Zasady dostępności serwisów internetowych dla osób niepełnosprawnych

1. Problem dostępności serwisów internetowych

Dostępność serwisu internetowego oznacza umożliwianie dostępu ludzi z dysfunkcjami do nowych technologii, związanych z funkcjonowaniem sieci Internet. Dysfunkcja to czasowa lub trwała niezdolność danej osoby do wykonywania określonych czynności. Powodem dysfunkcji może zaburzenie związane z organizmem danej osoby, może też to być spowodowane ograniczeniami używanego sprzętu i oprogramowania, może być związane z przebywaniem w określonym miejscu, wreszcie powodem może być korzystanie z Internetu przez łącza o niskiej szybkości transmisji.

Dostępny serwis internetowy zatem to taki, z którego mogą korzystać osoby w pełni zdrowie, ale również osoby niepełnosprawne, o różnym stopniu niepełnosprawności. We wstępie do artykułu „What is Web Accessibility" na stronach W3C[1] możemy przeczytać, że zagadnienie dostępności dotyczy w największym stopniu osób z niepełnosprawnością. Istota problemu polega zatem na tworzeniu takich stron, które będzie można łatwo używać – czyli odczytywać, rozumieć, łatwo nimi nawigować i używać dostępnych interakcji ze stroną. Ale oczywiście wprowadzenie takich rozwiązań niesie za sobą dużo więcej korzyści – jest to po prostu stworzenie na tyle elastycznego interfejsu, że będzie on dostępny dla każdego, w każdej sytuacji i przez dowolne urządzenie.

Według danych polskiego Biura Pełnomocnika Rządu ds. Osób Niepełnosprawnych, liczba osób w wieku 15 lat i więcej, zarejestrowanych w naszym kraju jako niepełnosprawne, wynosiła w 2010 r. 3 miliony 400 tysięcy, co stanowi około 10,7% ludności w tym wieku.[2] Nawet zatem, z punktu widzenia czysto komercyjnych projektów jest to atrakcyjny rynek, o który warto zabiegać i który warto zagospodarować. W tym miejscu warto wspomnieć o działaniach, podejmowanych przez instytucje życia publicznego (np. PFRON) na rzecz polepszenia dostępu do zasobów sieci WWW przez osoby niepełnosprawne[3].

Rodzajów niepełnosprawności jest bardzo wiele, również stopień nasilenia choroby może sporo zmieniać w sposobie, w jaki dana osoba korzysta z Internetu, a dodatkowo wiele osób używa urządzeń nietypowych, dostosowanych do indywidualnych potrzeb. Nierzadko też oczekiwania wobec aplikacji osób pełnosprawnych i dotkniętych niepełnosprawnością znacznie różnią się między sobą. Powoduje to, że nie można stwierdzić, iż dany serwis lub aplikacja jest „uniwersalnie dostępna". Można być pewnym, że mimo wysiłków podejmowanych w celu zapewnienia komfortu jej używania, znajdą się osoby, którym korzystanie z tego oprogramowania będzie sprawiać trudność.

Wiadomo również, że nie tylko osoby niepełnosprawne mogą mieć problemy z korzystaniem z Internetu. Ludzie starsi o ograniczonej motoryce i z osłabionym wzrokiem nie zawsze potrafią używać myszki. Dlatego, nawet wiele dobrze widzących osób powiększa tekst, by zwiększyć komfort czytania. Dlatego stwierdzenie „dostępna strona internetowa" jest pewnym uproszczeniem. Zwykle oznacza ono taki serwis, który został przetestowany z wynikiem pozytywnym pod kątem możliwości korzystania z niego jedynie przy pomocy klawiatury oraz przy użyciu określonych czytników ekranowych. Dostępny serwis przynosi korzyści nie tylko niepełnosprawnym. Możliwość operowania klawiaturą przydaje się również osobom mającym większą od przeciętnej biegłość w posługiwaniu się Internetem. Również wyszukiwarki lepiej radzą sobie z poprawnie sformatowanymi dokumentami (np. roboty Google „czytają" strony internetowe w sposób podobny do czytników ekranowych).

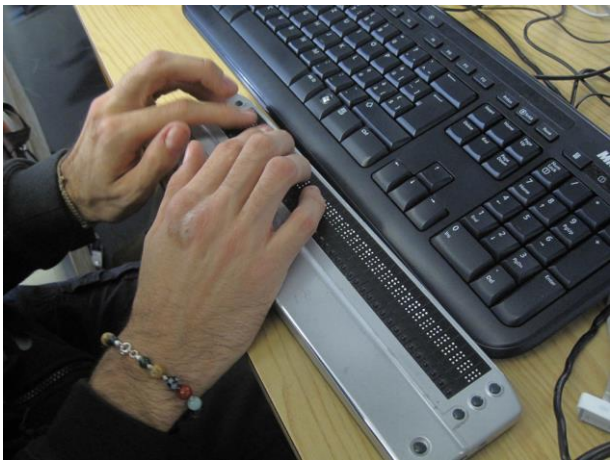
Warto pamiętać, że strona dostępna oznacza przede wszystkim stronę stworzoną zgodnie ze standardami; natomiast zależność ta nie zawsze działa w drugą stronę. Stosowanie standardów i sprawdzonych zasad praktycznych w dużym stopniu uniezależnia serwis internetowy od urządzenia, na którym jest on oglądany. To z kolei powoduje, że serwis taki jest dostępny nie tylko dla technologii wspomagających niepełnosprawnych użytkowników, ale też jest przyjazny dla osób posługujących się różnymi urządzeniami, np. smartphonami lub tabletami.

2. Osoby mające problemy ze wzrokiem a Internet

Projektanci i producenci oprogramowania zwykle tworzą swoje projekty z myślą o osobach widzących, posługujących się myszką lub podobnym urządzeniem wskazującym. Większość interakcji w serwisach projektowana jest z założeniem, że użytkownik najedzie na element i/lub naciśnie przycisk myszki. Tymczasem jest wiele sposobów na komunikowanie się z komputerem.

Osoby niewidome mogą używać czytnika ekranowego – programu, który przetworzy treść wyświetloną na ekranie na głos. Użytkownik niewidomy i niesłyszący może posługiwać się wyświetlaczem Braille'a. Tego typu urządzenie i programy noszą nazwę technologii wspomagających, w skrócie AT (od angielskiego *assistive technologies*).

Oprócz nich wyróżnia się także strategie wspomagające – AS (*assistive strategies*) czyli wszelkie zachowania użytkownika pomagające mu dotrzeć do potrzebnej mu zawartości. Przykładami takich strategii jest powiększanie fragmentów ekranu przez osoby słabo widzące lub po prostu powiększanie tekstu na ekranie np. w przeglądarce.



Ryc. 1: Czytnik Braille'a używany przez osoby niewidome.
Źródło:
http://www.flickr.com/photos/visualpunch/5201

Różne typy niesprawności wymagają różnych technologii i strategii wspomagających. Pewne zabiegi mogą go uczynić dostępnym dla określonej grupy osób, podczas gdy inne wciąż będą miały problemy z niektórymi obszarami. Percepcja serwisu może zmieniać się nie tylko w zależności od rodzaju niepełnosprawności, ale także jej nasilenia. Dlatego też tak trudne jest przeprowadzenie testów automatycznych dostępności.

2.1. Osoby niewidome

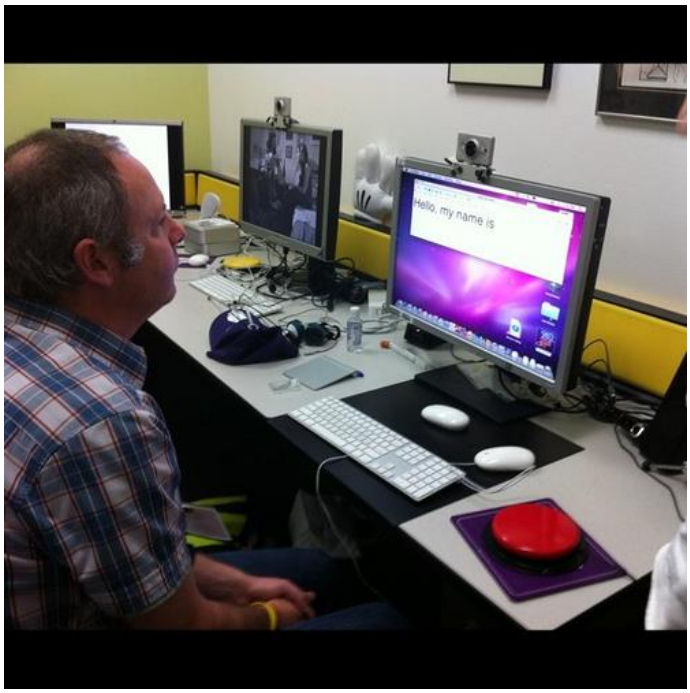
Użytkownicy niewidzący nie mogą korzystać z myszki, ponieważ nie są w stanie określić pozycji kursora na ekranie. Dostęp do treści wyświetlonej na ekranie zapewnia im specjalne oprogramowanie tzw. czytniki ekranowe (*screen readers*). Niektórzy posługują się też urządzeniami przetwarzającymi tekst na alfabet Braille'a.

Czytniki ekranowe to programy, które za pomocą syntezatora mowy odczytują treść dowolnego dokumentu. Urządzenia Braillowskie to rodzaj dotykowego wyświetlacza – ruchome wypustki są odpowiednio wypychane lub chowane, tworząc znaki, które użytkownik może odczytać opuszkami palców. Zwykle czytnik taki pozwala na wyświetlenie ok. 30 znaków, po czym po naciśnięciu odpowiedniego klawisza wypustki zmieniają swoje ułożenie i pojawia się następna porcja tekstu. Urządzenia takie są niestety drogie. To wydatek rzędu od 1000 do nawet 15000 USD[4].

Urządzenia tego typu zupełnie zmieniają percepcję dokumentów na ekranie. Osoba widząca patrzy na ekran całościowo. Analizuje wszystkie elementy jednocześnie i na podstawie ich wzajemnego ułożenia jest w stanie odnaleźć istotne fragmenty, gdziekolwiek się znajdują. Czytanie tekstu na głos lub jego wyświetlanie na urządzeniu, które potrafi pokazać jedynie niewielki jego fragment, wymusza zupełnie inny, liniowy dostęp do treści dokumentu. Wizualnie można przyrównać go do patrzenia na ekran przez wąską rurkę (*the soda straw approach*) – naraz widać tylko jedno słowo, a jedyną szansą zrozumienia treści jest czytanie jej od początku do końca. Rzut oka na wyrwany z kontekstu wyraz gdzieś na środku nie daje żadnych informacji.

Przetwarzanie wizualnej strony internetowej przez czytnik to w istocie przeniesienie dwuwymiarowej „kartki papieru” do postaci linearnego tekstu. Odczytanie w ten sposób form z natury dwuwymiarowych, jak tabela jest trudnym zadaniem dla użytkownika. Tylko prawidłowa struktura wykorzystująca semantyczne elementy, jak nagłówki kolumn (**<th>**) czy oznaczenia części tabeli (omówione powyżej znaczniki **<thead>**, **<tbody>**, **<tfoot>**) pozwoli czytnikowi prawidłowo zinterpretować jej treść. Podobnie jest w przypadku formularzy na stronach – zwykle każdemu polu takiego formularza towarzyszy etykieta wyjaśniająca, co ma się w nim pojawić (np. nazwisko, płeć, data urodzenia). Jeżeli etykieta nie jest stworzona za pomocą elementu <label> lub nie posiada atrybutu for tworzącego funkcjonalne powiązanie między nią a polem, użytkownik, który nie widzi ich wzajemnego położenia na stronie będzie miał problem, aby zorientować się, co ma wpisać w daną kontrolkę.

Jeżeli tabele stosowane są do rozmieszczenia treści na stronie, użytkownik usłyszy wiele zbędnych informacji dotyczących wierszy i kolumn. Również kolejność odczytywania treści może zostać zaburzona, bo domyślnie czytnik przechodzi komórki wierszami, nie kolumnami. Z tego powodu strony oparte na tabelach uważane są za niedostępne.



Ryc. 2: Użytkownik korzystający z powiększenia elementów na ekranie
Źródło:
http://www.flickr.com/photos/mjmonty/5087474397/

Czytniki ekranowe do pewnego stopnia mogą symulować wzrokowe skanowanie treści. Osoby widzące przeglądając tekst szukają wyróżniających się elementów. Są to zwykle grafiki, nagłówki, wypunktowania, pogrubienia, a na stronach internetowych także linki. Czytnik analizuje strukturę dokumentu i prezentuje użytkownikowi zestawienie tego typu elementów, czyli listę nagłówków, odnośników czy obrazków na stronie, pozwalając pomijać niektóre części i szybciej przechodzić do istotnych fragmentów.

Ustawienia programu oczywiście można dostosowywać do indywidualnych potrzeb. Co ciekawe, osoby używające czytników zwykle mają prędkość odczytywania tekstu ustawioną na tak wysokim poziomie, że jest on kompletnie niezrozumiały dla kogoś niewprawionego w tej technice.

Najpopularniejsze czytniki ekranowe to JAWS (59,2% w 2010), Window-Eyes (11,2%), NVDA (8,6%) i VoiceOver (9,8%)[5]. Trzy pierwsze działają pod systemem Windows, czwarty wbudowany jest w system Mac OS X. JAWS[6] i Windows-Eyes[7] są płatne (JAWS to ok 800-1000 USD za licencję, Window-Eyes to wydatek rzędu 300-900 USD), NVDA[8] jest darmowym projektem open source, co znacznie zwiększa jej popularność. VoiceOver[9] jest częścią systemu, więc również nie wymaga dodatkowych opłat. Co ciekawe, VoiceOver jest również częścią systemu iOS, i jest bardzo chwalony przez niewidomych użytkowników iPhone'a. Wszystkie wymienione programy w swoich najnowszych wersjach są dostępne w języku polskim.

Z rozmów z osobami niewidomymi wynika, że większość stron sprawia problem czytnikom, szczególnie te, które używają JavaScriptu do tworzenia złożonych elementów nawigacyjnych i dynamicznych manipulacji drzewem DOM. Mechanizm działania czytników opiera się bowiem na tworzeniu obrazu treści w momencie wejścia na stronę. Zmiana części strony przez JavaScript nie

powoduje odświeżenia tego obrazu, więc są one niewidoczne dla czytnika. Przeładowanie buforu następuje dopiero po wymuszeniu zaznaczenia (zdarzenie focus), na jakimś elemencie, który został dodany.

Duże nadzieje pokładane są w WAI-ARIA, która pomaga nadać odpowiednie znaczenie semantyczne dowolnym elementom i która wspiera regiony dynamicznie aktualizowane JavaScriptem.

2.2. Osoby słabo widzące

Tacy użytkownicy mogą używać programu powiększającego ekran zwanego popularnie lupą. Praktycznie każdy system operacyjny ma wbudowane takie narzędzie, można też kupić aplikacje o bardziej rozbudowanej funkcjonalności. Użytkownik może też po prostu powiększać czcionkę na ekranie.

W obu przypadkach na ekranie widoczny będzie jedynie fragment strony. Na pewno warto zatem sprawdzić, jak skaluje się tekst na ekranie. Zgodnie z konwencją powiększenie o 200% nie powinno skutkować pojawieniem się poziomego suwaka.

2.3. Osoby o zaburzonym widzeniu barw

Wady wzroku polegające na nie rozróżnianiu kolorów stosunkowo rzadko są przyczyną utrudnień w korzystaniem z aplikacji internetowych. Głównym źródłem problemów dla takich użytkowników są elementy wyróżnione na stronie jedynie za pomocą koloru, np. linki. Pozbawione podkreśleń niebieskie odnośniki są praktycznie niewidoczne dla osoby z tritanopią (nie rozpoznawanie barwy żółtej i niebieskiej).[10]

Istnieje wiele stron i programów, które pozwalają sprawdzić, jak wygląda określone zestawienie kolorów przy różnych rodzajach wady. Ponieważ jej nasilenie może być bardzo różne, najbezpieczniej stosować oprócz kolorów dodatkowe wyróżniki graficzne przy interaktywnych elementach – np. podkreślenie odnośników.

3. Standardy tworzenia dostępnych stron internetowych[11,12]

3.1. W3C

The World Wide Web Consortium (często używa się skrótu W3C) to międzynarodowa społeczność (złożona z organizacji i instytucji członkowskich, pracowników oraz pasjonatów) finansowana ze składek członkowskich, grantów i indywidualnych dotacji. Proponuje ona standardy (protokoły i wytyczne) zapewniające długodystansowy rozwój Internetu. Założona została 1 października 1994 roku przez Tima Berners-Lee, twórcę WWW, autora pierwszej przeglądarki internetowej i serwera WWW.

Należy podkreślić, że już ogólne zasady tworzenia stron WWW, obecnie określane jako HTML5 i CSS3, znacząco poprawia dostępność stron WWW. I tak na przykład, wykorzystanie nowych atrybutów dla znaczników HTML5, ułatwiających pracę z czytnikami ekranu, a także podawanie wszystkich wielkości w skali względnej znacząco poprawia dostępność serwisu. Dodatkowo, konsorcjum W3C opracowało i proponuje do stosowania szereg dodatkowych zaleceń, których uwzględnienie ma jeszcze wyraźniej poprawić dostępność stron WWW dla szerokich rzesz odbiorców. Traktują o tym kolejne podrozdziały.

3.2. WAI

WAI[13] (z ang. Web Accessibility Initiative) rozpoczęło swoją działalność w 1997r, stając się częścią konsorcjum W3C. Jest to grupa osób i organizacji zaangażowanych w rozwój i promowanie

standardów, służących poprawie dostępności usług internetowych (głównie stron WWW) dla osób niepełnosprawnych. W wyniku prac WAI powstały m.in. następujące dokumenty związane z dostępnością stron internetowych:

- Zbiór zasad dla oprogramowania, służącego do tworzenia stron internetowych, **ATAG**[14] (z ang. Authoring Tool Accessibility Guidelines).
- Zbiór zasad dotyczących projektowania przeglądarek internetowych, **UAAG**[15] (z ang. User Agent Accessibility Guidelines).
- Zbiór zasad dotyczących projektowania stron internetowych, przystosowanych do wymagań osób niepełnosprawnych, **WCAG**[16] (z ang. Web Content Accessibility Guidelines), omówiony w następnym podrozdziale.
- Opis zasad i narzędzi, służących do projektowania i budowy stron internetowych, w tym i z dynamiczną zawartością, opartą na technologiach dynamicznego HTML (DHTML), AJAX, itp. Skrótowa nazwa tego opracowania to **WAI-ARIA** (z ang. WAI Accessible Rich Internet Applications). Temu zbiorowi zasad poświęcony jest osobny rozdział tego opracowania.

3.3. WCAG[17]

WCAG jest opracowanym przez WAI zbiorem zasad projektowania, które zapewnią w miarę powszechny dostęp do treści na stronach WWW, zwłaszcza dla osób z dysfunkcjami. Dokument składa się ze zbioru wytycznych, dotyczących projektowania struktury dokumentu sieciowego, tak by była łatwa w odbiorze i kontrolowaniu, a przede wszystkim dostępna dla osób niepełnosprawnych. Obecnie zakończono pracę nad wersją 2.0 dokumentu, jednocześnie nadając jej status oficjalnej rekomendacji W3C. Oprócz zasadniczego dokumentu WCAG 2.0, w skład pakietu WCAG wchodzą również opracowania dodatkowe, np. przewodnik ułatwiający zrozumienie i stosowanie wytycznych czy opis technik, zalecanych do stosowania.

WCAG skierowane jest przede wszystkim do twórców stron internetowych, autorów oprogramowania i narzędzi oceniających dostępność. Opracowanie zawiera 12 wytycznych pogrupowanych w 4 nadrzędne zasady: **dostrzegalność** (ang. *perceivable*), **sprawność** bądź **operacyjność** (ang. *operable*), **zrozumiałość** (ang. *understandable*) oraz **solidność** (ang. *robust*). Zastosowanie każdej z nich można sprawdzić przez opisane kryteria sukcesu na trzech możliwych poziomach: A, AA i AAA. Zaleca się, aby strona internetowa spełniała choćby najniższy poziom A. Drugim, rozszerzonym stopniem dostępności, jest poziom AA. Stosowanie zasad z tego poziomu do całej zawartości serwisu jest rozsądne i zalecane, o ile tylko jest możliwe do osiągnięcia. Natomiast poziom AAA, który zapewnia najwyższy poziom dostępności, jest zazwyczaj niemożliwym do spełnienia dla niektórych elementów serwisu. W opracowaniu WCAG znajduje się szczegółowa lista wytycznych, wraz z podziałem względem poziomów dostępności. Podane zasady pokrótce scharakteryzowano w kolejnych podrozdziałach.

3.3.1. Dostrzegalność

W ogólnym przypadku, cecha oznacza, iż informacje i części składowe serwisu WWW, w tym zwłaszcza interfejsu użytkownika są przedstawione w sposób umożliwiający percepcję przez użytkowników:

- Tekst alternatywny dla nietekstowych elementów;
- alternatywny kanał przekazu dla treści audio i video;
- oddzielenie prezencji od struktury, adaptowalność;
- oddzielenie głównych informacji od tła – rozróżnialność.

3.3.1.1. Tekst alternatywny

Poziom A: jako powszechnie obowiązującą zasadę należy przyjąć, iż dla każdej treści innych niż tekstowa – obraz, wykres, element dźwiękowy lub inny nietekstowy element – należy zapewnić tekst, jako alternatywę, zawierająca taką samą bądź przynajmniej zbliżoną zawartość informacyjną i funkcjonalną. Należy również dążyć do tego, aby ów ekwiwalent można było zmieniać w inne formy, których potrzebują ludzie. Takimi formami mogą być np. powiększona czcionka, alfabet Braille'a, słowa, symbole, czy tekst dostępny dla czytnika ekranu.

Omówiona powyżej zasada dopuszcza kilka, ściśle określonych wyjątków.

- Zawartość pełniąca rolę kontrolną lub zawierająca dane wprowadzone przez użytkownika.
- Media z treścią zmienną w czasie.
- Dane zawierające test lub ćwiczenie.
- Dane służące do wytworzenia konkretnych wrażeń zmysłowych.
- Dane potwierdzające, że przeglądanie wykonuje osoba, a nie komputer, tzw. CAPTCHA.
- Dane realizujące funkcje ozdobne.

W przypadku czterech pierwszych punktów należy opisać treść zawartości, w kolejnym punkcie należy opisać cel danych, a w przypadku ostatniego punktu, dane powinny pełnić taką rolę, by technologie wspomagające mogły je zignorować.

3.3.1.2. Alternatywa dla treści medialnej zmiennej w czasie

Poziom A: dla treści audio i wideo należy zapewnić taką alternatywę, która w pełni opisuje treść pliku dźwiękowego bądź audiowizualnego, o ile nie jest to treść nadawana „na żywo”.

Poziom AA: należy zapewnić napisy dla treści audio nadawanej na żywo oraz dodatkową ścieżkę dźwiękową (lub inne media alternatywne, uprzednio nagrane) do wcześniej nagranego pliku wideo.

Poziom AAA: dla wszystkich materiałów dźwiękowych powinna być dostępna ich wersja w języku migowym, dla wcześniej nagranych plików wideo powinien być dostępny rozszerzony opis dźwiękowy, alternatywna treść i forma przekazu dla wszystkich, przekazywanych na żywo treści audio i wideo.

3.3.1.3. Adaptowalność

Poziom A: informacje, struktura i relacje przekazywane w treści mogą być programistycznie zdefiniowane w kodzie (kod semantyczny) lub dostępne w postaci tekstu; jeśli jest to istotne, kod strony ustala kolejność przekazywania informacji; brak zależności zrozumienia instrukcji od jednej tylko cechy sensorycznej (kształt, rozmiar, orientacja, dźwięk).

3.3.1.4. Rozróżnialność

Poziom A: oprócz zwracania uwagi na daną informację, kolor pełni funkcje dodatkowe (wskazuje działanie bądź podpowiedź, wyróżnia elementy wizualne); dla uruchamianego na stronie internetowej dźwięku (dłuższego niż 3 sekundy), powinna istnieć możliwość jego wyłączenia i niezależnej od ustawień systemu regulacji głośności.

Poziom AA: prezentowany tekst powinien mieć kontrast co najmniej 4,5:1 (wyjątki: duży rozmiar tekstu, tekst drugorzędny, logotypy); możliwość zmiany rozmiaru tekstu do 200% bez użycia narzędzi dodatkowych; należy unikać grafik z tekstem.

Poziom AAA: prezentowany tekst powinien mieć kontrast co najmniej 7:1; należy zapewnić niski poziom głośności dźwięku w tle lub brak takowego dźwięku; dla bloków tekstu należy zapewnić następujące możliwości: wybór kolorów tekstu i tła przez użytkownika, brak justowania tekstu, interlinia wynosi co najmniej 1,5 wiersza, powiększenie tekstu do 200% nie wymaga przewijania poziomego.

3.3.2. Sprawność

Oznacza, iż zawartość serwisu musi być wystarczająco solidna, by mogła być rzetelnie interpretowana przez różne oprogramowanie (przeglądarki, odtwarzacze multimedialne), włączając w to technologie wspierające:

- wszystkie funkcje dostępne za pomocą klawiatury;
- odpowiednia ilość czasu na interakcje użytkownika z serwisem;
- unikanie efektów, które mogą powodować napady padaczki;
- czytelna nawigacja, mechanizmy wyszukiwania.

3.3.2.1. Dostępna klawiatura

Poziom A: poza określonymi wyjątkami (gdy zadanie wymaga określonej siły bądź długości czasowej naciśnięcia klawisza), cała funkcjonalność serwisu może być uzyskana za pomocą klawiatury, łącznie z cofaniem kroków nawigacyjnych.

Poziom AAA: cała funkcjonalność – bez wykluczeń – serwisu może być uzyskana za pomocą klawiatury.

3.3.2.2. Stosowny czas na interakcję użytkownika z serwisem

Poziom A: należy zapewnić użytkownikowi możliwość dostosowania, zmiany bądź wyłączenia limitów czasu, przewidzianych na poszczególne czynności.

Poziom AAA: jak w punkcie A plus możliwość działania serwisu bez ograniczeń czasowych; możliwość przekładania przerw w korzystaniu z serwisu oraz możliwość ponownego uwierzytelnienia bez utraty danych.

3.3.2.3. Unikanie przyczyn wystąpienia ataków padaczki

Poziom A: strona nie zawiera niczego, co migałoby częściej niż trzy razy na sekundę, a miganie zajmuje mniej niż 25% pola widzenia.

3.3.2.4. Czytelna nawigacja i mechanizmy wyszukiwania

Poziom A: w serwisie istnieje możliwość obejścia tych bloków zawartości, które powtarzają się na wielu stronach; dla każdej strony serwisu zdefiniowano jej tytuł, określający cel i treść strony; cel każdego zdefiniowanego programistycznie odnośnika wynika z jego treści i kontekstu na stronie WWW.

Poziom AA: więcej niż jedna metoda nawigacji po serwisie (np. menu, mapa strony, etc.); nagłówki i etykiety na stronie opisują konkretny cel; przy korzystaniu z klawiatury, interfejs użytkownika zapewnia widoczność stanu aktywacji klawiatury (np. podkreślenie, obramowanie, etc.).

3.3.3. Zrozumiałość

Oznacza, że komponenty interfejsu i nawigacji powinny być sprawne, przystępne w użytkowaniu, a to oznacza:

- czytelny i zrozumiały tekst;
- przewidywalność;
- mechanizmy pomocy.

3.3.3.1. Czytelność

Poziom A: serwis zapewnia możliwość zdefiniowania języka dla całości.

Poziom AA: serwis zapewnia możliwość zdefiniowania języka dla całości i dla poszczególnych części.

Poziom AAA: serwis zawiera definicje i wyjaśnienia słów, terminów i skrótów, których zrozumienie nie jest powszechne; serwis zawiera treść uzupełniająca, gdy zrozumienie serwisu wymaga wiedzy powyżej określonego poziomu (np. powyżej poziomu gimnazjum); serwis posiada możliwość prezentacji wymowy trudniejszych słów i terminów.

3.3.3.2. Przewidywalność

Poziom A: dla składnika serwisu powodującego wskazanie tegoż elementu, nie następuje zmiana kontekstu; podobnie, wprowadzenie danych wejściowych w interfejsie przez użytkownika również nie powodują zmiany kontekstu serwisu.

Poziom AA: nawigacja po serwisie jest spójna, tzn. mechanizmy nawigacyjne rożnych części serwisu zawierają swoje identyczne elementy w tej samej kolejności; komponenty serwisu o tej samej funkcjonalności są identycznie oznaczane.

Poziom AAA: serwis zawiera możliwość włączenia i wyłączenia pewnych swoich części, „na życzenie”.

3.3.3.3. Mechanizmy pomocy

Poziom A: dane wejściowe są automatycznie weryfikowane, a błędy są przekazywane

użytkownikowi; informacje o konieczności wprowadzania danych przez stosowne etykiety lub instrukcje.

Poziom AA: serwis zawiera sugestie co do możliwości korekty popełnianych błędów; serwis minimalizuje zagrożenia (prawne, finansowe, informatyczne) związane z wprowadzaniem błędnych informacji, przez odwracalność (możliwość wycofania) zgłoszenia, sprawdzanie danych, końcowe potwierdzanie wprowadzonych danych.

Poziom AAA: przy wypełnianiu formularzy z danymi, dostępny jest tekst pomocniczy z wyjaśnieniami; zapobieganie błędom, opisane dla poziomu AA.

3.3.4. Solidność

Oznacza, że informacje i czynności, możliwe do wykonania dzięki interfejsowi, powinny być solidnie zbudowane, a to oznacza :

- kompatybilność z obecnymi i przyszłymi technologiami.

Poziom A: przestrzeganie reguł tworzenie poprawnego kodu, np. zamykanie znaczników, poprawne zagnieżdżanie, brak zduplikowanych atrybutów i identyfikatorów; w pełni poprawne generowanie elementów formularzy z ich atrybutami (nazwa, rola, wartość).

Należy tutaj podkreślić, że konsekwentne i zgodnie ze specyfikacjami stosowanie reguł języka HTML i kaskadowych arkuszy stylów, zapewnia sukces dla opisywanego warunku.

4. WAI-ARIA – dodatkowa warstwa semantyczna dla czytników ekranowych

WAI-ARIA – rozwijane jako *Web Accessibility Initiative – Accessible Rich Internet Applications* (inicjatywa na rzecz dostępnego Internetu – dostępne bogate aplikacje internetowe) to – w odróżnieniu od WCAG 2.0 – zestaw konkretnych narzędzi, których stosowanie ma na celu poprawę dostępności aplikacji internetowych przez stworzenie dodatkowej warstwy niosącej informacje istotne dla technologii wspomagających.

WAI-ARIA definiuje zestaw atrybutów, które, dodane do kodu HTML, wzbogacają go o semantykę i różnego rodzaju metadane, pozwalając na lepsze zrozumienie zawartości.

W tradycyjnym kodzie HTML elementy typu **`<div>`** czy **`<span>`** odpowiednio opisane za pomocą arkuszy CSS mogą stać się praktycznie wszystkim. Czytnik ekranu nie jest z kolei w stanie odróżnić, czy dany element jest tylko pojemnikiem na tekst, czy może złożonym elementem sterującym, np. drzewem katalogów. Dodanie roli WAI-ARIA umożliwia powiedzenie niewidomemu użytkownikowi, z czym tak naprawdę ma do czynienia.

```
<div>Dla czytnika ekranowego jestem bez znaczenia</div>
<div role="main">Czytnik wie, że to zwykła treść</div>
<div role="tree">
     <div role="treeitem">Czytnik rozpozna strukturę drzewa</div>
</div>
<div role="button">Rozpoznaje interaktywny element</div>
```

WAI-ARIA nie implementuje natomiast żadnych zachowań związanych z rolą pełnioną przez obiekt. Oznaczenie elementu jako drzewo nie dodaje automatycznie możliwości nawigacji po nim za pomocą klawiatury. Pozwoli natomiast czytnikowi rozpoznać typ i dostosować do tego komunikat głosowy. Podobnie rola **`button`** nie powoduje, że element staje się automatycznie możliwy do zaznaczenia za pomocą klawisza **Tab**, nie będzie na nim także działał klawisz **Enter**.[18] Zaimplementowanie tych funkcjonalności należy do osoby tworzącej dany widget.

Bardzo istotnym udogodnieniem związanym z WAI-ARIA jest obsługa regionów zmienianych przez JavaScript. Rozwiązuje ona tym samym największy problem dynamicznych aplikacji webowych wykorzystujących np. AJAX-a.

Wsparcie dla WAI-ARIA jest zróżnicowane, ale obecnie ani przeglądarki, ani technologie wspomagające nie wspierają tej specyfikacji w 100%.[19] Mimo to stosowanie atrybutów WAI-ARIA w kodzie jest zalecane ze względu na ogólną akceptację technologii przez zainteresowanych i stopniowo rosnące wsparcie ze strony oprogramowania.

Elementy WAI-ARIA dodane do kodu HTML nie powodują żadnych zmian w sposobie wyświetlania się strony. Te, których obsługa nie jest zaimplementowana, są ignorowane przez parser, pozostałe są zwykle udostępniane technologiom wspomagającym przez tzw. *accessibility API*, czyli interfejs używany przez to oprogramowanie do odczytywania zawartości ekranu.

Atrybuty WAI-ARIA można podzielić na role, własności i stany. W następnych podrozdziałach znajduje się omówienie poszczególnych typów, razem z przykładowymi wartościami.

4.1. Role

Role są niezmienne w czasie. Określają, czym dany element jest i jaką pełni funkcję. Dzięki rolom elementy HTML zaczynają pełnić funkcję punktów orientacyjnych (*landmarks*) dla technologii wspomagających – dodanie roli **`navigation`** do elementu listy **`<ul>`** powoduje, że czytnik ekranowy „wie”, że nie jest to przypadkowa lista linków, ale element nawigacji na stronie.

```
<ul role="navigation">
  <li><a href="/home">Strona główna</a></li>
  <li><a href="/products">Produkty</a></li>
  <li><a href="/contact">Kontakt</a></li>
</ul>
```

Role są podzielone na: opisujące na stronie regiony, po których można nawigować (*landmark*), takie, które opisują strukturę zawartości (*document structure*) oraz takie, które definiują elementy interfejsu użytkownika (*widget*).

Najważniejsze i najlepiej zaimplementowane są role typu *landmark*. Pozwalają one na sprawną nawigację po częściach strony. Czytniki, które wspierają tę funkcjonalność (JAWS, NVDA i VoiceOver na iPhonie i iPadzie[20]) potrafią użyć ich do przełączania się między elementami zawierającymi nagłówek strony, nawigację, główną treść czy informacje dodatkowe.

Implementacja pozostałych ról w przeglądarkach i technologiach wspomagających jest bardzo zróżnicowana. Internet Explorer 9 i najnowsze wersje Firefoxa obsługują praktycznie wszystkie z nich. Silnik WebKit (wykorzystywany w Safari i Chrome) rok temu obsługiwał niecałą połowę[21]. Testowanie wsparcia dla tych elementów jest dodatkowo utrudnione przez fakt, że czytniki ekranowe również ich nie implementują.

Zestaw wartości atrybutu role liczy ponad 70 elementów[22]. Poniższa tabela zawiera opis kilku z nich, obrazując ich różnorodność:

Rola	**Opis**
`application` (landmark)	Region zdefiniowany jako aplikacja w odróżnieniu od domyślnego trybu 'dokument'.
`article` (document structure)	Sekcja na stronie, która zawiera odrębną, niezależną całość.
`banner` (landmark)	Sekcja zawierająca główny tytuł lub nagłówek serwisu.

Rola	Opis
button (widget)	Element, do którego dowiązana jest jakaś interakcja użytkownika.
complementary (landmark)	Część treści związana z główną zawartością, ale stanowiąca odrębną całość. (Np. widget z informacjami o pogodzie przy artykule w portalu informacyjnym)
contentinfo (landmark)	Metadane strony (Przypisy, informacje o polityce prywatności, prawach autorskich itp.)
dialog (widget)	Dodatkowe okno aplikacji, pojawiające się aby wymóc na użytkowniku podjęcie jakiejś akcji – wpisania potrzebnej treści lub dokonania wyboru czy zaakceptowania danej opcji.
link (widget)	Interaktywny element zawierający odnośnik do jakiegoś zasobu. Jego aktywowanie przenosi użytkownika do tego zasobu.
list (document structure)	Grupa nieaktywnych, powiązanych ze sobą elementów
main (landmark)	Główna treść dokumentu
menu (widget)	Lista elementów do wyboru. Typ złożony, wymagający potomnych elementów menuitem
menuitem (widget)	Jedna z opcji w menu
navigation (landmark)	Lista interaktywnych elementów (zwykle odnośników) służących do nawigowania po serwisie.
note (landmark)	Treść pomocnicza dla głównej zawartości
scrollbar (widget)	Obiekt kontrolujący przewijanie widocznej zawartości
search (landmark)	Wyszukiwarka na stronie
slider (widget)	Interaktywny element pozwalający użytkownikowi wybrać wartość z zadanego przedziału.
tab (widget)	Etykieta służąca do przełączenia zawartości zorganizowanej w formie kart z fiszkami.
tablist (widget)	Lista elementów tab. Typ złożony.
tabpanel (widget)	Treść związana z danym elementem tab.
tooltip (widget)	Dodatkowe informacje o elemencie. Opis danego elementu.

Tab. 1: Role WAI-ARIA – przykłady

4.2. Stany i własności

Stany i własności opisują zachowanie elementu w danym momencie. Z założenia są to wartości dynamiczne, przy czym własności (*properties*) zmieniają się stosunkowo rzadziej niż stany (*states*).[23]

Niektóre stany i własności mogą być wykorzystywane przez dowolne elementy HTML. Inne wymagają elementu o określonej roli.

Oba typy przyjmują formę atrybutu elementu HTML poprzedzonego przedrostkiem aria-

Poniżej znajduje się przykład użycia stanu **aria-hidden** służącego do ukrywania i „pokazywania" zawartości elementu technologiom wspomagającym. Tak zdefiniowany element będzie widoczny w przeglądarce, natomiast ukryty dla czytników ekranowych.

```
<p role="button" aria-hidden="true">Accessibly hidden button</p>
```

Kolejny przykład obrazuje użycie własności **aria-labelledby** i **aria-describedby** określających relacje między etykietą elementu a związaną z nim treścią, np. w przypadku „tabów" i ich zawartości.

```
<ul role="tablist">
   <li role="tab" id="r-0-tab" aria-describedby="r-0">Recipe 1</li>
   <li role="tab" id="r-1-tab" aria-describedby="r-1">Recipe 2</li>
   <li role="tab" id="r-2-tab" aria-describedby="r-2">Recipe 3</li>
</ul>

<div role="tabpanel" id="r-0" aria-labelledby="r-0-tab">
     Recipe 1 content
</div>
<div role="tabpanel" id="r-1" aria-labelledby="r-1-tab">
     Recipe 2 content
</div>
<div role="tabpanel" id="r-2" aria-labelledby="r-2-tab">
     Recipe 2 content
</div>
```

Jeden element może mieć kilka atrybutów **aria-**, które mogą być zmieniane przy użyciu JavaScriptu. Specyfikacja definiuje aż 35 ich rodzajów. Kilka przykładowych znajduje się w tabeli poniżej.

Stan/własność	**Opis**
aria-activedescendant (property)	Oznacza aktywny element będący potomkiem widgetu złożonego (**tablist**, **list**).
aria-checked (state)	Wskazuje na zaznaczenie danego elementu typu **radio** lub **checkbox**.
aria-expanded (state)	Wskazuje, że element rozwijalny (np. drzewo) jest obecnie rozwinięty.
aria-grabbed (state)	Oznacza, że element, który można przeciągnąć i upuścić jest obecnie przeciągany.
aria-invalid (state)	Wprowadzona wartość jest nieprawidłowa.
aria-required (property)	Element jest wymagany w formularzu.

Stan/własność	Opis
aria-sort (property)	Informuje czy zawartość tabeli jest posortowana rosnąco czy malejąco.
aria-valuemax (property)	Element posiada maksymalną wartość równą wartości atrybutu.
aria-valuemin (property)	Analogiczny do **aria-valuemax**, określa minimalną możliwą wartość.

Tab. 2: Atrybuty WAI-ARIA – przykłady

Ze wsparciem dla stanów i własności jest podobny problem jak z rolami. Poszczególne programy oraz ich wersje znacznie różnią się pod względem stopnia wsparcia dla WAI-ARIA, a testowanie jest utrudnione przez to, że efekt wymaga implementacji po obu stronach – zarówno w przeglądarce jak i w czytniku.

Jeśli AT obsługuje te funkcjonalności, stany i wartości mogą dostarczyć użytkownikom korzystającym z technologii wspomagających informacji o tym, jakie zmiany zachodzą na stronie. Jedna z nich – aria-live – jest szczególnie użyteczna w przypadku dynamicznych zmian w obrębie jednej strony, dokonywanych np. przy użyciu AJAX-a lub manipulacji DOM-em za pomocą JavaScriptu.

4.3. Dynamiczne regiony na stronie WWW obsługiwane przez atrybut **aria-live**

Po wejściu na stronę internetową, czytnik ekranowy tworzy w pamięci zrzut jej zawartości, z której korzysta później przy odczytywaniu treści. Użycie wewnętrznego buforu powoduje, iż programy te nie potrafią samodzielnie wykryć, że część zawartości została zmieniona przez asynchroniczną komunikację z serwerem lub manipulacje JavaScriptem.

Jednym z popularnych obejść tego problemu było do tej pory używanie ukrytego elementu <input> i podmiana jego zawartości przy każdej zmianie. Ponieważ czytnik jest zaprogramowany na reagowanie na zmianę zawartości pól formularza, akcja ta zwykle skutkuje wyczyszczeniem pamięci i stworzeniem nowego obrazu strony. Zmiana wartości elementu i ustawienie zaznaczenia na nowy element powoduje, że użytkownik informowany jest o zaktualizowaniu treści. Nie zawsze jest to jednak rozwiązanie skuteczne, o czym można przeczytać w artykule Jamesa Edwardsa opisującym różne przypadki testowe.[24]

Użycie aria-live znosi konieczność stosowania tego kłopotliwego tricku. Element, który posiada ten atrybut, jest traktowany jako region dynamiczny, a wszelkie zmiany w jego obrębie są sygnalizowane użytkownikowi.

Atrybut ten może przyjmować jedną z trzech wartości określających „poziom grzeczności”. Wyznacza on priorytet, jaki mają komunikaty informujące o zmianach w regionie aria-live:[25]

Wartość atrybutu **aria-live**	Opis
aria-live="off"	Wartość domyślna, oznacza, że czytnik będzie w stanie odczytać aktualną wartość, ale użytkownik nie będzie informowany komunikatem o zmianach w tym regionie. Zalecany do używania w elementach, których wartość zmienia się bardzo często.
aria-live="polite"	Zmiany dokonywane w tym regionie będą zakomunikowane użytkownikowi, tylko jeśli w danym momencie nie robi nic

Wartość atrybutu `aria-live`	Opis
	innego. Jeśli użytkownik korzysta ze strony (np. odczytuje ją lub wypełnia formularz) informacja pojawi się, kiedy skończy daną czynność. Zalecany dla większości zastosowań aria-live.
`aria-live="assertive"`	Oznacza, że zmiany są istotne i użytkownik powinien zostać o nich poinformowany tak szybko, jak to możliwe (nie oznacza to, że ma się to stać natychmiast, niezależnie od tego, co robi). Zalecany dla komunikatów o dużym priorytecie, np. dla błędów wypełnienia formularza.

Tab. 3: Możliwe wartości atrybutu aria-live

Dokładny sposób informowania użytkownika o zmianach zależy od technologii wspomagającej, której używa, i oczywiście od tego, czy jest zaimplementowany w przeglądarce i czytniku.

Jeżeli zmiany są bardziej złożone (np. jest to seria aktualizacji przebiegająca w krótkim czasie), można tymczasowo wstrzymać odświeżanie zawartości obrazu strony w pamięci czytnika. Używa się do tego celu atrybutu **`aria-busy`**. Ustawienie jego wartości na true powoduje, że czytnik śledzi zmiany, ale nie informuje o nich użytkownika, dopóki nie zostanie on przestawiony na false. Dopiero wtedy odczytane zostaną wszystkie zmiany.

Atrybut **`aria-atomic`** informuje, czy zmiana wymaga odczytania tylko fragmentu, który został zaktualizowany, czy należy odczytać cały region. Ustawienie domyślne false oznacza, że zmiany będą odczytywane niezależnie. Przestawienie wartości atrybutu na true spowoduje, że za każdym razem odczytywany będzie cały region.

Trzeci z atrybutów – **`aria-relevant`** informuje o tym, czy dana zmiana jest istotna i powinna zostać odczytana użytkownikowi. Może on przyjmować cztery wartości, przy czym możliwe jest, że będzie to więcej niż jedna z nich jednocześnie. Domyślne ustawienie to **`aria-relevant="additions text"`**

Wartość `aria-relevant`	Opis
`aria-relevant="additions"`	Dodanie nowych elementów HTML będzie zakomunikowane użytkownikowi.
`aria-relevant="removals"`	Usunięcie elementów HTML będzie komunikowane użytkownikowi. Zwykle jest to zbędne, dlatego w domyślnym ustawieniu jest wyłączone.
`aria-relevant="text"`	Zmiana tekstowej zawartości istniejących elementów HTML będzie komunikowana użytkownikowi.
`aria-relevant="all"`	Równoznaczne jest z: `aria-relevant="additions removals text"`

Tab. 4: Możliwe wartości atrybutu aria-relevant

Regiony **`aria-live`** to bardzo wygodny i elegancki sposób na rozwiązanie problemu podmiany części zawartości strony bez konieczności stosowania obejść wykorzystujących luki w technologiach. Niestety jak w przypadku całej technologii ograniczeniem jest wsparcie ze strony producentów oprogramowania.

Obecnie `aria-live` jest w pełni wspierana przez JAWS (od wersji 10, aktualna to 12) i VoiceOver w systemie Lion (Mac OS X 10.7).

4.4. WAI-ARIA – użycie, zalety i problemy

WAI-ARIA została stworzona z myślą o niewidomych użytkownikach. Użycie dodatkowych atrybutów pozwala przypisać funkcje elementów interfejsu dowolnym tagom na stronie.

Adresuje ona wiele obszarów sprawiających kłopoty twórcom dynamicznych aplikacji internetowych. Zastosowanie regionów `aria-live` pozwala uporać się z problemem dynamicznych zmian w obrębie pojedynczej strony. Inne elementy np. `role="button"` pozwalają w prosty sposób stworzyć dostępne elementy interaktywne.

Podstawowy problem z WAI-ARIA polega na tym, że wciąż nie można liczyć na jej pełne wsparcie zarówno po stronie przeglądarek, jak i czytników ekranowych. Z czytników właściwie jedynie JAWS ma zaimplementowaną obsługę większości ról, stanów i własności, ale jest drogi i dostępny jest jedynie dla użytkowników systemu Windows. VoiceOver w obecnej wersji ma dobre wsparcie dla tej technologii, ale została ona wydana bardzo niedawno, bo w lipcu 2011 razem z systemem OS X Lion. VoiceOver dla wersji Mac OS X Snow Leopard w ogóle nie wspierał WAI-ARIA.

W przeglądarkach sytuacja też jest różna – IE w wersji 9 ma pełne wsparcie dla WAI-ARIA, ale mogą z niego korzystać jedynie użytkownicy Windowsa Visty i 7 (nie można go zainstalować pod wciąż popularnym Windowsem XP). Firefox jest dostępny na wszystkich platformach i ma bardzo dobre wsparcie dla WAI-ARIA, niestety nie można z niego korzystać na komputerach typu Mac, ponieważ nie jest kompatybilny z API dostępności Apple.

Kolejny problem to ilość dodatkowych atrybutów, które muszą pojawić się w tradycyjnym kodzie HTML i konieczność obsłużenia ich przez JavaScript. WAI-ARIA dokładnie określa, jakie atrybuty muszą zostać dodane i jak należy nimi manipulować, aby osiągnąć zamierzony efekt. Zmiany stanów widgetu powinny zatem znaleźć odzwierciedlenie w wartościach atrybutów WAI-ARIA, a osiągnąć to można przez aktualizowanie ich za pomocą JavaScriptu.

W niektórych przypadkach działanie WAI-ARIA może powodować problemy z działaniem dotychczasowo stosowanych rozwiązań. Takim przykładem są regiony `aria-live` i starsze metody obchodzenia problemu aktualizacji części strony przez zmianę stanu ukrytego pola formularza. Dodanie regionu `live` do istniejącej logiki może mieć nieprzewidziane wyniki w programach wspierających WAI-ARIA.

Tym sposobem technologia ta nie zmniejsza w żaden sposób ilości pracy, jaką programista musi włożyć w stworzenie interaktywnych widgetów. Role i atrybuty stanowią tylko informację semantyczną dla czytnika, a przeglądarki nie wiążą z nimi żadnych domyślnych akcji interfejsu. Oznacza to, że WAI-ARIA nie niesie ze sobą żadnych korzyści, jeśli chodzi o czas i koszty w projekcie. Wprost przeciwnie – dłuższy czas pracy programistów to droższy projekt.

Sporo elementów WAI-ARIA duplikuje się z elementami HTML (np. `role="form"` i element `<form>` lub `role="navigation"` i element HTML5 `<nav>`). Jest to wprawdzie podyktowane troską o zapewnienie kompatybilności z każdym możliwym kodem – `<div role="form">` jest traktowany przez technologie wspomagającą tak, jak gdyby był to po prostu `<form>` – ale powoduje, że stosowanie WAI-ARIA staje się bardziej uciążliwe i skomplikowane.

Warto też dodać, że według badań organizacji WebAIM z grudnia 2010 roku 30,9% użytkowników nie wie o istnieniu *landmarków* WAI-ARIA, a kolejne 25,9% nigdy ich nie używa.[26]

Czy zatem warto korzystać z tej technologii, skoro wiążą się z nią dodatkowy czas i koszty, a końcowy efekt dla użytkowników jest niepewny ze względu na różny stopień wsparcia w oprogramowaniu?

Odpowiedź jednak brzmi – tak, należy to robić. Przede wszystkim większość WAI-ARIA nie ma żadnego wpływu na dotychczasowe działanie aplikacji, więc można dodawać ją transparentnie. Poza tym wsparcie dla niej w programach stale rośnie, więc jest to działanie, które będzie miało korzyści w przyszłości. Niedogodności płynące z uciążliwego procesu wdrażania elementów WAI-ARIA można w dużym stopniu wyeliminować, korzystając z któregoś z popularnych JavaScriptowych frameworków, jak Dojo czy YUI. Ich twórcy sukcesywnie dodają do nich atrybuty ARIA, wystarczy więc ściągnąć nową wersję biblioteki, aby aplikacja zyskała dodatkowy walor dostępności.

5. Testowanie dostępności

Dostępność nie jest wielkością mierzalną w kategoriach logicznych prawda/fałsz. Różnorodność możliwych ograniczeń percepcji oraz liczba urządzeń i programów wspomagających stosowanych przez osoby niepełnosprawne powodują, że nie jest możliwe stworzenie testu, który byłby w stanie jednoznacznie określić, iż dana aplikacja jest dostępna zawsze i dla wszystkich.

Z tego samego powodu czynnik ludzki jest niezbędny w testach dostępności. Nie istnieje narzędzie, które pozwoliłoby automatycznie sprawdzić jej poziom. Wynika to głównie z jakościowego, a nie ilościowego lub logicznego charakteru kryteriów testowych. Oczywiście istnieją programy rodzaju Web Accessibility Toolbar[27], które mogą wspomóc ten proces, np. sprawdzając obecność wymaganych elementów typu `<label>` lub atrybutów takich jak `alt`. Żaden algorytm jednak nie jest w stanie określić, czy alt rzeczywiście zawiera adekwatny opis zawartości grafiki, czy jest jedynie przypadkowym tekstem. Podobnie możliwe jest automatyczne sprawdzenie kolorystycznego kontrastu danej strony za pomocą programu Contrast Analyser[28], ale już nie ocena ogólnej estetyki testowanego rozwiązania. Algorytmy nie radzą sobie także ze sprawdzaniem funkcjonalności spersonalizowanych widgetów i dynamiczną treścią na stronie.

Narzędzia testujące, takie jak WAVE[29] (Wave Accessibility Evaluation Tool) przygotowany przez organizację WebAIM, mogą dostarczyć wielu pomocnych informacji. Służą one jednak przede wszystkim do wyeliminowania najbardziej podstawowych błędów i powinny jedynie stanowić wstęp do głównych procedur testowych wykonywanych przy udziale niepełnosprawnych użytkowników. Innymi słowy – aplikacja, która nie przechodzi etapu testów automatycznych, na pewno nie spełnia kryteriów dostępności WAI, ale brak błędów jeszcze nie oznacza, że jest ona dostępna.

Jak zatem testować aplikacje i serwisy pod kątem dostępności? W pierwszym etapie jak najbardziej można wykorzystać testy automatyczne, które pozwolą poprawić błędy w kodzie, uzupełnić brakujące elementy i uporządkować treść. Główna część testowa powinna z kolei polegać na wykonywaniu scenariuszy przypadków użycia przez możliwie zróżnicowaną grupę osób niepełnosprawnych używających różnego rodzaju technologii wspomagających. Zebrane w ten sposób informacje pozwolą odnaleźć blokady i niedostępne obszary aplikacji.

Na zakończenie tego rozdziału warto wspomnieć, że w ramach działań podjętych przez Państwowy Fundusz Rehabilitacji Osób Niepełnosprawnych, znajdziemy i taką inicjatywę, w której wykonano testowanie około 200 stron internetowych – głównie instytucji publicznych – pod kątem ich dostępności dla osób z różnym rodzajami i stopniem niepełnosprawności, w wyniku czego powstało 200 raportów z zaleceniami zmian w serwisach WWW instytucji państwowych, urzędów, ministerstw czy organizacji pozarządowych[30]. Niestety, pod podanym adresem WWW nie znajdziemy opracowanych przy testowaniu rekomendacji zmian.

6. Przykłady rozwiązań zwiększających dostępność serwisów WWW tworzonych z użyciem języków HTML i JavaScript

Aplikacje tworzone przy użyciu języka JavaScript używają języka HTML do budowania i wyświetlania interfejsu użytkownika. Oznacza to, że występują w nich wszystkie problemy z dostępnością, które można znaleźć w tradycyjnych, statycznych witrynach internetowych.

Dodatkowo jednak mamy w nich do czynienia z nową kategorią problemów związanych z dynamicznym generowaniem zawartości strony i dużo większymi możliwościami interakcji z użytkownikiem.

Zgodnie z tym, co zostało opisane wyżej, nie można jednoznacznie stwierdzić, czy dany serwis jest dostępny, szczególnie bez przeprowadzenia testów z udziałem grupy osób niepełnosprawnych. Istnieje jednak szereg strategii, które pozwalają znacznie zwiększyć poziom dostępności takiej aplikacji w trakcie pisania kodu.

Strategie te sprowadzają się do zapewnienia, że:

1. Strony są prawidłowo sformatowane i zawierają wszystkie informacje umożliwiające ich odczytanie przez technologie wspomagające.
2. Elementy graficzne i multimedialne mają odpowiedniki, które mogą być odczytane przez osoby niewidome, słabo widzące lub niesłyszące.
3. Wszystkie interakcje są możliwe do wykonania za pomocą klawiatury.
4. Dynamiczne zmiany w obrębie pojedynczych stron są dostępne dla użytkowników niewidomych.

Rozwinięciem tej części pracy jest serwis stworzony w ramach niniejszego opracowania, który prezentuje w praktyce rozwiązania techniczne omówione w tym rozdziale. Zawiera on dostępne wersje zarówno najbardziej podstawowych elementów stron internetowych jak teksty czy formularze, jak i zaawansowanych, spersonalizowanych kontrolek w rodzaju kalendarza, czy rozwijanej listy wyboru, które rozszerzają zestaw narzędzi dostępnych domyślnie w przeglądarce.

Projekt ten objaśnia również, w jaki sposób zapewnić dostępność tak popularnych rozwiązań, jak interakcje wywoływane za pomocą zdarzenia **`onclick`**, komunikacja z wykorzystaniem AJAX-a, zmieniająca zawartość jedynie części strony, czy znany z systemów operacyjnych sposób przenoszenia elementów typu *drag & drop*.

Przykłady zaprezentowane w serwisie wykorzystują WAI-ARIA, nie polegają jednak wyłącznie na nich ze względu na wciąż słabe wsparcie ze strony czytników i przeglądarek. Tam, gdzie to możliwe, zaimplementowane zostały mechanizmy, które pozwalają na zapewnienie dostępności dla osób niepełnosprawnych tradycyjnymi środkami.

Teksty w projekcie napisane są w języku angielskim, aby umożliwić jego testowanie na możliwie największej ilości platform i urządzeń. Niestety, z powodu wciąż niewielkiej popularności komputerów firmy Apple na naszym rynku, wersja czytnika VoiceOver dla systemu Mac OS X Snow Leopard, dostępna wśród narzędzi, nie obsługuje języka polskiego. Wybór języka angielskiego był więc rozwiązaniem optymalnym z punktu widzenia procedur testowych.

Dalsza część tego rozdziału będzie opisywać dostępne metody pozwalające na spełnienie tych założeń. W przypadku obszarów związanych z użyciem WAI-ARIA, przedstawione zostaną możliwe sposoby na uzyskanie podobnych efektów bez ich użycia (ze względu na problemy ze wsparciem).

6.1. Struktura dokumentu

6.1.1. Język

Każdy dokument HTML powinien zawierać informację o języku, którego używa dana strona. Służy do tego atrybut lang, który może być zastosowany na całym dokumencie lub na dowolnym elemencie, jeśli teksty napisane są w kilku językach:

```
<html lang="pl">
```

```
<p lang="en">This paragraph is in English</p>
```

Informacja ta jest potrzebna czytnikom ekranowym, które w zależności od jego wartości dostosowują sposób odczytywania treści. Jego brak może spowodować, że odczytana zawartość będzie kompletnie niezrozumiała dla użytkownika.

6.1.2. Struktura treści i nagłówki

Ukłonem w stronę osób używających tego typu oprogramowania jest takie ułożenie treści na stronie, aby istotna treść znajdowała się jak najbliżej początku dokumentu, a elementy nawigacyjne w jego dalszej treści. Taka kolejność ma znaczenie, ponieważ czytniki przetwarzają cały dokument od początku. Umieszczając menu na początku, zmuszamy użytkownika do odsłuchania go przy każdym przejściu na nową stronę. Alternatywą jest dodanie linku-kotwicy, który „przeskakuje" nawigację i prowadzi do głównej treści (tzw. *skip links*).

Odpowiednia struktura dokumentu ma bardzo duże znaczenie ze względu na wspomnianą wcześniej funkcjonalność czytników ekranowych, polegającą na prezentowaniu zestawień najważniejszych elementów na stronie – nagłówków, obrazków, list czy odnośników. Przy kodowaniu strony należy pamiętać o tym, że wizualne wyróżnienie elementu powinno towarzyszyć odpowiedniemu elementowi HTML. Zdecydowanie należy unikać sytuacji, kiedy opisany w arkuszach CSS `<span>` udaje nagłówek.

Trzeba także pamiętać o stopniowaniu nagłówków od `<h1>` do `<h6>`. W numeracji nie powinno być przerw, a nagłówki najwyższego poziomu `<h1>` powinny zawierać tytuł serwisu i/lub danej strony. Wszystkie pozostałe treści powinny operować na nagłówkach `<h2>` do `<h6>`.

Elementy opisujące strukturę strony powinny mieć dodane role WAI-ARIA typu *landmark*. Dzięki temu użytkownicy czytników, które wspierają tę funkcjonalność, będą mogli w łatwiejszy sposób przemieszczać się po stronie.

```
<header>
    <div role="banner">
        <h1>Główny tytuł</h1>
    </div>
</header>
```

6.2. Obrazki

Grafikę w serwisach i aplikacjach internetowych można podzielić na dwie kategorie. Pierwsza to elementy dekoracyjne. Zaliczają się do niej zwykle wszystkie obrazki, które nie mają znaczenia dla treści, a jedynie mają uatrakcyjnić stronę wizualnie. Druga to grafiki istotne dla zawartości witryny, np. zdjęcia ilustrujące artykuł, logo firmy, do której należy strona, wykresy itp.

6.2.1. Użycie stylów (CSS)

Czytnik ekranowy odczytuje każdy element znajdujący się w kodzie HTML. Dlatego grafiki dekoracyjne nie powinny pojawiać się w strukturze dokumentu w postaci znacznika `<img>`. Odpowiednim miejscem do ich zdefiniowania są pliki CSS zawierające informacje o wizualnej części serwisu. W ten sposób użytkownik, który używa przeglądarki tekstowej, czytnika ekranowego lub ustawień wyświetlania dostosowanych do swoich potrzeb, nie musi przetwarzać elementów zbędnych.

6.2.2. Tekst alternatywny – atrybut `alt`

Grafiki, które mają istotne znaczenie dla treści serwisu, także mogą nie zostać wyświetlone przez oprogramowanie użytkownika lub użytkownik może ich nie widzieć. Dlatego koniecznie muszą zostać opisane przez atrybut **`alt`**.

Oprogramowanie wspomagające użyje go jako treści alternatywnej dla obrazka, dlatego powinien on jak najlepiej opisywać grafikę, którą zastępuje. Jeśli z jakichś powodów obrazek musi istnieć w kodzie jako element **`<img>`**, a nie powinien być przeczytany przez technologie wspomagające, można zastosować pusty atrybut **`alt`**, jego brak spowoduje bowiem odczytanie nazwy pliku.

6.3. Linki

6.3.1. Wizualne wyróżnienie

Wszystkie odnośniki powinny odróżniać się od „zwykłego” tekstu w sposób, który nie pozostawia wątpliwości co do ich interaktywności. Tradycyjnie jest to inny kolor i podkreślenie, dobrym rozwiązaniem jest jednak każdy rodzaj wyróżnienia, jeśli tylko pozwala na łatwe wizualne rozpoznanie linku.

Należy przy tym pamiętać, że wyróżnienie jedynie za pomocą koloru może nie wystarczyć. Rozmaite zaburzenia widzenia kolorów powodują problemy z rozróżnieniem różnych kolorów. Nie można polegać zatem jedynie na kontraście. Osoby z protanopią widzą czerwony jako odcień brązowego, dlatego będzie im trudno odnaleźć taki link wśród czarnego tekstu, jeśli nie zostanie on zaznaczony w jakiś dodatkowy sposób, np. przez podkreślenie. Podobny problem dotyczy cierpiących na tritanopię – niebieski link w czarnym tekście jest dla nich praktycznie niewidoczny.

6.3.2. Atrybut `title`

Zawartość tekstowa linku powinna pozwalać na jednoznaczne zidentyfikowanie zawartości, do której prowadzi. Dlatego należy zadbać o odpowiednią treść każdego odnośnika. W przypadku linków będących np. tytułami artykułów stosowanie atrybutu **`title`** nie jest konieczne – sam tekst niesie ze sobą wystarczającą ilość informacji.

Zdarzają się jednak odnośniki, których treść jest dość lakoniczna – typu „więcej”, „czytaj dalej” lub linki będące grafikami. W takim przypadku atrybut **`title`** pozwoli uzupełnić brakujące informacje.

6.4. Formularze

Formularze są drugim po odnośnikach najczęściej występującym elementem interaktywnym. Często używane są do przesyłania bardzo istotnych danych, nierzadko są skomplikowane i składają się nawet z kilkudziesięciu pól. Z tego względu wymagają znacznego zaangażowania użytkownika, i muszą być zaprojektowane tak, aby minimalizować możliwość pomyłki.

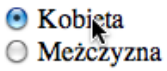
6.4.1. Dostępność z klawiatury

Najważniejszym testem na dostępność formularza jest próba wypełnienia go używając jedynie klawiatury. Jeśli formularz wymaga użycia myszki, oznacza to, że został źle wykonany. Przechodzenie między polami odbywa się przez naciśnięcie klawisza **Tab**. Czytniki ekranowe mogą mieć swoje kombinacje klawiszy służące do przechodzenia między elementami formularza.

Szczególną uwagę należy zwrócić na pola, pod które podpięte są dodatkowe kontrolki. Popularnym rozwiązaniem jest np. dodanie kalendarza do pól zawierających daty. Użytkownik zamiast

wpisywać datę w określonym formacie wybiera ją w dodatkowym boksie, który pojawia się po kliknięciu pola. Taki widget, bardzo wygodny dla osoby używającej myszki może stanowić problem dla kogoś posługującego się tylko klawiaturą.

Rozwiązaniem, którego zdecydowanie należy unikać, jest automatyczne przechodzenie na inną stronę po wybraniu elementu na liście rozwijanej. Odbywa się to przez przypisanie odpowiedniej logiki do zdarzenia **onchange**. Taka interakcja nie sprawia problemów przy użyciu myszki, ale przeglądanie zawartości listy przy użyciu czytnika ekranowego powoduje wywołanie **onchange** przy każdym przejściu między jej elementami, czyniąc nawigację praktycznie niemożliwą. Lepszym podejściem jest pozwolić użytkownikowi przejrzeć listę i wybrać którąś z opcji, a potem zatwierdzić wybór dodatkowym przyciskiem.



Ryc. 3: Etykieta pozwala na wygodniejsze zaznaczanie pól

6.4.2. Etykiety pól

Podstawowym elementem, który zapewnia dostępność formularza, jest etykieta czyli **<label>**. Za pomocą atrybutu **for** pozwala ona powiązać pole formularza z jego opisem:

```
<label for="email-address">Adres email</label>
<input type="text" id="email-address" name="email" />
```

W przypadku pól typu **radio** i **checkbox** element **<label>** ma jeszcze jedno zastosowanie – bez niego zaznaczenie pola jest możliwe tylko przez bezpośrednie kliknięcie w niewielką kontrolkę, co może być bardzo trudnym zadaniem dla niewprawnego użytkownika lub dla osoby o ograniczonych zdolnościach manualnych. Dodanie etykiety powoduje, że pole zaznaczane jest także przez kliknięcie w tekst

```
<input id="f" type="radio" name="plec"/> <label
for="f">Kobieta</label>
<input id="m" type="radio" name="plec"/> <label
for="m">Mężczyzna</label>
```

6.4.3. Oznaczanie pól wymaganych

Do dobrych praktyk należy odróżnianie pól, których podanie jest wymagane, od tych, które są opcjonalne. Obowiązuje tutaj taka sama zasada, jak w przypadku odnośników – sam kolor to za mało, potrzebne jest dodatkowe oznaczenie graficzne. Przyjętą konwencją jest używanie do tego celu symbolu „gwiazdki” - *****

Dla użytkowników czytników można użyć dodatkowego ukrytego elementu informującego, że pole jest wymagane (patrz rozdział 6.6.1.)

```
<label for="email-address">
     Adres email:
     <span class="offscreen">Pole jest wymagane</span>
</label>
<input type="email" name="email-address" id="email-address"/>
```

6.5. Elementy interaktywne

6.5.1. Wybór elementu HTML

Często spotykanym problemem jest nadużywanie tagu **<a>** do wszystkich elementów interaktywnych. Ze względu na wizualną odmienność odnośników i kursor „rączki” używa się ich do wyświetlenia większości elementów, którymi może manipulować użytkownik. Ponieważ z kolei

nie istnieje zasób, do którego taki link miałby prowadzić, atrybut **href** zostaje ustawiony na **"#"** a domyślna akcja (przeskok na górę strony) anulowana jest za pomocą JavaScriptu.

Jest to przykład złej praktyki i mieszania warstwy prezentacji i funkcji elementu. Do takich celów należy wykorzystywać raczej tag **<button>** odpowiednio opisany w arkuszach stylów.

Można również wybrać każdy inny element np. ****, stworzenie prawdziwego, dostępnego przycisku wymaga jednak dużej ilości pracy, bo trzeba symulować domyślne zachowania przeglądarki, czyli dodać możliwość zaznaczania z klawiatury, reagowanie na kliknięcia i standardowe akcje klawiszy „enter" i „space". Oprócz tego, aby był prawidłowo rozpoznawany przez czytniki, wymaga roli **button**.

6.5.2. Dostęp za pomocą klawiatury – **tabindex**

Domyślnie jedyne elementy, które można zaznaczyć za pomocą klawiatury, to pola formularza **(<input>**, **<select>** i **<textarea>**) oraz linki.

Dzięki użyciu atrybutu **tabindex** każdy element może zostać zaznaczony za pomocą klawiatury. Ustawienie go na wartość 0 (zero) powoduje, że element staje się osiągalny przy użyciu klawisza tab.

```
<span tabindex="0">Element stał się zaznaczalny tabem!</span>
```

Samo ustawienie możliwości zaznaczenia nie jest wystarczające, aby czytnik prawidłowo zinterpretował interaktywną funkcję elementu. Pomóc może w tym WAI-ARIA i rola **button**:

```
<span role="button" tabindex="0">
     Element stał się zaznaczalny tabem i jest rozpoznawany jako
przycisk.
</span>
```

Jeżeli element ma być dostępny też dla przeglądarek/czytników, które nie wspierają WAI-ARIA, wewnątrz elementu należy dodać tekst informujący o możliwości wykonania akcji, który będzie niewidoczny dla użytkowników, a dostępny dla czytników (patrz rozdział 6.6.1.)

```
<span role="button" tabindex="0">
     Element stał się zaznaczalny tabem i jest rozpoznawany jako
przycisk.
     <span class="offscreen">Wciśnij enter, żeby użyć</span>
</span>
```

6.5.3. Obsługa akcji za pomocą klawiatury

Zgodnie z wytycznymi WCAG 2.0. każdy element powinien być dostępny za pomocą klawiatury. Oznacza to, że wszelkie akcje użytkownika, które zwykle wykonywane są przy użyciu myszki, powinno dać się także wykonać przez zaznaczenie elementu klawiszem tab i naciśnięciu enter.

Przypisanie akcji do zdarzenia **onclick** na elemencie, który jest domyślnie zaznaczalny często wystarczy, aby ta sama logika została wywołana przy użyciu entera. Nie jest to jednak reguła, dlatego bezpieczniej jest napisać obsługę tego typu zdarzeń, która uwzględnia oba przypadki.

Jeżeli **onclick** przypisany jest do elementu, który nie jest aktywny, własna obsługa naciśnięcia klawisza enter jest wymagana, aby akcja została wywołana (oczywiście trzeba też pamiętać o argumencie **tabindex**, który umożliwia zaznaczenie elementu z klawiatury, oraz dodaniu odpowiedniej roli WAI-ARIA).

W takich przypadkach zawsze warto rozważyć zastąpienie problematycznego elementu którymś ze standardowych interaktywnych tagów. Najbardziej uniwersalny jest **<button>**, ale ostateczna

decyzja powinna zależeć od tego, czy klikany element jest prawidłowym odnośnikiem do jakiegoś zasobu, kontrolką formularza, czy jakąś funkcjonalnością działającą jedynie w obrębie tej jednej strony. Style CSS pozwalają na nadanie dowolnego wyglądu takiemu przyciskowi, a można dzięki temu uniknąć konieczności tworzenia własnej logiki obsługi zdarzenia.

6.6. Ukrywanie elementów na stronie

Ukrywanie elementu na stronie tak, aby był dostępny dla czytników ekranowych, bywa przydatne, kiedy trzeba przekazać niewidomym użytkownikom dodatkowe informacje, a jednocześnie nie powinny się one pojawiać na ekranie.

Istnieje kilka metod na ukrycie zawartości za pomocą stylów CSS, mają one jednak różny wpływ na zachowanie strony w czytniku:[31]

CSS	Wyświetlanie	Czytniki
`visibility: hidden;`	Element nie jest widoczny, ale wciąż zajmuje tyle samo miejsca na ekranie.	Zawartość jest ignorowana.
`display: none;`	Element nie jest widoczny, ani nie zajmuje miejsca na ekranie.	Zawartość jest ignorowana.

`height: 0;`
`width: 0;`
`overflow: hidden;`
	Wymiary elementu zostały zmniejszone do zero – zawartość jest niewidoczna.	Zawartość jest ignorowana.

`text-indent: -999em;` `overflow: hidden;`	Zawartość elementu jest przesunięta poza ekran. Element zajmuje miejsce na ekranie, mogą pojawić się problemy z tłami.	Czytniki mają dostęp do zawartości.
`position: absolute;` `left: -999em;`	Element jest usunięty z domyślnego ułożenia i przesunięty poza ekran. Nie zajmuje miejsca na ekranie.	Czytniki mają dostęp do zawartości.

Tab. 5: Ukrywanie elementów na stronie - różnice w stopniu dostępności poszczególnych metod

Ostatni sposób, polegający na pozycjonowaniu elementu poza ekranem, jest zwykle najlepszym rozwiązaniem dla dostępnego ukrywania, bo nie wpływa na wygląd strony.

```
.offscreen {
  position: absolute;
  left: -999em;
}
```

```
<label for="email-address">
     Adres email:
     <span class="offscreen">Pole jest wymagane</span>
</label>
```

6.7. Dynamicznie generowane treści

Jest to chyba najpopularniejsze zastosowanie JavaScriptu, które jednocześnie jest najtrudniejsze do przeniesienia w świat użytkowników niewidomych. Dzieje się tak z powodu buforowania

zawartości stron przez czytnik w momencie ich załadowania. Wszystkie późniejsze zmiany dokonane przez JavaScript są dla niego niewidoczne.

Jest to bardzo poważne ograniczenie, które mocno limituje możliwości tworzenia dynamicznych aplikacji dostępnych dla osób niewidomych, dlatego WAI-ARIA zawiera elementy, które rozwiązują ten problem – są to regiony **`aria-live`**. Obszar na stronie oznaczony w ten sposób będzie odświeżany w pamięci czytnika, kiedy tylko jego zawartość się zmieni.

Niestety niepełna implementacja regionów **`aria-live`** w przeglądarkach i czytnikach w dalszym ciągu powoduje konieczność szukania innych rozwiązań. Najbardziej popularne i skuteczne z nich opiera się na wykorzystaniu faktu, że czytniki odświeżają zawartość bufora w momencie, kiedy zmienia się wartość któregoś z pól w formularzu.

Strategia postępowania w tym przypadku wygląda zatem następująco:

1. JavaScript wykonuje jakąś akcję, która zmienia zawartość strony, np. zapytanie typu AJAX lub wygenerowanie nowego elementu interfejsu i dodanie go do drzewa DOM.
2. Po zakończeniu modyfikowania strony JavaScript zmienia wartość atrybutu **`value`** pola formularza ukrytego w dostępny sposób (patrz punkt 6.6.1). Podmieniona wartość może być dowolna, ale powinna ona informować użytkownika o tym, co się stało.
3. Po odczytaniu użytkownikowi komunikatu o zmianach, wywołane zostaje zdarzenie **`focus`**, zaznaczające element na początku zmienionej sekcji i przenoszące tam kursor użytkownika.

Ta złożona procedura jest niestety tylko obejściem problemu i łatwo sobie wyobrazić, jak skomplikowane może być zaprojektowanie interakcji na stronie, na której znajduje się kilka takich dynamicznych regionów. Poza tym tego typu zachowanie przerywa użytkownikowi proces współpracy ze stroną.

WAI-ARIA daje dużo lepsze narzędzia, pozwalając ustawić poziom „grzeczności” komunikatów (jak bardzo powinny ingerować w to, co robi użytkownik), sposób komunikowania zmian (cały region, czy tylko jego zmienioną część), a nawet co powinno być komunikowane (dodanie elementu, usunięcie elementu, zmiana treści elementu). Szczegółowy opis działania regionów **`aria-live`** znajduje się w rozdziale 5.3.3.

Niestety jednoznaczna odpowiedź na pytanie, jak postępować w przypadku dynamicznie podmienianej treści nie jest możliwa. Jeżeli strona jest stosunkowo prosta i sztuczka z podmianą zawartości pola formularza nie prowadzi do dużego skomplikowania kodu, warto ją zastosować. W przypadku bardziej złożonych zmian, rozwiązanie to może okazać się jednak zbyt ingerujące w istniejącą logikę.

W wielu wypadkach zastosowanie regionów **`aria-live`** mogłoby rozwiązać ten problem. Poziom ich implementacji jest niestety zbyt niski (obecnie tylko JAWS je obsługuje), więc nie można liczyć na wsparcie z ich strony. W takiej sytuacji jednym z możliwych sposobów działania jest napisanie aplikacji z wykorzystaniem regionów **`aria-live`** i umieszczenie o tym ukrytej informacji na początku strony. W przypadku, kiedy oprogramowanie użytkownika obsługuje ten element WAI-ARIA, może on z niej skorzystać, jeśli nie, powinien mieć możliwość przełączenia się do prostszej wersji serwisu.

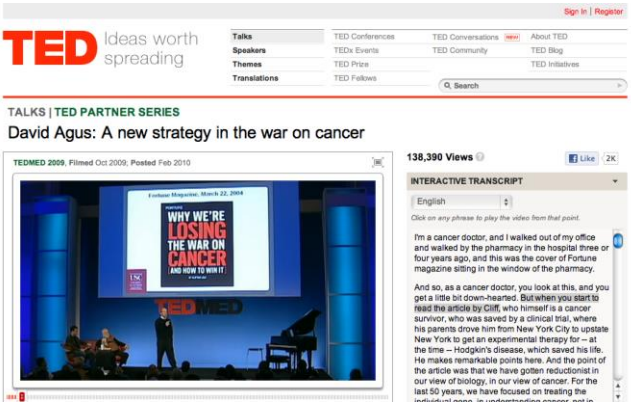
6.8. Multimedia

W przypadku multimediów w grę wchodzą dwa problemy z dostępnością – pierwszy to zawartość alternatywna dla osób niesłyszących, drugi dotyczy dostępu z klawiatury i współpracy z czytnikami ekranowymi.

6.8.1. Zawartość alternatywna

Treść alternatywna dla filmów wideo to najczęściej ścieżka napisów zsynchronizowana z obrazem. Najczęściej jest to transkrypcja dialogów, czasem wzbogacona o dodatkowe informacje dotyczące np. reakcji publiczności. Bardzo rzadko można także spotkać ścieżki w języku migowym.

Przy tworzeniu nagrań wideo przeznaczonych do publikacji w serwisie internetowym należy też zadbać o odpowiednie przygotowanie prezentacji z myślą o osobach niewidomych. Jeżeli tylko jest to możliwe, występujący powinni opowiadać o rzeczach, które pokazują, tak aby jak najwięcej informacji wizualnych miało swój dźwiękowy odpowiednik.



Ryc. 4: Przykład interaktywnych, zsynchronizowanych napisów na stronie organizacji TED. Kliknięcie w część transkrypcji przenosi do odpowiedniego miejsca w pliku wideo.

Pod względem technicznym napisy mogą być zakodowane w tym samym pliku co film lub też mogą być wczytywane i obsługiwane przez odtwarzacz multimediów z osobnego pliku tekstowego. Drugie rozwiązanie jest wygodniejsze, ponieważ daje większą swobodę w posługiwaniu się napisami – pozwala zwiększać czcionkę, czy tworzyć ścieżki w wielu językach. Możliwość zwiększenia rozmiaru czcionki jest istotna dla użytkowników gorzej widzących.

Dużą zaletą stosowania zsynchronizowanych napisów jest to, że przydaje się nie tylko osobom głuchym. Dzięki nim można wyeliminować problem niskiej jakości dźwięku, hałasy tła, stanowią one także dużą pomoc dla osób mniej biegłych w języku, w którym nagrano dany film. Tekstowe odpowiedniki plików wideo pozwalają także na lepsze indeksowanie ich treści przez wyszukiwarki. Stosunkowo proste jest napisanie odtwarzacza, który pozwoli obsłużyć napisy, można również kupić któryś z dostępnych na rynku programów, które są bardzo łatwe w obsłudze (np. firmy Nomensa[32])

Niestety stworzenie takich napisów jest dość czasochłonne i kosztowne. Wymaga ono bowiem nie tylko transkrypcji (która sama w sobie oznacza kilkakrotne obejrzenia filmu), ale także obróbki pliku i odpowiedniego zsynchronizowania go z obrazem. Jeśli w grę wchodzą także tłumaczenia na inne języki, cena i czas rosną lawinowo. W Stanach Zjednoczonych sama transkrypcja to koszt ok 1-10 USD za minutę materiału dźwiękowego, a czas wykonania zlecenia to ok 3-10 dni.[33] W Polsce cena za jedną stronę spisanego tekstu wynosi 3-12 PLN (w języku polskim). Jedna z firm mających w swojej ofercie taką usługę wycenia, że przy takich stawkach koszt spisania 30-minutowego nagrania wynosi 50-70 PLN.

Jeszcze trudniejsze jest zapewnienie treści alternatywnej dla transmisji na żywo. Jest to wprawdzie możliwe, ale wymaga takiego nakładu pracy i kosztów, że jest w zasadzie niespotykane.

Istnieją serwisy takie jak dotSUB[34], które upraszczają proces tworzenia transkrypcji dla plików wideo dzięki udostępnianiu interfejsu, który umożliwia synchronizowanie dźwięku z napisami (użytkownik zatrzymuje film w dowolnym momencie, wpisuje tekstowy odpowiednik i zatwierdza), jednak wciąż jest to proces czasochłonny.

W przypadku plików audio zawartością alternatywną może być transkrypcja w pliku tekstowym, dostępna np. jako osobna strona HTML albo plik do pobrania.

6.8.2. Dostępność z klawiatury

Jak wszystkie pozostałe elementy interfejsu, odtwarzacze plików multimedialnych także powinny być dostępne za pomocą klawiatury. Najłatwiej osiągnąć to kupując gotowy odtwarzacz (np. Nomensa) lub wykorzystując któryś z wielu darmowych projektów (np. JW Player[35] lub The Workshop[36]).

Jeżeli z jakichś względów jest to niemożliwe, również stworzenie takiej aplikacji nie jest szczególnie trudne. Elementy sterujące powinny być elementami HTML, które mogą być zaznaczone i aktywowane klawiaturą za pomocą tradycyjnej kombinacji klawiszy `Tab` i `Enter`, i które są dobrze opisane na potrzeby czytników ekranowych (np. za pomocą atrybutu `alt` lub ukrytego elementu HTML). Interakcja użytkownika powinna spowodować, że przypięty do nich JavaScript skomunikuje się z filmem czy klipem dźwiękowym i odpowiednio zmodyfikuje jego zachowanie.

Według WCAG 2.0 funkcjonalności, jakie musi udostępniać taki odtwarzacz, to standardowe przyciski do startowania, pauzowania i zatrzymywania odtwarzania oraz regulacja głośności. Powinien on też umożliwiać zmianę rozmiaru czcionki użytej w napisach.[37]

6.9. Nieinwazyjny JavaScript

Jednym z kryteriów dostępności stron internetowych jest ich niezależność od urządzenia używanego do przeglądania internetu oraz jakichkolwiek dodatkowych technologii w rodzaju Flasha czy JavaScriptu. Inaczej mówiąc – strona powinna działać tak w najprostszych przeglądarkach tekstowych, które nie posiadają żadnego z tego typu udogodnień, jak i w tradycyjnych przeglądarkach przy wyłączonym JavaScripcie.

Technika stosowana w tym celu to tzw. *progressive enhancement,* co oznacza stopniowe wzbogacanie funkcjonalności w miarę zwiększania się możliwości używanych przeglądarek. Oznacza to, że każda aplikacja napisana w JavaScripcie musi mieć swoją statyczną wersję.

Zgodnie z zasadą *progressive enhancement,* tworzenie serwisów należy zaczynać od stworzenia najprostszej działającej jego wersji przy użyciu tradycyjnych elementów HTML. I dopiero na tej podstawie powinna pojawić się bardziej złożona logika wprowadzana przez JavaScript.

Trudno jednoznacznie odpowiedzieć na pytanie, w jaki sposób odtworzyć JavaScriptowe funkcjonalności w HTML-u. Nie wszystkie z nich muszą być dostępne w wersji podstawowej. Udogodnienia, które nie są wymagane dla osiągnięcia danego celu (np. dokonania zakupu) mogą zostać wyłączone. Zapytania AJAX-owe można dość łatwo zastąpić tradycyjnymi zapytaniami do serwera, a podmienianie części strony – jej przeładowaniem. Taka statyczna wersja nie będzie oczywiście aktualizować się automatycznie, więc można poinformować użytkownika, że powinien odświeżać ją, aby mieć dostęp do aktualnych treści. Elementy interfejsu powinny mieć HTML-owe odpowiedniki. Może to być element formularza, w który użytkownik będzie mógł wpisać wymagane dane, sprawdzi się tu dobrze opisane zwykłe pole tekstowe, lista rozwijana, zestaw przycisków radio lub pole typu checkbox. Najważniejsze jest, aby użytkownik mógł dotrzeć do celu.

Należy także pamiętać, że nieinwazyjny JavaScript nie oznacza JavaScriptu dostępnego. Nierzadkie jest przekonanie, że jeśli wersja HTML jest dostępna, nie trzeba już martwić się o dostępność widgetu rozszerzającego tę funkcjonalność. Nic bardziej mylnego – według badań organizacji WebAIM z grudnia 2010 roku, 98,4% użytkowników czytników ekranowych ma włączony JavaScript[38] i nigdy nie zobaczy podstawowej wersji strony. Dlatego wszystkie elementy stworzone przy użyciu tego języka powinny być dostępne za pomocą klawiatury i dla czytników ekranowych.

7. Podsumowanie

Opracowanie niniejsze poświęcone jest ważnemu problemowi dostępności stron internetowych w ogólności, w tym szczególnie dostępności stron WWW dla osób niepełnosprawnych. W rozdziale pierwszym omówiono zasadnicze pojęcia ogólne, związane z problemem dostępności. W kolejnej części opracowania (rozdział drugi) omówiono rodzaje dysfunkcji u potencjalnych odbiorców stron WWW, w szczególności u osób mających problemy ze wzrokiem. W rozdziale trzecim, zawarto opis istniejących w tym zakresie zaleceń gremiów eksperckich, zwłaszcza z konsorcjum W3C. Rozdział czwarty poświęcony jest części omówieniu systemu WAI-ARIA, dedykowanego metodom tworzenia stron WWW z zawartością dynamiczną dla osób z upośledzonym sposobem widzenia. Osobny rozdział (rozdział piąty) poświęcono kwestiom testowania dostępności stron WWW. Wreszcie, praktycznym aspektom tworzenia dostępnych stron WWW z użyciem technologii HTML, CSS i JavaSript dedykowany jest ostatni, szósty rozdział tego opracowania.

Należy podkreślić, że na dobrej dostępności serwisów internetowych skorzystać mogą szerokie grupy odbiorców stron WWW. Oprócz osób niepełnosprawnych, chodzi m.in. o następujące grupy: starsi ludzie (jako szeroko pojęta i rosnąca w liczbę grupa odbiorców), użytkownicy urządzeń mobilnych, użytkownicy przeglądarek tekstowych, użytkownicy nietypowych systemów operacyjnych, nietypowych przeglądarek, z nietypowymi ustawieniami programów, osoby łączące się z Internetem za pośrednictwem nisko przepustowych łącz internetowych, osoby tymczasowo niesprawne – np. ze skręconym nadgarstkiem, osoby, które ze względu na okoliczności korzystają z serwisu w nietypowym dla siebie środowisku – np. z ograniczonych funkcjonalnie przeglądarek na lotniskach.

Problem dostępności stron internetowych dotyczy projektów nowo tworzonych, ale jeszcze mocniej aplikacji już działających. O ile bowiem w tym pierwszym przypadku, konsekwentne stosowane aktualnych zaleceń daje dużą szansę na powstanie dostępnego serwisu WWW, o tyle przekonstruowanie istniejących serwisów jest bardzo trudnym logistycznie i finansowo zagadnieniem, co nie znaczy, że beznadziejnym[39]. Można tylko wyrazić nadzieję, że w miarę upływu czasu procent serwisów WWW spełniających podstawowe kryteria dostępności będzie się wyraźnie powiększał. Jest to przekonanie o tyle uzasadnione, że wdrażane są nowe projekty, których celem jest właśnie poprawa dostępności serwisów internetowych dla osób niepełnosprawnych[40].

8. Bibliografia

[1] W3C, *Supported States and Properties*, http://www.w3.org/WAI/intro/accessibility.php

2 Biuro Pełnomocnika Rządu ds. Osób Niepełnosprawnych, *Dane dotyczące sytuacji osób niepełnosprawnych, zwłaszcza na rynku pracy, w 2010 roku*, http://www.niepelnosprawni.gov.pl/niepelnosprawnosc-w-liczbach/opracowania-analityczno-tabelary/

[3] "Dostępność serwisów internetowych – podręcznik na temat dobrych rozwiązań w projektowaniu dostępnych serwisów internetowych dostępnych dla osób z różnymi rodzajami niepełnosprawności", http://www.pfron.org.pl/ftp/publikacje/PODRECZNIK.pdf

4 American Foundation for the Blind, Refreshable Braille Displays, http://www.afb.org/ProdBrowseCatResults.asp?CatID=43

5 WebAIM, Screen Reader User Survey #3 Results, Primary Screen Reader http://webaim.org/projects/screenreadersurvey3/#primary

6 Freedom Scientific, JAWS for Windows Screen Reading Software, http://www.freedomscientific.com/products/fs/jaws-product-page.asp

7 GW Micro, Window-Eyes, http://www.gwmicro.com/Window-Eyes/

8 NVDA Project, http://www.nvda-project.org/

9 Apple, Accessibility, VoiceOver in Depth, http://www.apple.com/accessibility/voiceover/

10 Vischeck, Colorblindness Simulator, http://www.vischeck.com/

[11] Miłosz Polak, „Dostępność stron internetowych", praca licencjacka na Wydziale Zarządzania i Komunikacji Społecznej UJ, Kraków, 2006.

[12] Joanna Bobko, „Przykładowy kurs e-learningowy przy użyciu narzędzi dostępnychna platformie e-learningowej Moodle: Dostępność stron internetowych", praca magisterska na Wydziale Zarządzania i Komunikacji Społecznej UJ,

Kraków, 2011.
[13] http://www.w3.org/WAI/
[14] http://www.w3.org/TR/WAI-AUTOOLS/
[15] http://www.w3.org/TR/UAAG20/
[16] http://www.w3.org/TR/WCAG20/
[17] http://www.wcag.pl/wytyczne-wcag/8-wytyczne-wcag-20 - polskie tłumaczenie wytycznych WCAG.
18 Może się zdarzyć, że klawisz enter zadziała. Będzie to jednak implementacja konkretnej przeglądarki. Obecnie specyfikacje nie regulują tego zachowania w żaden sposób.
19 The Paciello Group Blog, *JAWS Support for ARIA*, http://www.paciellogroup.com/blog/2010/10/jaws-support-for-aria/;
The Paciello Group Blog, *HTML5 Accessibility Chops: ARIA landmark support*, http://www.paciellogroup.com/blog/2011/07/html5-accessibility-chops-aria-landmark-support/
20 HTML5 Accessibility, *ARIA Landmark role support tests*, http://www.html5accessibility.com/tests/landmarks.html
21 The Chromium Projects, *Accessibility*, https://sites.google.com/a/chromium.org/dev/developers/design-documents/accessibility#TOC-WAI-ARIA-Support
22 W3C, *The Roles Model*, http://www.w3.org/TR/wai-aria/roles
23 W3C, *Supported States and Properties*, http://www.w3.org/TR/wai-aria/states_and_properties
24 J. Edwards, *Ajax and Screenreaders: When Can it Work?*, http://www.sitepoint.com/ajax-screenreaders-work/
25 W3C, *WAI-ARIA 1.0 Authoring Practices*, http://www.w3.org/WAI/PF/aria-practices/#liveprops
26 WebAIM, *Screen Reader User Survey #3 Results*, ARIA Landmarks
http://webaim.org/projects/screenreadersurvey3/#landmarks
27 The Paciello Group, Web Accessibility Toolbar, http://www.paciellogroup.com/resources/wat-ie-about.html
28 The Paciello Group, Conrast Analyser, http://www.paciellogroup.com/resources/contrast-analyser.html
29 WebAIM, WAVE – Web Accessibility Evaluation Tool, http://wave.webaim.org
[30] Lista przetestowanych projektów WWW w ramach projektu „Wsparcie osób niepełnosprawnych w swobodnym dostępie do informacji i usług zamieszczonych w Internecie", http://dostepnestrony.pl/wp-content/uploads/2011/02/Lista-podmiot%C3%B3w-i-ich-stron-internetowych.pdf
31 A. Gustafson, *Adaptive Web Design*, Easy Readers 2011, strona 100
32 Nomensa, *Accessible Media Player*, http://www.nomensa.com/web-accessibility/accessible-media-player
33 uiAccess, Transcription Services, http://www.uiaccess.com/transcripts/transcript_services.html
34 dotSUB, http://dotsub.com/
35 Longtailvideo, JW Player, http://www.longtailvideo.com/support/jw-player/jw-player-for-flash-v5
36 The Workshop, Video Player, http://www.theworkshop.co.uk/video-player
37 W3C, Pause, Stop, Hide: Understanding SC 2.2.2, http://www.w3.org/TR/UNDERSTANDING-WCAG20/time-limits-pause.html ; W3C, Audio Control: Understanding SC 1.4.http://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-dis-audio.html
W3C, Resize text: Understanding SC 1.4.4, http://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-scale.html
38 WebAIM, Screen Reader User Survey #3 Results, http://webaim.org/projects/screenreadersurvey3/#javascript
[39] Jon Duckett, "XHTML i CSS. Dostępne witryny internetowe", wydawnictwo Helion, Gliwice, 2008.
[40] Projekt realizowany przez Państwowy Fundusz Rehabilitacji Osób Niepełnosprawnych i przez Stowarzyszenie Przyjaciół Integracji: „Wsparcie osób niepełnosprawnych w swobodnym dostępie do informacji i usług zamieszczonych w Internecie", http://dostepnestrony.pl/projekt/