



UNIWERSYTET ŁÓDZKI
Wydział Fizyki i Informatyki Stosowanej UŁ
kierunek: Informatyka

Jarosław Janiak

Nr albumu: 368675

**Widgety BEM na strony internetowe dostępne dla
osób z niepełnosprawnością**

Praca inżynierska
napisana pod kierunkiem
dr Bartosza Zielińskiego

Katedra Informatyki
Wydział Fizyki i Informatyki Stosowanej
Uniwersytetu Łódzkiego

ŁÓDŹ 2020

SPIS TREŚCI:

1. Wprowadzenie
 2. Metodologia BEM
 3. Standard WAI-ARIA
 4. Szkic projektu
 5. Nawigacja/Menu
 6. Zwijane Sekcje (Collapsible Sections)
 7. Karty linków (Cards)
 8. Audyt strony
 9. Podsumowanie
- Bibliografia

1. Wprowadzenie

Poniższa praca powstała z myślą opracowania i zrealizowania kilku efektownych widgetów/komponentów możliwych do umieszczenia na stronach internetowych stworzonych ściśle według reguł metodologii BEM i zgodnie ze standardem WAI-ARIA. Same widgety stanowią przy tym użyteczne i funkcjonalne elementy strony internetowej (responsywne menu, składane bloki sekcji, kafelki/karty linki) łatwe w integracji także przy realizacji innych projektów przy jednoczesnym zachowaniu dobrej strony wizualnej (estetyka wraz z funkcjonalnością). W oparciu o te kryteria realizowana jest idea połączenia założeń metodologii BEM z atrybutami standardu WAI-ARIA.

Poza dokumentem HTML na projekt składają się pliki stylów (pliki CSS) oraz pliki skryptów (pliki JS - JavaScript) umożliwiające wdrożenie na stronie internetowej elementów, dzięki którym strona ta może, poza wyświetlaniem statycznych informacji, obsługiwać również zmianę treści odpowiednio do sytuacji/zdarzeń będących w przypadku tej pracy wynikiem odpowiedzi na aktywności użytkownika na stronie.

Zarówno pliki stylów jak i pliki skryptów poddano konkatencji (złączeniu) oraz w przypadku plików stylów minifikacji. W tym celu posłużono się mechanizmem GruntJS Task Runner (narzędzie do automatyzacji zadań). Wyniki tych operacji wdrożono bezpośrednio do projektu (zastępując mnogość plików jednym właściwym dla danej jego funkcji - styl/skrypt). Znajdują się one w podfolderze 'build'.

Struktura dokumentu HTML projektu to: blok nawigacji i blok - kontener osadzone w bloku głównym z sekcjami i podsekcjami a w nich widgety które same także stanowią bloki/elementy/komponenty w rozumieniu metodologii BEM wzbogacone o zbiór atrybutów poprawiających dostępność aplikacji (WAI-ARIA).

Część sekcji to elementy puste służące do zwiększenia objętości menu tak by uwypuklić efekt **ScrollSpy** będący cechą wzbogacającą nawigację w obrębie strony. Sam efekt opiera się na dynamicznej modyfikacji stylu elementów składowych menu będącej odpowiedzią na aktywne monitorowanie (obserwację) stanu położenia określonych elementów strony inicjowanego za pomocą skryptów dołączonych do projektu.

2. Metodologia BEM

BEM (ang. **Block, Element, Modifier**) to oparte na komponentach podejście do tworzenia stron internetowych. Zostało ono przygotowane przez firmę Yandex w celu opracowania witryn, które powinny być uruchamiane szybko i obsługiwane przez długi czas. Ideą jest podzielenie interfejsu użytkownika a przez to wewnętrznej struktury projektu na niezależne bloki. To sprawia, że budowa interfejsu jest łatwa i szybka nawet przy jego wysokim złożeniu i pozwala na tworzenie rozszerzalnych i wielokrotnego użytku jego elementów.

W rozwoju sieci końcowy produkt (taki jak strona internetowa) składa się z różnych technologii (HTML, CSS, JS itd.). W BEM praca ze wszystkimi tymi technologiami wykorzystuje tę samą terminologię i podejście do wdrażania. Oznacza to, że cały zespół projektu BEM otrzymuje zuniifikowany język komunikacji, działający pod względem bloków, elementów i modyfikatorów (jak chociażby opis znaczników strony internetowej).

BEM nie używa selektorów znaczników i identyfikatorów. Style bloków i elementów są definiowane przez selektory klas. Dostęp do selektora klasy można uzyskać poprzez atrybut `class`, który należy zdefiniować dla każdego elementu HTML. Wartością atrybutu `class` może być lista słów oddzielona spacjami. Pozwala to na użycie różnych encji BEM w tym samym węźle DOM.

Metodologia BEM nie zaleca łączenia znaczników i klas w selektorze. Połączenie tagu i klasy (na przykład `button.button`) powoduje, że reguły CSS są bardziej szczegółowe, co utrudnia ich przesłonięcie. Prowadzi to do konfliktów priorytetów, w których arkusze stylów są ładowane przez zbyt skomplikowane selektory.

Używając BEM, możemy tworzyć znaczniki HTML ręcznie lub generować je automatycznie. W obu przypadkach kod HTML działa zgodnie z tymi samymi zasadami.

Znaczniki strony są opisane w kategoriach bloków, elementów i modyfikatorów.

Aby określić, że blok, element lub modyfikator znajduje się w węźle DOM, jego nazwa jest zapisywana w atrybucie `class`.

W najprostszym przypadku pojedynczy węzeł DOM odpowiada pojedynczemu blokowi.

Główne zasady BEM możemy stosować także do organizacji i przechowywania kodu w JavaScript: logika każdego bloku oraz jego opcjonalne elementy i modyfikatory są opisane w osobnych plikach (kod dzielony jest w ten sposób na osobne powiązane znaczeniowo z strukturą BEM projektu części).

Pliki JavaScript dla każdego komponentu są przechowywane zgodnie z zasadami organizacji struktury plików projektu BEM.

2.1 Blok

Główny kontener (pojemnik) zawierający część kodu. Charakteryzuje się niezależnością, która sprawia, że nasz pojemnik zawsze będzie wyświetlany w taki sam sposób. Każdy blok pisany jest pod wielokrotne użycie, co znacząco ułatwia pracę przy jego późniejszym wykorzystaniu czy to w danym projekcie czy przyszłych projektach.

Jego nazwa powinna być unikatowa i jednoznacznie wskazywać na przeznaczenie. Nazwa bloku opisuje jego cel („co to jest?” - menu lub przycisk), a nie jego stan („jak to wygląda?” - czerwony lub duży). Np.:

```
<main class="main" role="main">
.
.
.
</main>
```

Całą stronę internetową można umieścić w jednym bloku i dodawać w niej kolejne bloki odpowiedzialne za np. panel boczny, stopkę, nawigację. Bloki można zagnieżdżać w sobie, przy tym liczba poziomów zagnieżdżenia jest nieograniczona:

```
<main class="main" role="main">

    <nav class="section--nav menu__container">...</nav>

    <div class="main__container">...</div>

</main>
```

(w tym przykładzie elementy menu są pokazane jako linki. Ta struktura bloków jest implementowana za pomocą zagnieżdżonych elementów)

```
<ul class="menu">
    <li class="menu__item">
        <a class="menu__link" href="https://"> ... </a>
    </li>
</ul>
```

Blok nie powinien wpływać na środowisko/otoczenie w obrębie którego się znajduje, co oznacza, że nie należy ustawiać jego zewnętrznej geometrii (np. marginesu) ani pozycjonowania (zapewnia to niezbędną niezależność przy jego ponownym wykorzystywaniu lub przenoszeniu z miejsca na miejsce). I jest to raczej zalecenie niż odgórny nakaz wobec czego nic nie stoi na przeszkodzie aby stworzyć wyjątek od tej reguły.

Aby określić, że blok, element lub modyfikator znajduje się w węźle DOM, jego nazwa jest zapisywana w atrybucie `class`. W najprostszym przypadku pojedynczy węzeł DOM odpowiada pojedynczemu blokowi.

Aby połączyć style i zachowanie kilku jednostek BEM należy umieścić je w tym samym węźle DOM. Aby to zrobić, należy dodać nazwy jednostek BEM oddzielone spacjami do atrybutu `class`. To podejście z kolei nazywa się miks'em (mieszaniem):

```
<nav class="section--nav menu__container">...</nav>
```

Na przykład możemy użyć miksów, aby dodać modyfikator (patrz sekcja niżej) do bloku lub elementu.

Te same bloki mogą być używane w różnych projektach. Aby zapobiec identyczności tych projektów, bloki można modyfikować za pomocą:

- modyfikatorów
- miksów
- poziomu redefinicji
- kontekstu

2.2 Element

Złożona część bloku, której nie można używać oddzielnie. Jest ona jego nieodłączną częścią, która znajduje się tylko w wybranym bloku i wchodzi w jego skład. Określa przeznaczenie poszczególnych elementów umieszczonych w pojemniku który sama także stanowi.

Element jest zawsze częścią bloku i nie należy go używać oddzielnie od bloku.

Jeśli część kodu może zostać ponownie użyta i nie zależy to od implementacji innych składników strony należy utworzyć blok.

Jeśli sekcja kodu nie może być użyta osobno bez jednostki nadrzędnej (bloku). Utwórz element.

Wyjątkiem są elementy, które należy podzielić na mniejsze części - podelementy - w celu uproszczenia rozwoju. W metodologii BEM nie można tworzyć elementów elementów. W takim przypadku zamiast tworzyć element, należy utworzyć blok serwisowy.

Nazwa elementu opisuje jego cel („co to jest?” - element, tekst itp.), A nie jego stan („jaki typ lub jak on wygląda?” - czerwony, duży itp.).

Struktura pełnej nazwy elementu to:

```
block-name__element-name
```

Nazwa elementu jest oddzielona od nazwy bloku podwójnym podkreśleniem (__).

Elementy tak samo jak bloki można zagnieżdżać w sobie i podobnie jak w przypadku bloków liczba poziomów zagnieżdżenia może być dowolna.

Element jest zawsze częścią bloku, a nie innym elementem. Oznacza to, że nazwy elementów nie mogą zdefiniować hierarchii, takiej jak:

```
block-name__elem1__elem2.
```

Nazwa bloku określa przestrzeń nazw, która gwarantuje, że elementy są zależne od bloku (block__elem).

2.3 Modyfikator

Element, który określa wygląd, stan lub zachowanie bloku lub elementu.

Nazwa `modyfikatora` opisuje jej wygląd („Jaki rozmiar?” Lub „Który motyw?” ltd. - `size_s` lub `theme_islands`), jego stan („Czym różni się od innych?” - wyłączony, skupiony itp.) Oraz jego zachowanie („Jak się zachowuje?” lub „Jak reaguje na użytkownika?” - np. `kierunki_left-top`).

Nazwa modyfikatora jest oddzielona od nazwy bloku lub elementu pojedynczym podkreśleniem (`_`).

Boolean

Używany, gdy ważna jest tylko obecność lub brak modyfikatora, a jego wartość nie ma znaczenia. Na przykład `wyłączone`. Jeśli obecny jest modyfikator boolowski, jego wartość przyjmuje się za prawdę.

Struktura pełnej nazwy modyfikatora jest zgodna ze wzorem:

```
block-name_modifier-name
```

```
block-name__element-name_modifier-name
```

Kluczowa wartość (Key-value)

Używane, gdy ważna jest wartość modyfikatora.

Struktura pełnej nazwy modyfikatora jest zgodna ze wzorem:

```
block-name_modifier-name_modifier-value
```

```
block-name__element-name_modifier-name_modifier-value
```

Modyfikatora nie można używać samodzielnie!

Z perspektywy BEM modyfikatora nie można używać w oderwaniu od zmodyfikowanego bloku lub elementu. Modyfikator powinien zmienić wygląd, zachowanie lub stan obiektu, a nie go zastąpić.

3. Standard WAI-ARIA

WAI-ARIA (**Web Accessibility Initiative - Accessible Rich Internet Applications**) jest dokumentacją stworzoną przez [Web Accessibility Initiative \(WAI\)](#) z inicjatywy [W3C \(The World Wide Web Consortium\)](#). To technologia dzięki której możliwe jest udostępnienie technologiom asystującym AT (ang. assistive technologies) brakujących informacji na temat bardziej skomplikowanych komponentów witryny. Dzięki możliwości opisanie ich roli w aplikacji nawigacja poza wartością serwisu staje się możliwa dla osób z niepełnosprawnościami.

ARIA jest relatywnie młodą specyfikacją ale jest coraz lepiej wspierana. Posiada wsparcie głównych przeglądarek, technologii wspomagających, narzędzi JavaScript oraz aplikacji. Ponieważ część użytkowników może nadal używać starszych wersji tych narzędzi czy aplikacji. W takim przypadku można rozważyć implementację ARIA ulepszając swoją aplikację progresywnie, na przykład przy użyciu JavaScript, zamiast znaczników HTML w celu lepszego wsparcia dla użytkowników wciąż korzystających ze starych technologii.

ARIA dostarcza semantykę opisującą role (ang. roles), statusy (ang. states) oraz funkcjonalności dla wielu komponentów stron internetowych takich jak: nawigacje (menu), slajdery (ang. sliders), okna dialogowe czy drzewa rozwijane (częściej występujące w aplikacjach desktopowych). Dostarcza również informacje strukturalne, które pomagają twórcom identyfikować odpowiednie komponenty. ARIA pozwala dynamicznym aplikacjom internetowym opartym o JavaScript na dopasowanie do technologii pochodzenia desktopowego.

Tym samym ARIA definiują specjalne role obiektów a dodatkowo dostarczają inne przydatne opcje, takie jak długość pasku postępu, mogą zostać użyte w celu aktywacji bądź dezaktywacji obiektu, stanowić punkt informacyjny (np. navigation) czy opisać szereg właściwości, które mogą ulec zmianie (np. aria-autocomplete).

ARIA jest przeznaczona tylko dla API (ang. application programming interface) technologii pomocniczych/asystujących AT i nie ma wpływu na DOM czy style. ARIA jest tylko dodawanym do elementu HTML atrybutem, który ma wspomóc technologie pomocnicze AT API (assistive technologies API). Mimo, że sama ARIA nie zmienia stylów to w połączeniu z atrybutem HTML możemy odpowiednio wystylizować elementy ARIA w CSS:

```
.tab-panel[aria-hidden="true"] { display: none; }  
  
.tab-panel[aria-hidden="false"] { display: block; }
```

Atrybuty ARIA umieszczone wewnątrz tagu HTML mogą być dodawane i aktualizowane w JavaScript.

ARIA oferuje trzy elementy ułatwiające komunikację przeglądarki z czytnikiem ekranu: **role, właściwości i statusy**.

ROLE (ROLES)

Role informują aplikacje asystujące, że element, którego dotyczą, jest w rzeczywistości czymś innym, niż wynika z jego znacznika. W przypadku większości znaczników HTML ich domyślna rola została już określona w dokumentacji tego języka. Zmiana przeznaczenia takiego elementu wynika często z manipulacji strukturą DOM przez skrypty JavaScript.

Role mogą również nadawać znaczenie, które w języku HTML nie istnieje, np. ostrzeżenie. Mogą również stanowić landmark – określać, jaką częścią dokumentu są, nie podając przeznaczenia elementu.

Role ARIA dzielą się na grupy:

- role struktury dokumentu (odnoszące się do struktury treści przedstawionej na stronie)
 - document
 - group
 - presentation
 - toolbar
- opisujące przeznaczenie poszczególnych elementów
 - alert
 - button
 - dialog
 - progressbar
 - scrollbar
 - slider
 - spinbutton
 - tab
 - tooltip
 - tree
- landamrki (punkty orientacyjne na witrynie, umożliwiające sprawniejszą nawigację)
 - application
 - banner
 - complementary
 - contentinfo
 - form
 - navigation
 - search
- role abstrakcyjne (nie powinny być używane w treści)
 - command
 - section
 - window

WŁAŚCIWOŚCI (PROPERTIES) I STATUSY (STATES)

Właściwości ARIA labels określają dodatkowe znaczenie danego elementu. Podczas gdy każdy z komponentów na stronie może posiadać wyłącznie jedną rolę, w przypadku stanów i właściwości to ograniczenie nie istnieje. Wszystkie atrybuty z tej grupy zaczynają się od przedrostka „aria-“. Stany te mogą ulec zmianie w odpowiedzi na działania użytkownika.

Statusy pozwalają określić **aktualny stan elementu**, często zmieniany dynamicznie za pomocą kodu JavaScript. Na podstawie ich zawartości możemy również modyfikować style CSS danego elementu.

Wśród atrybutów tego rodzaju możemy wymienić:

- aria-activedescendant
- aria-atomic
- aria-autocomplete
- aria-busy
- aria-checked
- aria-controls
- aria-describedby
- aria-disabled
- aria-dropeffect
- aria-expanded
- aria-flowto
- aria-grabbed
- aria-hidden
- aria-label
- aria-labelledby
- aria-live
- aria-owns
- aria-posinset
- aria-pressed
- aria-readonly
- aria-relevant
- aria-required
- aria-selected
- aria-setsize

Przed stworzeniem komponentu, do którego dodajemy atrybut ARIA, powinniśmy upewnić się, czy w semantyce języka HTML nie istnieje już element spełniający tę funkcję.

Nie należy nadpisywać nowej roli dla elementów, które już domyślnie mają przypisaną rolę. Mowa tu o sytuacji, gdy na przykład nagłówek ma już określoną rolę (`role="heading"`), więc dopisanie mu innej jest sprzeczne z zasadami.

Wszystkie elementy na witrynie, zawierające atrybut ARIA, a także ich funkcjonalności, powinny być dostępne z poziomu klawiatury. Sytuacja, w której komponent spełnia jakąś funkcję po kliknięciu myszą, a nie jest możliwe aktywowanie jej poprzez naciśnięcie przycisku “Enter” bądź “Return” jest niedopuszczalna.

Nie powinniśmy również stosować ARIA tam, gdzie nie jest to konieczne, na przykład przypisywać elementom na stronie role, które już pełnią.

ARIA posiada wsparcie następujących przeglądarek:

Nazwa	Wersja minimum	Informacja dodatkowa
Firefox	3.0+	Działa z NVDA, JAWS 10+ i Orca
Chrome	15+	Czytniki ekranowe mogą mieć problem z Chrome 15 i starszymi
Safari	4+	Działa z VoiceOver od iOS5 oraz OS X Lion
Opera	9.5+	Wymaga VoiceOver na OS X w początkowych wersjach
Internet Explorer	8+	Działa z JAWS 10+ i NVDA.

4. Szkic projektu

Blok nawigacji `<nav>`, blok-kontener `<div>` osadzone są w bloku głównym `<main>` podzielonym na dwie kolumny za pomocą przypisanej mu właściwości stylu `display: grid`:

```
<html>
*
*
*
  <main class="main" role="main">

    <nav id="section--nav" class="section--nav menu__container">...</nav>

    <div class="main__container">...</div>

  </main>
*
*
*
</html>

.main {
  display: grid;
  grid-template-columns: 20% 80%;
  max-width: 90em;
  margin: 0 auto;
  border: 1px solid #efefef;
}
```

Blok `<div class="main__container">` stanowi pojemnik na sekcje i podsekcje a w nich widżety które same także stanowią bloki/elementy/komponenty w rozumieniu metodologii BEM wzbogacone o zbiór atrybutów ARIA.

Umieszczone w projekcie widżety reprezentują typowe dla interfejsu witryny sieciowej elementy: nawigację/menu, rozwijalne sekcje, linki.

Struktura tych elementów stworzona została według reguł metodologii BEM a więc ich wewnętrzna organizacja uwzględnia zapis oraz podział kodu HTML na bloki, elementy oraz modyfikatory i tak jak wymaga tego BEM struktura ta ma odzwierciedlenie przede wszystkim w nazewnictwie klas ale także w formie i roli obiektów opisanych znacznikami HTML a także w komunikacji z tymi elementami stylów i skryptów.

Style tych komponentów oparte są tylko i wyłącznie o selektory klas ew. poszerzone o selektory atrybutów ARIA.

Dynamiczny aspekt strony realizowany przy użyciu JavaScript także opiera się o selekcję elementów przez nazwę klas (głównie w obrębie skryptów realizujących efekt **ScrollSpy**).

Wyjątek stanowi nawigacja która komunikację oraz identyfikację elementów docelowych realizuje poprzez wykorzystanie atrybutu `href` którego to argument stanowi `id` (atrybut znacznika) obiektu do którego nawigacja ma nas doprowadzić i jest to jedyne użycie identyfikatorów w obrębie projektu (przy czym same widżety są całkowicie od nich separowalne).

Poza strukturą narzuconą metodologią BEM komponenty strony wyposażono w atrybuty ARIA specyfikując tam gdzie jest to konieczne pełnioną rolę w interfejsie witryny (jak chociażby w obrębie nawigacji/menu).

Mamy zatem elementy których nazwy klas wskazują na charakter jednostki strukturalnej którą pełnią pozwalając na wygodne zarządzanie ich stylem oraz zachowaniem czy to poprzez arkusze stylów CSS czy też poprzez JavaScript wzbogacone o pozwalające na ich identyfikację dla technologii asystujących, w kontekście pełnionej roli, atrybuty ARIA.

Smooth Scrolling

Płynne przewijanie w obrębie witryny (**Smooth Scrolling**) uzyskujemy stosując pojedynczą linię kodu stylu CSS:

```
html {  
    font-family: sans-serif;  
    scroll-behavior: smooth;  
    overflow: scroll;  
    overflow-x: hidden;  
    background-color: #222;  
}
```

Przeźroczystość Sekcji poza widokiem głównym

Sekcjom znajdującym się poza widokiem głównym nadano przeźroczystość (opacity) na poziomie 0 (całkowicie przeźroczyste). Odpowiedzialny jest za to skrypt który realizuje także efekt **ScrollSpy**

Sekcje pojawiające się w widoku uzyskują po przekroczeniu jego progu pełną widoczność.

JavaScript: `js\section--nav__overlay.js`

```
if (entry.intersectionRatio > 0.0) {  
    if (window.innerWidth > 920) {  
        nameClassTarget[0].style.transition = "all 2000ms ease";  
        nameClassTarget[0].style.paddingTop = "6.5rem"  
        nameClassTarget[0].style.opacity = "1";  
    }  
  
    document.querySelector(`nav li a[href="#${id}"]`).parentElement.classList.add('active');  
    document.querySelector(`nav li  
a[href="#${id}"]`).parentElement.classList.remove('unactive');
```

```

}

else {
    if (window.innerWidth > 920) {
        nameClassTarget[0].style.opacity = "0";
    }

    document.querySelector(`nav li a[href="#${id}"]`).parentElement.classList.add('unactive');
    document.querySelector(`nav li a[href="#${id}"]`).parentElement.classList.remove('active');
}

```

Responsywność

Responsywność witryny specyfikowana jest dla rozdzielczości których `max-width` wynosi 920px. Główna zmiana w obrębie dokumentu, wynikająca z faktu wyświetlania po przekroczeniu tej granicznej wartości dla szerokości okna witryny, dotyczy układu głównych jej komponentów. Zarówno blok `<main class="main" role="main">` jak i wszystkie podrzędne względem niego elementy (nawigacja , sekcje) wyświetlane są w układzie blokowym, ich szerokość rozciągnięta jest na całą szerokość okna, nadano im jednakowy parametr kolejności warstw `z-index: 1`

```

.main {
    display: block;
    width:100%;
}

.main > * {
    display: block;
    width:100%;
    margin: 0;
    padding:0;
    z-index: 1;
}

```

Ukryto nagłówek pierwotnie umiejscowiony u góry witryny:

```

.main__content-header__sticky {
    display: none;
}

```

Pojemnik na nawigację `<nav class="section--nav menu--container">`

k który również wyświetlany jest jako blok umiejscowiony został na stałe u góry witryny `position: sticky`. Wprowadzone zmiany względem jego pierwotnej stylizacji wynikają z dostosowania wymiarów elementu oraz składających się na jego strukturę podelementów oraz zawartej w nich treści:

```

.section--nav {
    position: -webkit-sticky;
    position: sticky;
    top:0.5rem;
}

```

```

padding-left: 0;
width: 100%;
height: 500px;
z-index:1;
}

.section--nav {
    height: auto;
}

.section--nav * {
    background-color: transparent;
    padding-bottom: 0.1rem;
}

.section--nav__overlay {
    background: linear-gradient(to right, rgba(255,255,255, 1) 50%, rgba(255,255,255,
0.5) 100%);
}

.section--nav__overlay a{
    font-size: 0.75rem;
}

.section--nav .overlay-content a:hover,
.section--nav .overlay-content a:focus {
    background: linear-gradient(to right, rgba(255,0,0, 0.1) 0%, rgba(255,255,255, 0.5)
100%);
    border-radius: 2px;
    transition: 500ms;
}

```

Modyfikacji uległ sposób interakcji w obrębie menu specyfikowany przez JavaScript. Prócz standardowych sposobów otwarcia/zamknięcia menu w oparciu o domyślne elementy takie jak przyciski wprowadzono dodatkowo efekt zamknięcia menu po kliknięciu łącza (linku) nawigacyjnego. Pozwala to na przemieszczanie się do wybranego miejsca przy jednoczesnym zamknięciu menu które w innym przypadku zasłaniałoby widok strony.

Zmiana rozmiaru okna wychwytywana jest przez `addEventListener` przypisany do okna strony:

```

window.addEventListener("resize", function(){} );

```

W konsekwencji realizowany jest kod w bloku funkcji `function()` który linkom nawigacyjnym przypisuje `addEventListener` reagujący na zdarzenie `click`, przy którego wystąpieniu skolei aktywowana jest funkcja `openCloseNav`:

JavaScript: js\menu__active--track.js

```
function(){
  if (window.innerWidth <= 920) {
    document.querySelectorAll(`nav li a`).forEach((navlink) => {
      navlink.addEventListener('click', opencloseNav);
    });
  }
  else {
    document.querySelectorAll(`nav li a`).forEach((navlink) => {
      navlink.removeEventListener('click', opencloseNav);
    });
  }

  document.querySelector(".window__InnerHeight").innerHTML = window.innerHeight;
  document.querySelector(".window__InnerWidth").innerHTML = window.innerWidth;
}
```

JavaScript: js\section--nav__overlay.js

```
opencloseNav = () => {
  let expanded = document.getElementById("openbtn").getAttribute('aria-expanded') === 'true';
  document.getElementById("openbtn").setAttribute('aria-expanded', !expanded);

  if(expanded == false) {
    openNav();
  }
  else {
    closeNav();
  }
}

openNav = () => {
  document.getElementById("section--nav__overlay").style.height = "100%";
  document.getElementById("section--nav__overlay").style.opacity = "1";
  document.getElementById("section--nav__overlay").style.visibility = "visible";
  document.getElementById("section--nav__overlay").style.display = "block";
  document.getElementById("closebtn").style.display = "block";
  document.getElementById("openbtn").style.color = "#111";
}

closeNav = () => {
  document.getElementById("section--nav__overlay").style.height = "0%";
  document.getElementById("section--nav__overlay").style.opacity = "0";
  document.getElementById("section--nav__overlay").style.visibility = "hidden";
  document.getElementById("closebtn").style.display = "none";
  document.getElementById("openbtn").style.color = "#efefef";

  if (window.innerWidth <= 920) {
    document.getElementById("section--nav__overlay").style.display = "none";
  }
}
```

5. Nawigacja/Menu

Efekt ScrollSpy

Aby określić miejsce do którego strona jest przewijana i podświetlić aktywny i związany z tym miejscem element nawigacji niezbędny jest realizujący te cele skrypt JS. Aktywowana na stronie treść to w tym przypadku element menu oraz blok kontener w obrębie którego następuje przewijanie witryny. Odpowiednim do tego narzędziem jest `Intersection Observer API` które to właśnie jest odpowiedzialne za efekt **ScrollSpy**.

Dzięki `IntersectionObserver` możemy zaimplementować `ScrollSpy`. Używamy go do obserwacji wszystkich sekcji dokumentu:

```
document.querySelectorAll('section[class]').
```

Struktura BEM projektu a tym samym unikatowe nazwy klas pozwalają na identyfikację obserwowanych sekcji przez nazwy klas właśnie:

```
const nameClass = entry.target.getAttribute('class');  
let nameClassTarget = document.getElementsByClassName(nameClass);
```

Nie mniej elementy menu których styl ulega modyfikacji wraz z pojawieniem się w widoku obserwowanych sekcji wskazywane są przez zapytania wykorzystujące przypisany tym elementom atrybut `id`:

```
const id = entry.target.getAttribute('id');  
document.querySelector(`nav li a[href="#${id}"]`)
```

Jeśli elementy te przecinają się w widoku głównym, czyli gdy próg widoku (ang. `threshold`) zostanie przekroczony w jednym lub drugim kierunku `entry.intersectionRatio > 0.0` (obiekt może pojawić się z dołu lub z góry) to dodajemy klasę `.active` do każdego linku - elementu nawigacji, który do niego prowadzi oraz klasę `.unactive` tym linkom dla których przypisane im sekcje są poza widokiem.

(Próg 1.0 oznacza, że gdy 100% celu jest widoczne wywoływana jest funkcja - wywołanie zwrotne określająca zachowanie względem interesujących nas elementów witryny w naszym przypadku elementów nawigacji oraz sekcji pojawiających się w widoku).

Aktywny styl nie dodajemy do samego linku, ale do otaczającego go elementu (rodzica):

```
.parentElement.classList.add('active').
```

JavaScript: js\section--nav__overlay.js

```
const observer = new IntersectionObserver(entries => {
  entries.forEach(entry => {
    const id = entry.target.getAttribute('id');
    const nameClass = entry.target.getAttribute('class');
    let nameClassTarget = document.getElementsByClassName(nameClass);

    if (entry.intersectionRatio > 0.0) {
      if (window.innerWidth > 920) {
        nameClassTarget[0].style.transition = "all 2000ms ease";
        nameClassTarget[0].style.paddingTop = "6.5rem"
        nameClassTarget[0].style.opacity = "1";
      }

      document.querySelector(`nav li a[href="#${id}"]`).parentElement.classList.add('active');
      document.querySelector(`nav li
a[href="#${id}"]`).parentElement.classList.remove('unactive');
    }

    else {
      if (window.innerWidth > 920) {
        nameClassTarget[0].style.opacity = "0";
      }

      document.querySelector(`nav li a[href="#${id}"]`).parentElement.classList.add('unactive');
      document.querySelector(`nav li a[href="#${id}"]`).parentElement.classList.remove('active');
    }
  });
});

document.querySelectorAll('section[class]').forEach((section) => {
  section.style.color = "#333";
  observer.observe(section);
});
```

Nawigacja/Menu - WAI-ARIA

Menu (w sensie WAI-ARIA) identyfikuje się technologiom asystującym przez przypisanie mu roli menu (dla kontenera) `role="menu"` oraz ról dla elementów względem niego podrzędnych `role="menuitem"` (mogą mieć zastosowanie także inne role podrzędne). Kreuje to rodzaj komponentu hybrydowego, który wskazuje użytkownikowi dzięki atrybutom ARIA czy lista linków jest otwarta a także nadaje komponentowi semantyczną strukturę menu.

(nie jest wskazane umieszczenie roli `menuitem` na każdym łączu (linku) które to spowodowałoby pominięcie niejawnej roli łącza)

Element:

```
<button id="openbtn" class="menu__button__openbtn" aria-expanded="true"
aria-haspopup="true" aria-controls="menu-list"><span>&#9776;</span></button>
```

odpowiedzialny jest za otwarcie/zamknięcie menu.

Menu posiada także domyślny przycisk zamknięcia:

```
<button id="closebtn" class="menu__button__closebtn"><span>&times;</span></button>
```

Otwarcie/zamknięcie menu realizowane jest przez JavaScript. W oparciu o informację o stanie menu skrypt modyfikuje styl przycisków a także argument atrybutu ARIA `aria-expanded`. Stan ten (otwarte/zamknięte menu) ma w oczywisty sposób swą wizualną reprezentację (kolor przypisany do przycisku zależny od aktualnego stanu menu), ale informację o nim należy również przekazać technologiom asystującym. Robimy to nadając właśnie argument atrybutu początkowo w stanie prawdy `aria-expanded="true"` (otwarte menu).

JavaScript: js\section--nav__overlay.js

```
(function() {  
  let openBtn = document.getElementById("openbtn");  
  let closeBtn = document.getElementById("closebtn");  
  
  openBtn.onclick = () => {  
    let expanded = document.getElementById("openbtn").getAttribute('aria-expanded') === 'true';  
    openBtn.setAttribute('aria-expanded', !expanded);  
  
    if(expanded == false) {  
      openNav();  
    }  
    else {  
      closeNav();  
    }  
  }  
  
  closeBtn.onclick = () => {  
    let expanded = document.getElementById("openbtn").getAttribute('aria-expanded') === 'true';  
    openBtn.setAttribute('aria-expanded', !expanded);  
  
    if(expanded == false) {  
      openNav();  
    }  
    else {  
      closeNav();  
    }  
  }  
})()
```

Nawigacja/Menu - struktura BEM

Blok menu stanowi element `<nav class="section--nav menu--container">`. W nazwie klasy użyto miksów wskazującego na funkcję komponentu w jego ogólnym znaczeniu jako pojemnika na "menu" `"menu--container"` jak i uszczegółowionej roli jaką pełni w obrębie projektu czyli nawigacji w obrębie sekcji `"section--nav"`. Pozwala to na modyfikację jego stylu konkretnie pod projekt bez interakcji z ogólną stylizacją samego komponentu. Miksów użyto również w odniesieniu do elementów kontrolnych nawigacji czyli przycisków otwarcia i zamknięcia menu.

Poszczególne elementy zawarte w bloku `"menu--container"` to element kontrolne menu pozwalający na interakcję z nim (takie jak przyciski) oraz elementy składające się na jego strukturę czyli pojemnik na menu oraz listy wraz z ich elementami.

Mamy zatem elementy których nazwy klas wskazują na charakter jednostki strukturalnej którą pełnią pozwalając na wygodne zarządzanie ich stylem oraz zachowaniem czy to poprzez arkusze stylów CSS czy też poprzez JavaScript.

Na blok nawigacji składają się zatem:

```
menu-container
menu
```

Elementy w obrębie tych bloków to:

```
menu__overlay-content
menu__ordered-list
menu__unordered-list
menu__button
menu__item
menu__link
```

Modyfikatory elementów to:

```
menu__button__openbtn
menu__button__closebtn
```

BEM:

```
menu-container (BLOK):  
  menu__button (ELEMENT):  
    menu__button__openbtn (MODYFIKATOR)  
  
  menu (BLOK):  
    menu__button (ELEMENT)  
    menu__button__closebtn (MODYFIKATOR)  
  menu__overlay-content (ELEMENT)  
  menu__ordered-list (ELEMENT)  
  menu__unordered-list (ELEMENT)  
  menu__item (ELEMENT)  
  menu__link (ELEMENT)
```

HTML: index_scrolling_active_state.html

```
<nav class="section--nav menu--container">  
  
  <button id="openbtn" class="openbtn menu__button__openbtn" aria-expanded="true"  
    aria-haspopup="true" aria-controls="menu-list" title="open"><span>&#9776;</span></button>  
  
  <div id="section--nav__overlay" class="section--nav__overlay menu" role="menu">  
  
    <button id="closebtn" class="closebtn menu__button__closebtn"  
      title="close"><span>&times;</span></button>  
    <div class="menu__overlay-content">  
      <ol class="menu__ordered-list">  
        <li role="menuitem" class="menu__item">  
          <a class="menu__link">...</a></li>  
        <li role="menuitem" class="menu__item">  
          <a class="menu__link">...</a>  
          <ul class="menu__unordered-list">  
            <li role="menuitem" class="menu__item">  
              <a class="menu__link">...</a>  
            </li>  
            .  
            .  
            .  
          </ul>  
        </li>  
        <li role="menuitem" class="menu__item">  
          <a class="menu__link">...</a></li>  
      </ol>  
    </div>  
  </div>  
</nav>
```

6. Zwijane Sekcje (Collapsible Sections)

Zwijane sekcje są jednymi z najbardziej podstawowych wzorców interaktywnego projektowania w sieci. Pozwalają na zmianę widoczności treści poprzez kliknięcie 'etykiety' elementu sekcji który reprezentują. Choć interakcja jest prosta, jest to interakcja, która nie ma spójnej naturalnej implementacji we wszystkich przeglądarkach co zmusza do projektowania wskazanej interakcji przy użyciu JavaScript i WAI-ARIA.

(Jedną z zalet zwijania treści jest to, że nagłówki stają się sąsiadującymi elementami, dając użytkownikowi przegląd dostępnej treści bez konieczności przewijania strony. Rozszerzanie treści to wybór użytkownika jej zobaczenia)

Zwijana sekcja bez powiązanego z jej treścią JavaScript to po prostu sekcja. To znaczy: podtytuł wprowadzający jakąś treść (opisujący ją i do niej prowadzący).

Kolejną zaletą jest to, że użytkownicy klawiatury nie muszą przechodzić przez wszystkie elementy strony, które mogą być aktywowane, aby dostać się tam, gdzie chcą: na ukrytą zawartość nie można ustawić "fokusu" (atrybut `focusable="false"`).

Dostosowanie znacznika nagłówka

Podłączenie modułu obsługi kliknięcia do nagłówka w celu rozwinięcia powiązanej treści nie jest interakcją przekazywaną programom pomocniczym lub możliwą do osiągnięcia za pomocą klawiatury. Zamiast tego musimy dostosować znaczniki wprowadzając standardowy element przycisku:

```
<h4>
  <button class="collapsible-sections--item__button">
    Collapsible sections - Subsection 1
  </button>
</h4>
<div class="collapsible-sections--item__content">
  <section class="collapsible-sections--item__content--section">
    .
    .
    .
  </section>
</div>
```

Przycisk jest elementem podrzędnym nagłówka. Oznacza to, że gdy użytkownik czytnika ekranu skupia się na `<button>`, identyfikowany jest sam przycisk, ale także obecność jego elementu nadrzędnego.

Konwersja nagłówka na przycisk przy użyciu funkcji `ARIA role = "button"`, spowodowałaby semantyczne nadpisanie roli nagłówka. Użytkownicy czytników ekranu straciliby informację o nagłówku jako wskazówkę strukturalną i nawigacyjną.

Ponadto odpowiedniego dostosowania wymagałyby wszystkie zachowania przeglądarki, które `<button>` daje nam automatycznie, takie jak `focus` i przypisania klawiszy.

Informacja o stanie nagłówka

Komponent może znajdować się w jednym z dwóch wzajemnie wykluczających się stanów: zwinięty lub rozwinięty. Stan ten można zasugerować wizualnie, ale informację o nim należy również przekazać poza wizualną stronę nagłówka. Możemy to zrobić, nadając atrybut ARIA dla przycisku `aria-expanded="false"` - początkowo w stanie fałszu (zwiniętym). Co z kolei wymaga ukrycia powiązanego z komponentem kontenera na treść sekcji `<div class="collapsible-sections--item__content">` przez nadanie mu atrybutu `hidden`:

```
<h4>
  <button aria-expanded="false" class="collapsible-sections--item__button">
    Collapsible sections - Subsection
  </button>
</h4>
<div hidden class="collapsible-sections--item__content">
  <section class="collapsible-sections--item__content--section">
    .
    .
    .
  </section>
</div>
```

Niektóre starsze przeglądarki nie obsługują atrybutu `hidden`, należy zatem pamiętać, aby umieścić następujące informacje w plikach stylów CSS:

```
[hidden] {
  display: none;
}
```

Stan jest zatem przekazywany za pomocą elementu sterującego, którego używa się do jego przełączania.

Nie musimy dodawać roli ARIA `role = "button"`, ponieważ element `<button>` niejawnie ma tę rolę (rola ARIA służy tylko do naśladowania natywnej roli zastępując role określone przez znaczniki HTML).

Styl nagłówka

Elementem wskazującym potencjalny sposób oddziaływania przycisku jest intuicyjny i tradycyjny symbol plus/minus. Domyślnie plus wskazuje, że sekcję można rozwinąć, a minus, że można ją zwinąć.

Proste kształty, takie jak prostokąty `<rect>`, są bardzo wydajnym sposobem tworzenia ikon w formacie SVG. Rendering ikony jest zatem efektywny i optymalny i opiera się właśnie o ten format:

```
<svg aria-hidden="true" focusable="false" viewBox="0 0 10 10"
class="collapsible-sections--item__svg">
  <rect class="collapsible-sections--item__svg__vert" height="8" width="2" y="1"
x="4"/>
  <rect height="2" width="8" y="4" x="1"/>
</svg>
```

`aria-hidden="true"` ukrywa SVG przed czytnikami ekranu i innymi technologiami pomocniczymi

`focusable="false"` usuwa domyślne ustawienie skupienia w Internet Explorer / Edge na SVG

Zmiana ikony między kształtem plus i minus (powiązana z aktualnym stanem komponentu) odbywa się za pomocą selektora klasy CSS przy czym realizacja tej funkcjonalności wymaga nadania nazwy klasie dla elementu SVG reprezentującego pionową rozpórkę :

```
<rect class="collapsible-sections--item__svg__vert" height="8" width="2" y="1" x="4"/>

[aria-expanded="true"] .collapsible-section__button_vert {
  opacity: 0.0;
  transition: all 100ms;
}
```

Warto zauważyć że organizacja nazwy przedstawia blok `collapsible-section` (reprezentujący cały komponent), element `button` i modyfikator `vert` (reprezentujący powiązanie przeznaczenia, stylu i stanu) w rozumieniu metodologii BEM.

Prostota interakcji oraz wszystkie elementy i semantyka wpisane w strukturę komponentu mają przełożenie na realizujący jego funkcjonalność JavaScript.

JavaScript: js\collapsible-section.js

```
(function() {  
  const headings = document.querySelectorAll('h4');  
  
  Array.prototype.forEach.call(headings, h => {  
    let btn = h.querySelector('button');  
    let target = h.nextElementSibling;  
  
    target.classList.add('target-unexpanded');  
  
    btn.onclick = () => {  
      let expanded = btn.getAttribute('aria-expanded') === 'true';  
      btn.setAttribute('aria-expanded', !expanded);  
  
      if(expanded == true) {  
        target.classList.add('target-unexpanded');  
        target.classList.remove('target-expanded');  
        target.hidden = expanded;  
      }  
      else {  
        target.classList.add('target-expanded');  
        target.classList.remove('target-unexpanded');  
        target.hidden = expanded;  
      }  
    }  
  });  
})();
```

Zwijane sekcje - struktura BEM

Blok główny `class="collapsible-sections"` zawiera elementy podrzędne `class="collapsible-sections--item"` stanowiące przy tym jednostki od niego separowalne, także bloki w rozumieniu BEM, którego to funkcja sprowadza się do roli kontenera na te komponenty (tym samym można wykorzystywać je od niego niezależnie).

Elementy komponent w rozumieniu struktury BEM to elementy kontrolne takie jak `<button>` jednostki mówiące o stanie komponentu: `<svg>` a także pojemniki na zawartą w nim treść takie jak podrzędne sekcje czy element `<div>`.

Modyfikator wskazuje obiekt sygnalizujący zmianę stanu przycisku `<rect>`.

Na blok nawigacji składają się zatem:

```
collapsible-sections
collapsible-sections--item
```

Elementy w obrębie tych bloków to:

```
collapsible-sections--item__button
collapsible-sections--item__button__svg
collapsible-sections--item__subitem--content
collapsible-sections--item__content
```

Modyfikatory elementów to:

```
collapsible-sections--item__button__vert
```

BEM:

collapsible-sections (BLOK):

```
collapsible-sections--item (BLOK):  
    collapsible-sections--item__button (ELEMENT)  
    collapsible-sections--item__svg (ELEMENT):  
        collapsible-sections--item__button__vert (MODYFIKATOR)  
  
    collapsible-sections--item__subitem--content (ELEMENT)  
    collapsible-sections--item__content (ELEMENT)
```

HTML: index_scrolling_active_state.html

```
<div style="overflow:auto;" class="collapsible-sections">  
  
  <div class="collapsible-sections--item">  
    <h4>  
      <button aria-expanded="false" class="collapsible-sections--item__button">  
        Collapsible sections - Subsection 1  
        <svg class="collapsible-sections--item__svg"  
          aria-hidden="true" focusable="false" viewBox="0 0 10 10">  
          <rect class="collapsible-sections--item__svg__vert" height="8" width="2" y="1"  
            x="4"/>  
          <rect height="2" width="8" y="4" x="1"/>  
        </svg>  
      </button>  
    </h4>  
  
    <div class="collapsible-sections--item__subitem--content">  
      <section class="collapsible-sections--item__content" style="margin: 0; padding:  
0;">  
        <p>...</p>  
      </section>  
    </div>  
  </div>  
  
  <div> another collapsible-sections--item </div>  
  <div> another collapsible-sections--item </div>  
  .  
  .  
  .  
</div>
```

7. Karty linków (Cards)

Podstawowa karta zawiera ilustrację, „tytuł”, opis i przypisanie.

Podobnie jak w przypadku znaczników listy, nagłówki zapewniają wskazówki nawigacyjne dla użytkowników czytników ekranu. Każda karta ma nagłówek tego samego poziomu - w tym przypadku `<h2>` - ponieważ należą one do płaskiej hierarchii list. Obraz jest traktowany jako dekoracyjny i jest „wyciszany” w danych wyjściowych czytnika ekranu przy użyciu pustej wartości atrybutu `alt`.

Aby umożliwić kliknięcie całej karty jako linku bez zmiany hierarchii i struktury ustalonych przez nas znaczników nadajemy pojemnikowi na kartę własność `position: relative`

```
.cards__item{
  border: 1px solid;
  border-radius: 0.25rem;
  display: flex;
  flex-direction: column;
  position: relative;
  margin-top: 1.5rem;
}
```

Dla treści występującej po linkach tzn. wskazanej selektorem `::after` nadajemy własność `position: absolute` natomiast własnościom `left`, `top`, `right` i `bottom` wartość `0`

```
.cards__item a::after {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  right: 0;
  bottom: 0;
}
```

Zabieg ten rozciąga układ linku na całą kartę, dzięki czemu jest klikalny jak przycisk zaś karta przyjmuje styl kursora wskaźnika.

Afordancja

Modyfikując styl karty należy zasygnalizować użytkownikowi jej interaktywny charakter jako całości wspierając postrzegana afordancję elementu. Realizują to style: `:hover`, `:focus`, `:focus-within`

```
.cards__item a:focus {
  outline: none;
  text-decoration: underline;
}
```

```

.cards__item:focus-within {
  box-shadow: -0px 2px 8px 0.1rem;
  top: -4px;
}

.cards__item:hover {
  box-shadow: -0px 2px 8px 0.1rem;
  top: -4px;
  transition: all 0.25s;
  transition-timing-function: ease;
}

.cards__item:focus-within a:focus {
  text-decoration: none;
}

```

Tam, gdzie istnieją style po wskazaniu elementu `:hover` (najechaniu kursorem), powinien być również styl skupienia `:focus`. Używając `:focus`, możemy zastosować styl tylko do samego linku aby stylizacja była dostępna również dla użytkowników z poziomu klawiatury wymagane jest użycie `:focus-within`.

Pseudoklasa `:focus-within` reprezentuje element, który otrzymał fokus lub zawiera element, który fokus otrzymał czyli reprezentuje element, który sam jest określony przez tą pseudoklasę lub ma określonego przez nią potomka.

Układ

Używając `@supports`, implementujemy prosty, jednokolumnowy układ, a następnie rozszerzamy go o siatkę `display: grid` jeśli jest obsługiwana.

Umieszczenie karty w kontekście CSS Flexbox lub siatki CSS, powoduje że karty rozciągają się do tej samej wysokości: wysokości karty z największą zawartością.

Grid (siatka) i Flexbox mogą mieć ten efekt przy czym `grid-gap` to najłatwiejszy sposób na dystrybucję kart stąd to siatka wykorzystana jest w projekcie.

Maksymalna szerokość to 60 znaków w kontenerze tekstu. Zapobiega to nadmiernemu wydłużaniu się linii dla kart na dużych ekranach, które nie obsługują siatki.

Używając `aria-describedby="description-card"` możemy dołączyć "read more" do linku jako opis gdzie pomimo zastosowania `aria-hidden="true"`, utworzona relacja pozostaje nienaruszona, a opis jest dostępny dla łącza. Jest to przydatne, gdy chcemy użyć elementu do opisu, ale nie chcemy, aby technologie wspomagające bezpośrednio rozpoznawały element. Dla użytkownika czytnika ekranu byłoby mylące, gdyby mógł przejść od wezwania do działania i usłyszeć nakaz "czytaj więcej" ("read more") bez elementu odgrywającego rolę lub mającego interaktywny charakter.

Karty linków - Struktura BEM

Blok główny `"cards__container cards"` zawiera także zagnieżdżony w nim blok `"cards-item card"` oraz podrzędne względem tego ostatniego elementy `"cards-item__img img"`, `"cards-item__text"`, `"cards-item__link"` oraz `"cards-item__text_readmore"`.

Na blok nawigacji składają się zatem:

```
cards__container cards
cards-item card
```

Elementy w obrębie tych bloków to:

```
cards-item__img img
cards-item__text
cards-item__link
cards-item__text_readmore
```


BEM:

```
cards__container cards (BLOK):  
  
  cards-item card (BLOK):  
    cards-item__img img (ELEMENT)  
    cards-item__text (ELEMENT)  
    cards-item__link (ELEMENT)  
    cards-item__text_readmore (ELEMENT)
```

HTML: index_scrolling_active_state.html

```
<div class="cards__container cards" >  
  
  <ul>  
    <li class="cards-item card">  
      <div class="cards-item__img img"></div>  
      <div class="cards-item__text text">  
        <h2>  
          <a href="https://inclusive-components.design/" target="_blank"  
            class="cards-item__link" aria-describedby="description-card"  
            >...</a>  
        </h2>  
        <p>Commodo ut laborum fugiat aliqua eiusmod voluptate pariat. </p>  
        <small class="cards-item__text_readmore" aria-hidden="true">  
          Read more <span>&rarr;</span>  
        </small>  
      </div>  
    </li>  
  
    <li>another cards__item</li>  
    <li>another cards__item</li>  
    .  
    .  
    .  
  </ul>  
</div>
```

8. Audyt strony

Chrome DevTools - Lighthouse Report

Stronę poddano badaniu przy użyciu Google Lighthouse pluginu Chrome DevTools zestawu narzędzi dla programistów internetowych wbudowanych bezpośrednio w przeglądarkę Google Chrome.

Performance score – wydajność strony

Audyt wydajności strony obejmuje różne badane aspekty działania strony. Jednym z przykładów jest audyt First Meaningful Paint (FMP), który bada czas wyświetlenia wizualnej treści strony. Mierzony jest tu dokładnie czas pomiędzy rozpoczęciem procesu ładowania strony a czasem renderowania elementów graficznych. Według danych Google, parametr FMP dla stron osiągających bardzo dobre wyniki wynosi około 1,220 ms. Osiągnięcie takiego czasu oznacza uzyskanie oceny 99.

Nie każdy aspekt działania strony ma identyczną wagę w ocenie końcowej. [Tutaj](#) dowiesz się, jaką wagę mają w niej poszczególne aspekty działania strony.

Przeprowadzając parokrotnie audyt strony przy pomocy Lighthouse, prawdopodobnie zauważysz różnice w wynikach wydajności pracy strony. Dzieje się tak dlatego, że strona za każdym razem będzie ładować się w różny sposób i z różną prędkością. Jest to całkowicie normalne.

Progressive Web App score

Progressive Web App score lub PWA to normalna strona internetowa posiadająca jednak funkcje aplikacji, która dostępna jest również offline. To oznacza, że strona może być używana jako rodzaj uproszczonej aplikacji. Jest to obecnie ważny kierunek rozwoju, gdyż niemal wszystko odbywa się za pośrednictwem urządzeń mobilnych.

W tym obszarze audytu sprawdza się zatem, na ile dobrze strona internetowa działa jako uproszczona aplikacja mobilna.

Wynik PWA opiera się na 'Baseline Progressive Web App Checklist' firmy Google. Obejmuje ona 14 parametrów, które musi spełnić badana strona internetowa. Lighthouse podczas audytu automatycznie sprawdza 11 z 14 parametrów z tej listy. W odróżnieniu od wyniku w

obszarze Wydajności (Performance score), tutaj każdy parametr ma tę samą wagę. Każdy spełniony parametr odpowiada zatem 9 punktom dla oceny uzyskiwanej w obszarze PWA.

Accessibility score – dostępność

Wynik w obszarze Accessibility uwzględnia różne parametry w różny sposób (prawie tak jak w obszarze Performance). Różnica polega jednak na tym, że punkty oceny przyznawane są tylko wtedy, gdy strona w całości spełnia kryteria danego parametru. W obszarze Performance, za częściowe spełnienie danego warunku, otrzymuje się odpowiednio pomniejszoną punktację. Tutaj tak to nie działa – w tym obszarze liczy się wszystko albo nic.

W obszarze Accessibility istnieją bardzo jasno określone warunki, które należy spełnić, takie jak: obrazy mają tekst alternatywny, język strony jest zrozumiały, elementy strony mają wyraźną strukturę, itp.

Best practices – najlepsze praktyki

Tutaj sprawdzane są różne parametry, takie jak użycie HTTPS, to, czy użytkownicy mogą wkleić tekst w pole „hasło” oraz inne kwestie techniczne.

Są to zatem elementy, które są często używane na innych stronach internetowych i dobrze na nich działają. Jest to zatem pewien standard dla współczesnych stron internetowych. Wynik dla tego obszaru pokazuje, jak strona się w niego wpisuje.

SEO score – wynik SEO

Wynik dla obszaru SEO pokazuje, na ile badana strona jest przyjazna dla wyszukiwarek internetowych i jak pozycjonuje się w wyszukiwarce Google (i innych).

Czy Twoja strona jest przyjazna dla urządzeń mobilnych? Czy wielkość czcionki jest normalna? Czy dokonuje się rozróżnienia pomiędzy H1 i H2?

Chodzi głównie o to, jak strona wygląda i czy może uzyskiwać dobry wynik i wysoką pozycję w Google.

Wyniki audytu:



0-49 50-89 90-100



Performance

Metrics

<ul style="list-style-type: none">● First Contentful Paint 1.4 s First Contentful Paint marks the time at which the first text or image is painted. Learn more.	<ul style="list-style-type: none">● Time to Interactive 1.8 s Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.
<ul style="list-style-type: none">● Speed Index 1.4 s Speed Index shows how quickly the contents of a page are visibly populated. Learn more.	<ul style="list-style-type: none">● Total Blocking Time 110 ms Sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in milliseconds. Learn more.
<ul style="list-style-type: none">● Largest Contentful Paint 1.4 s Largest Contentful Paint marks the time at which the largest text or image is painted. Learn More	<ul style="list-style-type: none">● Cumulative Layout Shift 0 Cumulative Layout Shift measures the movement of visible elements within the viewport. Learn more.

Values are estimated and may vary. The [performance score is calculated](#) directly from these metrics. [See calculator.](#)





Opportunities — These suggestions can help your page load faster. They don't [directly affect](#) the Performance score.

Opportunity Estimated Savings

▲ Serve images in next-gen formats 21.6 s ^

Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. [Learn more](#).

☐ Show 3rd-party resources (0)

URL	Resource Size	Potential Savings
 /images/e4.jpg (192.168.1.3)	3,918.7 KB	3,410.3 KB
 /images/b&w0cities.png (192.168.1.3)	765.8 KB	676.5 KB
 /images/img_cities0.jpg (192.168.1.3)	254.1 KB	103.4 KB
 /images/e44.jpg (192.168.1.3)	352 KB	82.7 KB
 /images/e43.jpg (192.168.1.3)	242.7 KB	9.3 KB

▲ Efficiently encode images 16.8 s v

■ Enable text compression 0.45 s v

■ Remove unused JavaScript 0.15 s v

Diagnostics — More information about the performance of your application. These numbers don't [directly affect](#) the Performance score.

▲ Serve static assets with an efficient cache policy — 8 resources found	▼
▲ Does not use HTTP/2 for all of its resources — 9 requests not served via HTTP/2	▼
▲ Avoid enormous network payloads — Total size was 5,797 KB	▼
● Avoid chaining critical requests — 2 chains found	▼
● Keep request counts low and transfer sizes small — 11 requests • 5,797 KB	▼
● Largest Contentful Paint element — 1 element found	▼
● Avoid large layout shifts — 1 element found	▼
Passed audits (19)	
● Eliminate render-blocking resources — Potential savings of 0 ms	▼
● Properly size images	▼
● Defer offscreen images	▼
● Minify CSS	▼
● Minify JavaScript	▼
● Remove unused CSS	▼
● Preconnect to required origins	▼
● Initial server response time was short — Root document took 10 ms	▼
● Avoid multiple page redirects	▼
● Preload key requests	▼
● Use video formats for animated content	▼
● Avoids an excessive DOM size — 252 elements	▼
● User Timing marks and measures	▼
● JavaScript execution time — 0.4 s	▼
● Minimizes main-thread work — 1.6 s	▼
● All text remains visible during webfont loads	▼
● Minimize third-party usage	▼
● Uses passive listeners to improve scrolling performance	▼
● Avoids <code>document.write()</code>	▼



Accessibility

These checks highlight opportunities to [improve the accessibility of your web app](#). Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

ARIA — These are opportunities to improve the usage of ARIA in your application which may enhance the experience for users of assistive technology, like a screen reader.

- ▲ [aria-*) attributes do not have valid values
- ▲ ARIA IDs are not unique

Tables and lists — These are opportunities to improve the experience of reading tabular or list data using assistive technology, like a screen reader.

- ▲ Lists do not contain only `` elements and script supporting elements (`<script>` and `<template>`).

Screen readers have a specific way of announcing lists. Ensuring proper list structure aids screen reader output. [Learn more](#).

Failing Elements

Introduction BEM Widgets: Collapsible Sections Collapsible Sections with Cards ...

```
<ol class="menu__ordered-list">
```

Collapsible Sections Collapsible Sections with Cards Section with Cards ...blan...

```
<ul class="menu__unordered-list">
```

Additional items to manually check (10) — These items address areas which an automated testing tool cannot cover. [Learn more in our guide on conducting an accessibility review](#).

Passed audits (20)

- [aria-*] attributes match their roles
- [aria-hidden="true"] is not present on the document <body>
- [aria-hidden="true"] elements do not contain focusable descendents
- [role]s have all required [aria-*] attributes
- Elements with an ARIA [role] that require children to contain a specific [role] have all required children.
- [role]s are contained by their required parent element
- [role] values are valid
- [aria-*] attributes are valid and not misspelled
- Buttons have an accessible name
- The page contains a heading, skip link, or landmark region
- Background and foreground colors have a sufficient contrast ratio
- Document has a <title> element
- [id] attributes on active, focusable elements are unique
- Heading elements appear in a sequentially-descending order
- <html> element has a [lang] attribute
- <html> element has a valid value for its [lang] attribute
- Image elements have [alt] attributes
- Links have a discernible name
- List items () are contained within or parent elements
- [user-scalable="no"] is not used in the <meta name="viewport"> element and the [maximum-scale] attribute is not less than 5.



Best Practices

Trust and Safety

- ▲ Does not use HTTPS — 9 insecure requests found
- ▲ Links to cross-origin destinations are unsafe

General

- ▲ Browser errors were logged to the console

Passed audits (11)

- Avoids requesting the geolocation permission on page load
- Avoids requesting the notification permission on page load
- Avoids front-end JavaScript libraries with known security vulnerabilities
- Allows users to paste into password fields
- Displays images with correct aspect ratio
- Displays images with appropriate size
- Page has the HTML doctype
- Properly defines charset
- Avoids Application Cache
- Detected JavaScript libraries
- Avoids deprecated APIs



SEO

These checks ensure that your page is optimized for search engine results ranking. There are additional factors Lighthouse does not check that may affect your search ranking. [Learn more](#).

Content Best Practices — Format your HTML in a way that enables crawlers to better understand your app's content.

▲ Document does not have a meta description

Meta descriptions may be included in search results to concisely summarize page content. [Learn more](#).

Additional items to manually check (1) — Run these additional validators on your site to check additional SEO best practices.

Passed audits (10)





- Has a `<meta name="viewport">` tag with `width` or `initial-scale`
- Document has a `<title>` element
- Page has successful HTTP status code
- Links have descriptive text
- Page isn't blocked from indexing
- Image elements have `[alt]` attributes
- Document has a valid `hreflang`
- Document uses legible font sizes — 97.06% legible text
- Document avoids plugins
- Tap targets are sized appropriately — 100% appropriately sized tap targets



Progressive Web App

These checks validate the aspects of a Progressive Web App. [Learn more.](#)

















Fast and reliable

-  Page load is fast enough on mobile networks 
-  Current page does not respond with a 200 when offline 
-  `start_url` does not respond with a 200 when offline **No usable web app manifest found on page.** 

Installable

-  Does not use HTTPS — **9 insecure requests found** 
-  Does not register a service worker that controls page and `start_url` 
-  Web app manifest does not meet the installability requirements **Failures: No manifest was fetched.** 

PWA Optimized

-  Does not redirect HTTP traffic to HTTPS 
-  Is not configured for a custom splash screen **Failures: No manifest was fetched.** 
-  Does not set a theme color for the address bar.
Failures: No manifest was fetched, No ``<meta name="theme-color">`` tag found. 
-  Content is sized correctly for the viewport 
-  Has a `<meta name="viewport">` tag with `width` or `initial-scale` 
-  Contains some content when JavaScript is not available 
-  Does not provide a valid `apple-touch-icon` 
-  Manifest doesn't have a maskable icon **No manifest was fetched** 

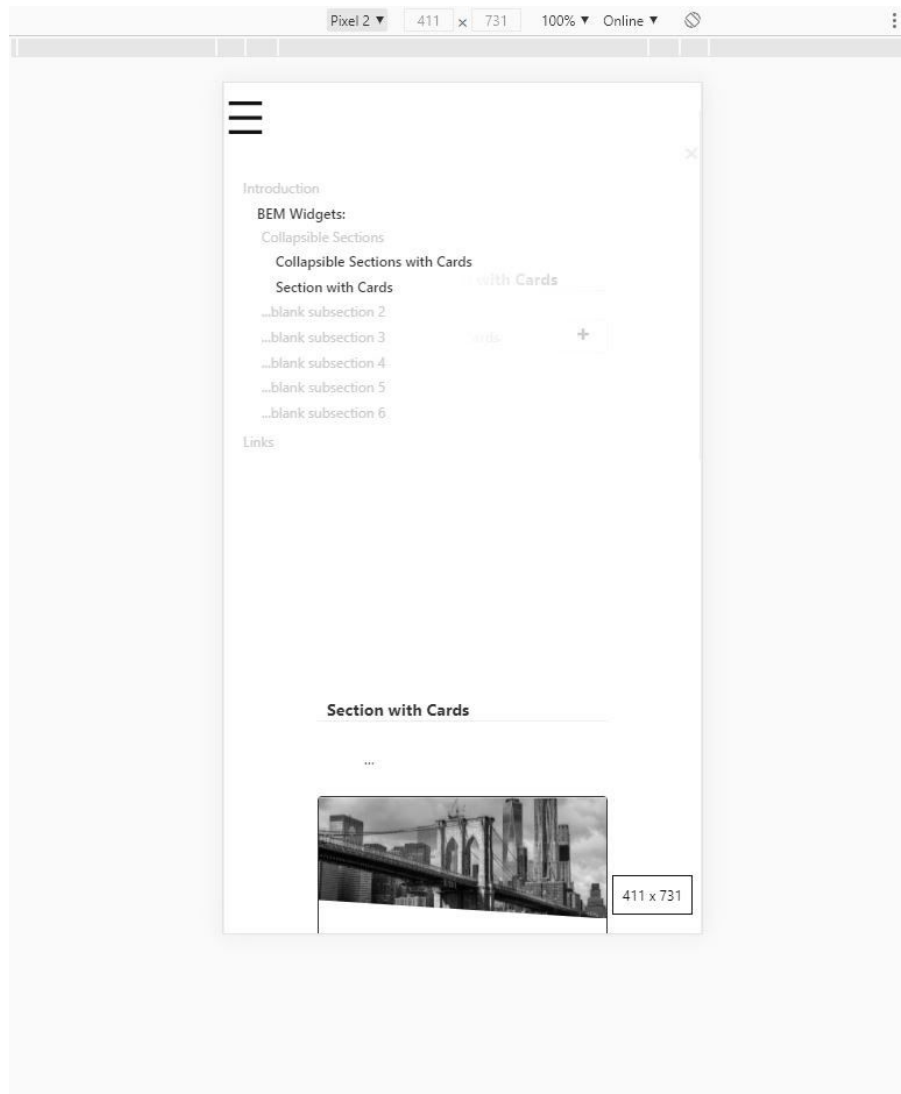
Runtime Settings

URL	http://192.168.1.3:8080/index_scrolling_active_state.html
Fetch Time	Sep 23, 2020, 7:22 PM GMT+2
Device	Emulated Moto G4
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
Channel	devtools
User agent (host)	Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Moto G (4)) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3963.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	604

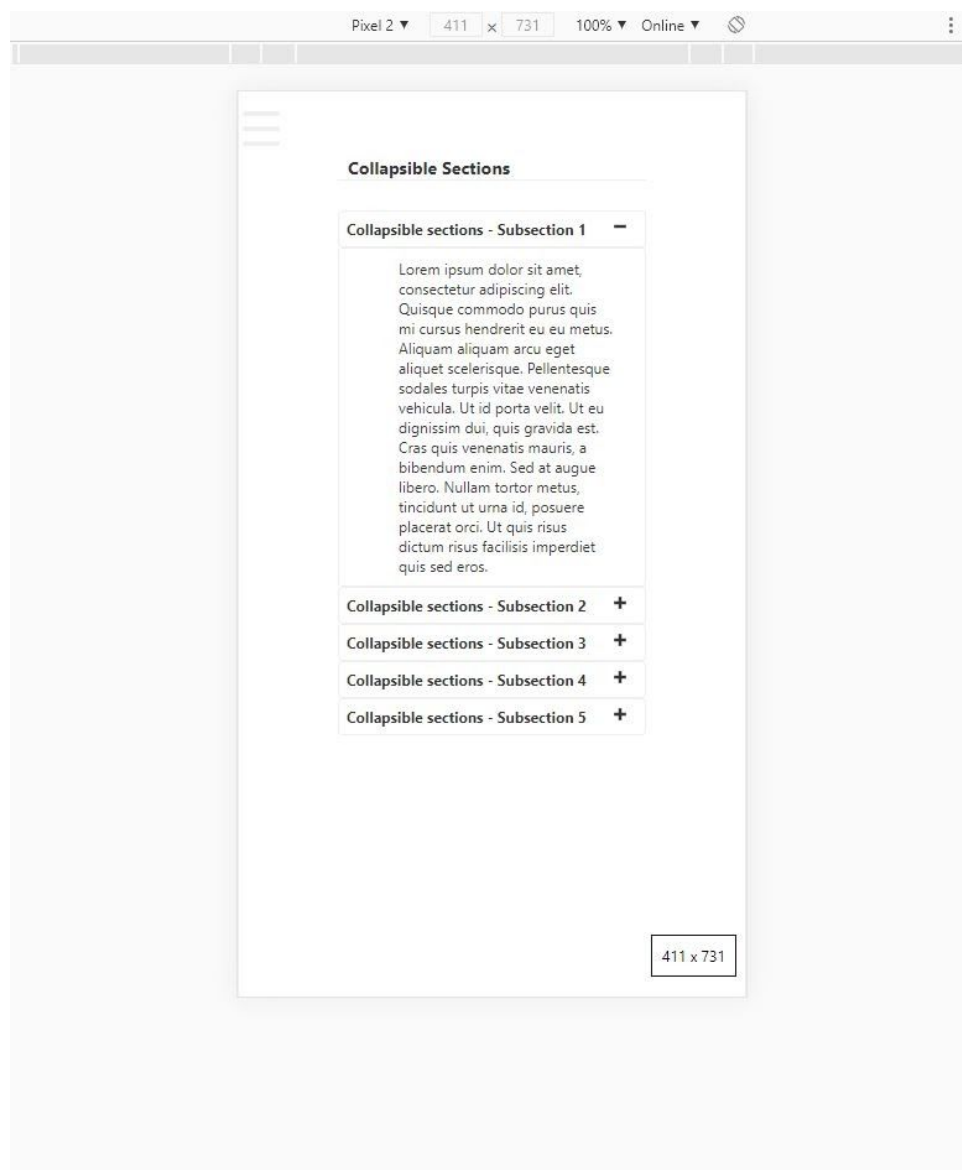
Generated by **Lighthouse** 6.0.0 | [File an issue](#)

Responsywność (Chrome DevTools)

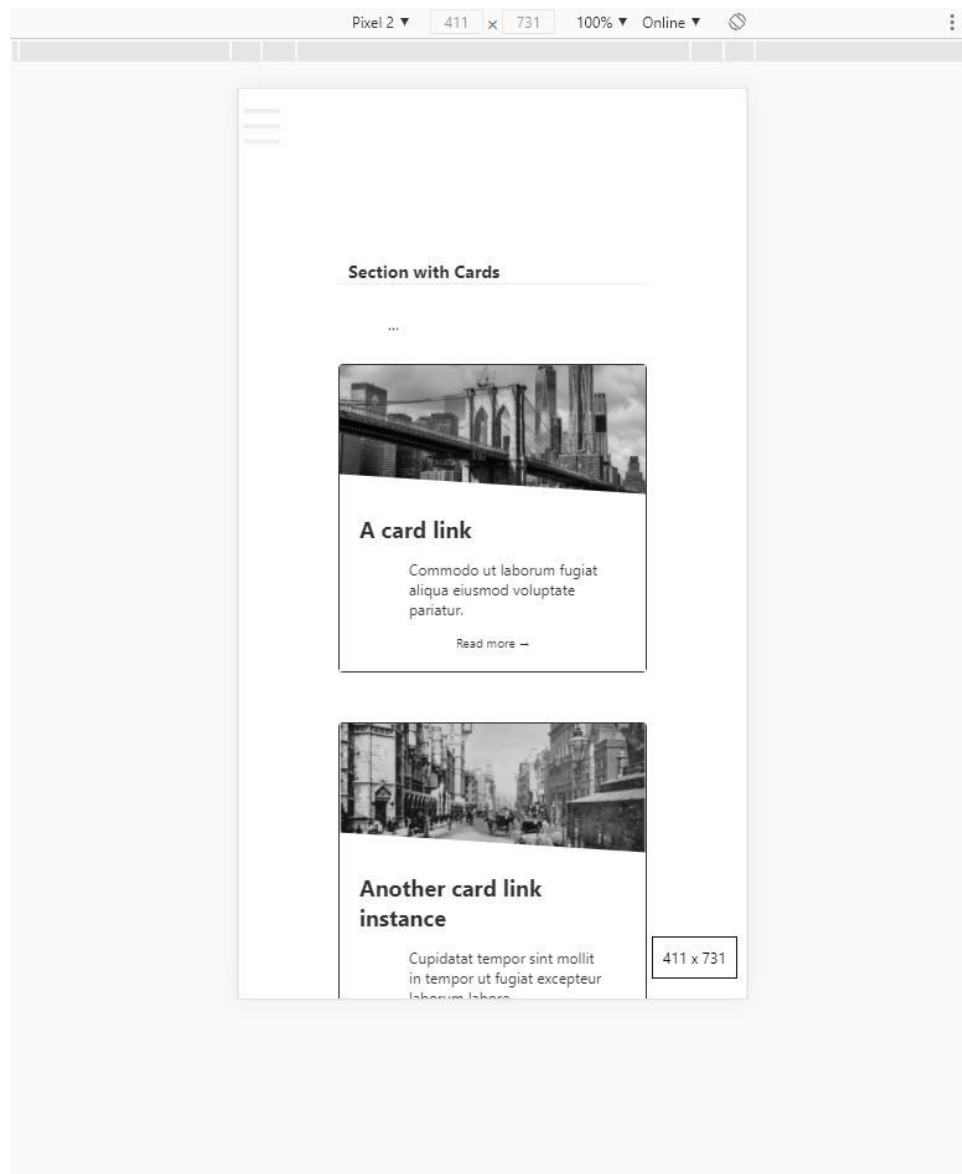
Badanie responsywności wykonano dla dwu wybranych typów wyświetlaczy: telefonu i tabletu zarówno w układzie pionowym jak i poziomym. Istotna w tym przypadku była ocena dostosowania menu jak i układu treści strony względem nowych parametrów wyświetlacza. Otrzymano następujące wyniki:



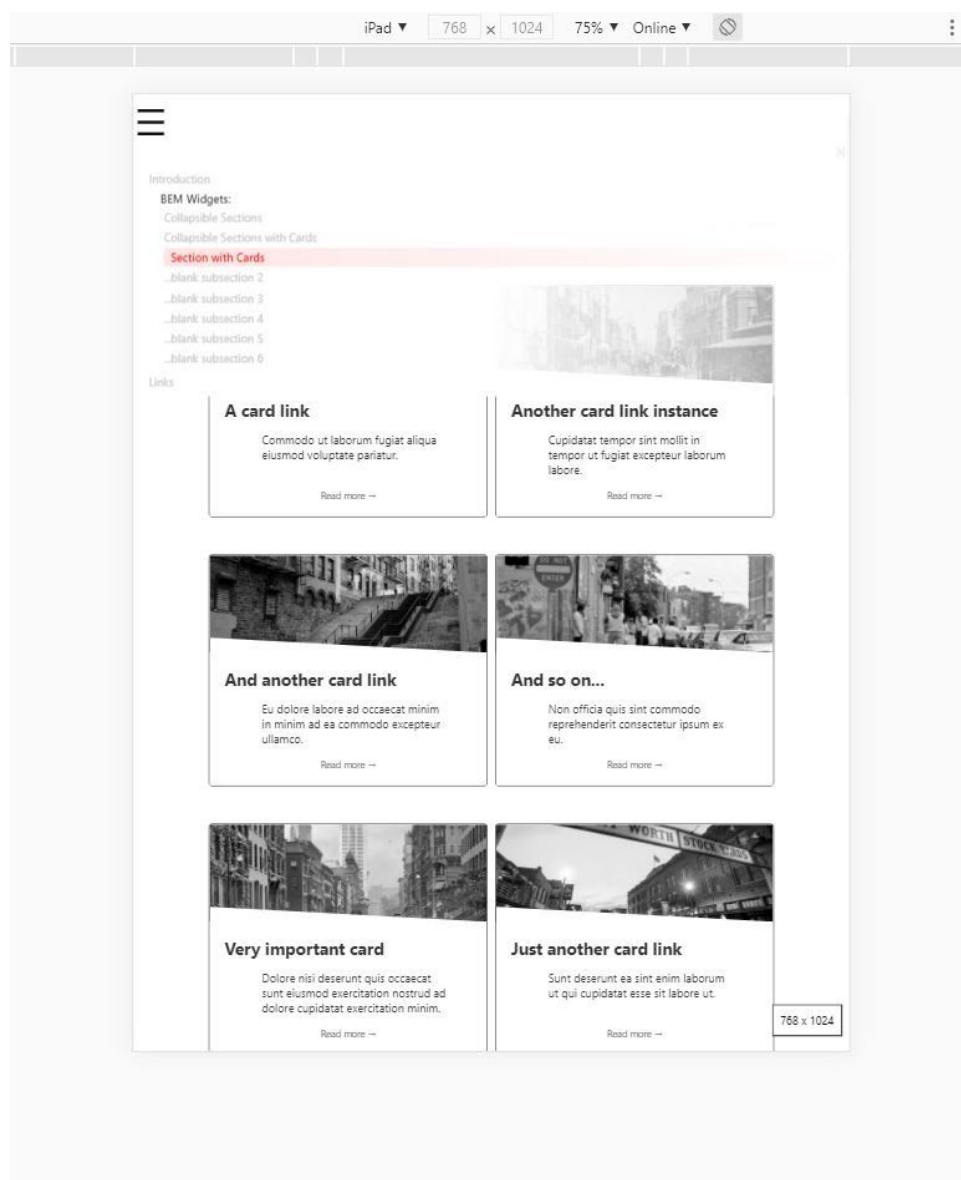
Rys 1 Telefon pixel2 (układ pionowy, rozwinięte menu)



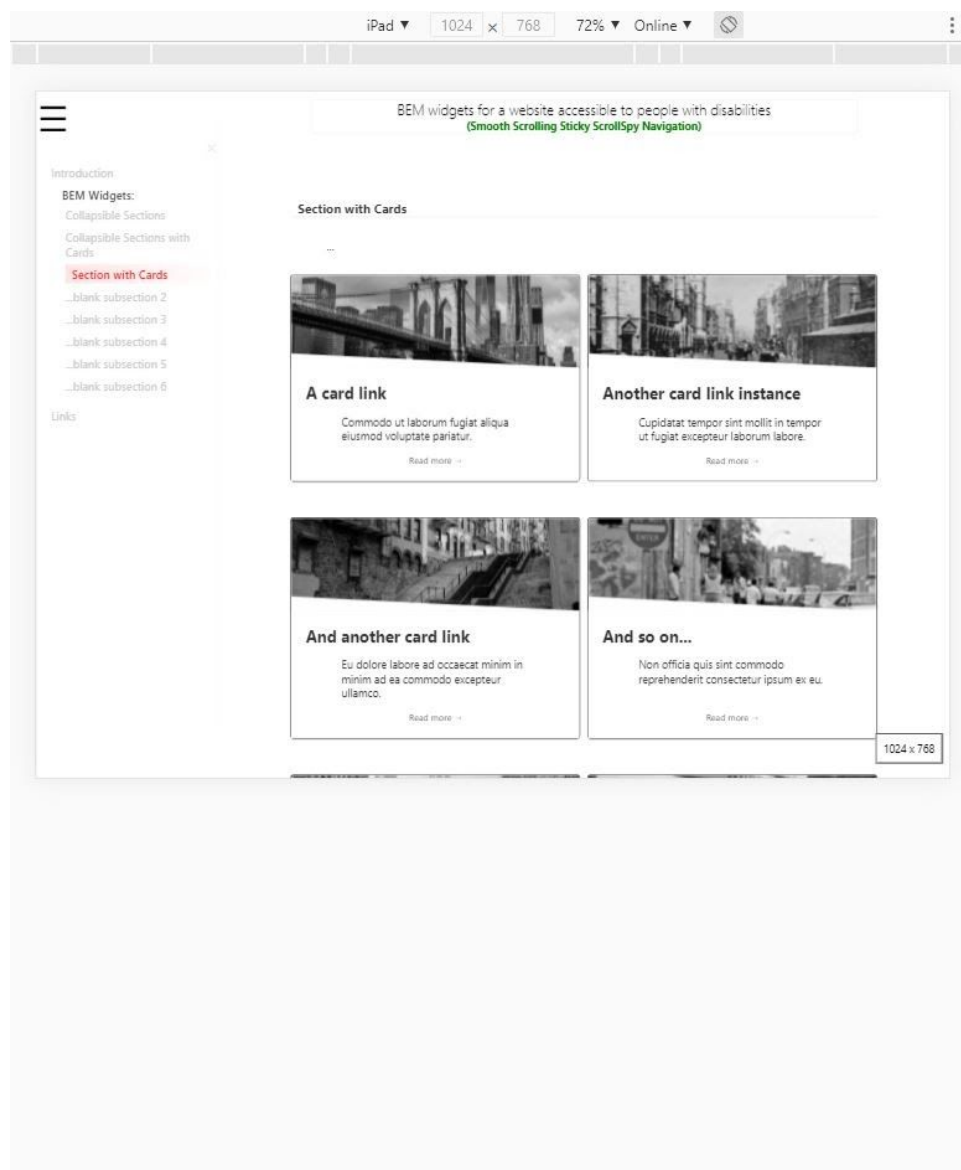
Rys 2 Telefon pixel2 (układ pionowy, widoczny układ rozwijanych sekcji)



Rys 3 Telefon pixel2 (układ pionowy, widoczny rozkład kart linków)



Rys 4 Tablet iPad (układ pionowy, rozwinięte menu, widoczny rozkład kart linków)



Rys 5 Tablet iPad (układ poziomy, układ strony dwukolumnowy, widoczny rozkład kart linków)

9. Podsumowanie

Opracowane widżety/komponenty jak ukazuje praca posiadają cechy konieczne do wdrażania je na strony internetowe dla osób z niepełnosprawnością. Struktura BEM wprowadza zunifikowany język komunikacji, działający pod względem bloków, elementów i modyfikatorów (jak chociażby opis znaczników strony internetowej) które składają się na strukturę komponentu wskazując na jego pełnioną w obrębie dokumentu rolę ale także do organizacji i przechowywania kodu w JavaScript (logika każdego bloku oraz jego opcjonalne elementy i modyfikatory są opisane w osobnych plikach - kod dzielony jest w ten sposób na osobne powiązane znaczeniowo z strukturą BEM projektu części). Mamy zatem elementy których nazwy klas wskazują na charakter jednostki strukturalnej którą pełnią pozwalając na wygodne zarządzanie ich stylem oraz zachowaniem czy to poprzez arkusze stylów CSS czy też poprzez JavaScript.

Semantyka wprowadzona wraz z strukturą BEM dopełniona jest przez semantykę opisującą role, statusy oraz funkcjonalności komponentów stron internetowych takich jak: nawigacja/menu, rozwijane sekcje, karty linków czyli przez semantykę zawartą w technologii ARIA która to dostarcza również informacje strukturalne, które pomagają twórcom identyfikować odpowiednie komponenty, definiuje specjalne role obiektów a dodatkowo dostarcza inne przydatne opcje które mogą zostać użyte w celu aktywacji bądź dezaktywacji obiektu, stanowić punkt informacyjny (np. navigation) czy opisać szereg właściwości, które mogą ulec zmianie (np. `aria-hidden`). ARIA jest przeznaczona tylko dla API (ang. application programming interface) technologii pomocniczych/asystujących AT i nie ma wpływu na DOM czy style. ARIA jest tylko dodawanym do elementu HTML atrybutem, który ma wspomóc technologie pomocnicze AT API (assistive technologies API). Mimo, że sama ARIA nie zmienia stylów to w połączeniu z atrybutem HTML możemy odpowiednio wystylizować elementy ARIA w CSS.

Poza dokumentem HTML na projekt składają się pliki stylów (pliki CSS) oraz pliki skryptów (JavaScript) umożliwiające wdrożenie na stronie internetowej elementów, dzięki którym strona ta może, poza wyświetlaniem statycznych informacji, obsługiwać również zmianę treści odpowiednio do sytuacji/zdarzeń będących wynikiem odpowiedzi na aktywności użytkownika na stronie czyli generowanych w trakcie jej użytkowania interakcji i choć z reguły interakcje te są proste często są to interakcje które nie mają spójnej naturalnej implementacji we wszystkich przeglądarkach co zmusza do ich projektowania przy użyciu JavaScript i WAI-ARIA właśnie.

W oparciu o te kryteria powyższa praca realizuje ideę połączenia, jak się wydaje udanego, założeń metodologii BEM z atrybutami standardu WAI-ARIA.

BIBLIOGRAFIA:

1. **Quick start to BEM methodology**
<https://en.bem.info/methodology/quick-start/>
2. **WAI-ARIA Overview**
<https://www.w3.org/WAI/standards-guidelines/aria/>
3. **Sticky Table of Contents with Scrolling Active States**
https://css-tricks.com/sticky-table-of-contents-with-scrolling-active-states/?fbclid=IwAR1_t5U_erE-0gUrxswpj9UW8zCuVRisAj51UI_49_KzNlc4dNDt-kfdWlrl
4. **Intersection Observer API**
https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API
5. **Inclusive Components**
<https://inclusive-components.design/>
6. **MDN Web Docs**
<https://developer.mozilla.org/en-US/>
7. **w3schools.com**
<https://www.w3schools.com/>