dobre praktyki zaczynając od samego inicjowania projektu, jego analizy, rozwijania, aż do publikacji. Wykorzystuj tego typu podejście w kolejnych projektach i zadaniach, a zobaczysz, że nie tylko praca stanie się łatwiejsza, ale również jej jakość zdecydowanie wzrośnie.