POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Patryk Studniak
Nr albumu s16183

Reorganizacja struktury stron i aplikacji internetowych pod względem kompatybilności z popularnymi czytnikami ekranów dla niepełnosprawnych.

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, czerwiec 2018

Streszczenie

Temat dostępności sieci i kompatybilności stron internetowych z urządzeniami asystującymi, w ostatnich latach zyskał na popularności, dzięki nowej interpretacji ustawy amerykańskiego prawa *Americans with Disabilities Act (ADA)*. Odświeżona wersja ustawy, uznaje strony internetowe za miejsca publiczne i koniecznym jest zapewnienie równego dostępu do wszystkich oferowanych przez nie funkcjonalności, bez względu na rasę, wyznanie, zdolności poznawcze czy stopień niepełnosprawności.

Praca obejmuje zagadnienia związane ze standardami dostępności sieci, w kontekście kompatybilności struktur stron internetowych z popularnymi czytnikami ekranu.

Zawiera prezentację i omówienie standardów dostępności sieci definiowanych przez *W3C*. Przedstawia korzyści, które niesie pełna implementacja zdefiniowanych zasad oraz możliwe konsekwencje wynikające z jej braku.

Projekt pracy magisterskiej składa się z dwóch części. Całość projektu jest niekonwencjonalną i innowacyjną propozycją rozwiązania najczęściej powtarzanych błędów i zaniedbań w implementacji standardu *Web Content Accessibility Guidelines 2.0 (WCAG 2.0)*. Oferuje reorganizację struktury strony dla lepszego wsparcia czytników ekranu.

Pierwszą częścią projektu jest prosta w użyciu biblioteka języka *JavaScript*, zaprojektowana i zaprogramowana na potrzeby pracy, zawierająca pełną funkcjonalność reorganizacji struktury *HTML* strony internetowej.

Druga część projektu, którą jest dedykowane rozszerzenie narzędzi deweloperskich przeglądarki *Google Chrome*, umożliwia łatwe wykorzystanie możliwości biblioteki, bez znajomości technik programistycznych i zagadnień szeroko pojętej inżynierii oprogramowania.

Generyczne rozwiązanie zaprezentowane w ramach projektu pracy, jest odpowiedzią na potrzeby programistów i właścicieli witryn internetowych, borykających się z brakiem, bądź błędami w implementacji standardów dostępności sieci, które coraz częściej są egzekwowane z ramienia prawa.

Słowa kluczowe

Sieć, standard WCAG 2.0, dostępność sieci, reorganizacja, HTML, JavaScript, czytniki ekranu, oprogramowanie asystujące, niepełnosprawność

Spis treści

1. Wstęp

Jak wspomina dyrektor *World Wide Web Consortium (W3C)* oraz wynalazca WWW Tim Berners-Lee [2], sieć została zaprojektowana w sposób dający możliwość korzystania z jej zawartości i funkcjonalności wszystkim osobom, bez względu na oprogramowanie, możliwości sprzętowe czy lokalizację. Kiedy strona internetowa spełnia ten cel, jest dostępna dla wszystkich osób w takim samym stopniu bez względu na zakres słuchu, wzroku czy zdolności poznawczych. Sieć wprowadza radykalne zmiany w postrzeganiu niepełnosprawności usuwając bariery w komunikacji i interakcji, z którymi nawet dziś spotykamy się w fizycznym świecie. Niestety kiedy technologie i usługi są źle zaprojektowane mogą ponownie tworzyć bariery wykluczające ludzi z korzystania z sieci. Dbanie o jakość i stopień dostępności aplikacji internetowych jest niezwykle istotne dla programistów pragnących dostarczać najwyższej jakości programy burzące bariery a nie wykluczające kolejnych użytkowników. Zapewne tego pragnął twórca sieci WWW.

Zapis amerykańskiego prawa ADA (Americans with Disabilities Act), wymusza zarówno na dużych korporacjach jak i mniejszych przedsiębiorstwach prowadzących działalność w obszarze Internetu, dostosowanie wykorzystywanych komercyjnie produktów do standardu dostępności aplikacji internetowych (Web Content Accessibility Guidelines WCAG-2) [3].

Web Content Accessibility Guidelines WCAG-2 są szerokim zakresem wytycznych obejmujących udostępnianie treści w sieci. Przestrzeganie zasad zdefiniowanych w standardzie WCAG-2 sprawia, że treść zawarta w sieci jest dostępna dla większej ilości osób niepełnosprawnych, w tym słabowidzących, głuchych oraz dla osób z ograniczonymi funkcjami poznawczymi czy trudnościami w zrozumieniu treści. Specyfikacje WCAG-2 są zdefiniowane jako zasady testowalne i niespecyficzne dla danej technologii. Standard WCAG-2 jest zgodny z wytycznymi opublikowanymi w 1999 roku w ramach standaryzacji WCAG-1 przez organizację W3C. Rekomendowanym przez World Wide Web Consortium standardem jest WCAG-2.0 [4].

W sytuacji, kiedy podmiot gospodarczy jest prawnie zmuszony wprowadzić aktualizacje poprawiające dostępność strony internetowej, pod groźbą otrzymania wysokiej kary finansowej, należy zastanowić się nad rozwiązaniem wymagającym jak najmniejszego nakładu pracy i nie droższym w implementacji niż zasądzona grzywna.

W takiej sytuacji znajdowały się w ubiegłych latach amerykańskie korporacje z branży hotelowej. Ich propozycją rozwiązania problemu było wykorzystanie firm zewnętrznych oferujących przeprowadzanie testów A/B aplikacji internetowych. Jeden z wariantów testu, serwowany dla stu procent odwiedzających, za pomocą wstrzykniętego kodu JavaScript zmieniał strukturę markupu aplikacji, poprawiając jej dostępność. Rozwiązanie było skuteczne, ale nie należało do tanich. Dzięki badaniom rynku oraz możliwościom modyfikacji kodu HTML za pomocą języka JavaScript, rozwiązanie zaproponowane w związku z tematem pracy magisterskiej może okazać się atrakcyjne dla wspomnianych przedsiębiorstw.

Rozwiązaniem jest biblioteka JavaScript o małym rozmiarze, zawierająca zestaw metod pomagających między innymi w poprawie jakości strony pod względem nawigacji klawiszowej, atrybutów ARIA, atrybutów HTML oraz meta danych. Biblioteka może być używana przez dowolną stronę internetową. Dodatkowym atutem rozwiązania jest możliwość konfiguracji strony poprzez załadowanie pliku konfiguracyjnego w formacie JSON lub JavaScript. Umożliwia to wykorzystanie biblioteki bez konieczności programowania. Aby jeszcze bardziej ułatwić tworzenie plików konfiguracyjnych projekt pracy magisterskiej oferuje rozszerzenie narzędzi developerskich przeglądarki

Chrome, dzięki któremu możemy za pomocą przyjaznego i intuicyjnego interface'u wygenerować i pobrać plik konfiguracyjny. Następnie plik ten możemy dołączyć do kodu naszej aplikacji. Takie rozwiązanie wymaga jednorazowej implementacji bez konieczności modyfikacji istniejącego kodu. Aktualizacja z wykorzystaniem biblioteki może zostać zaimplementowana przez każdą osobę znającą w podstawowym stopniu technologie informatyczne.

1.1. Wprowadzenie do amerykańskiej regulacji prawnej "Americans with Disabilities Act"

ADA jest skróconą nazwą amerykańskiej ustawy Americans with Disabilities Act. Została ona uchwalona jako prawo w Stanach Zjednoczonych w 1990 roku. Jest to ustawa prawa obywatelskiego regulująca kwestie związane z zakazem dyskryminacji osób niepełnosprawnych we wszystkich sferach życia publicznego. Jako sfery życia publicznego zapis prawa ADA definiuje miejsca pracy, szkoły, transport oraz wszystkie otwarte dla ogółu społeczeństwa miejsca publiczne i prywatne. Ostatni z wymienionych przykładów sfer życia publicznego, swoim zasięgiem obejmuje również wszelkiego rodzaju publiczne strony, aplikacje i serwisy internetowe, w szczególności te oferujące konkretne usługi. Głównym założeniem ustawy jest zapewnienie osobom niepełnosprawnym takich samych praw i możliwości jakie są oferowane wszystkim innym. ADA zapewnia ochronę praw obywatelskich podobną do tej, którą amerykańskie prawo oferuje w związku z dyskryminacją osób ze względu na wiek, płeć, rasę, kolor skóry, pochodzenie czy religię. Gwarantuje równe szanse osobom niepełnosprawnym w miejscach publicznych, zatrudnieniu, transporcie, usługach samorządowych oraz telekomunikacji [5].

Istotną informacją jest to, że w kontekście prawa ADA i tej pracy magisterskiej, niepełnosprawność jest terminem prawnym a nie medycznym. Ponieważ w niektórych prawach amerykańskich jako definicję niepełnosprawności przyjmuje się termin medyczny, definicja niepełnosprawności ADA może się różnić. ADA definiuje osobę niepełnosprawną jako osobę z upośledzeniem fizycznym lub umysłowym, znacznie ograniczającym co najmniej jedną czynność życiową. Oznacza to również, że w świetle zapisu ADA osoba może być traktowana jako niepełnosprawna nawet jeśli została uznana medycznie za osobę w pełni sprawną. Definicja niepełnosprawności ADA zabrania również dyskryminacji osób będących w stałych związkach z osobami o określonym stopniu niepełnosprawności [6].

1.1.1 Czym jest ADA?

ADA jest skróconą nazwą amerykańskiej ustawy Americans with Disabilities Act. Została ona uchwalona jako prawo w Stanach Zjednoczonych w 1990 roku. Jest to ustawa prawa obywatelskiego regulująca kwestie związane z zakazem dyskryminacji osób niepełnosprawnych we wszystkich sferach życia publicznego. Jako sfery życia publicznego zapis prawa ADA definiuje miejsca pracy, szkoły, transport oraz wszystkie otwarte dla ogółu społeczeństwa miejsca publiczne i prywatne. Ostatni z wymienionych przykładów sfer życia publicznego, swoim zasięgiem obejmuje również wszelkiego rodzaju publiczne strony, aplikacje i serwisy internetowe, w szczególności te oferujące konkretne usługi. Głównym założeniem ustawy jest zapewnienie osobom niepełnosprawnym takich samych praw i możliwości jakie są oferowane wszystkim innym. ADA zapewnia ochronę praw obywatelskich podobną do tej, którą amerykańskie prawo oferuje w związku z dyskryminacją osób ze względu na wiek, płeć, rasę, kolor skóry, pochodzenie czy religię. Gwarantuje równe szanse osobom niepełnosprawnym w miejscach publicznych, zatrudnieniu, transporcie, usługach samorządowych oraz telekomunikacji [5].

Istotną informacją jest to, że w kontekście prawa ADA i tej pracy magisterskiej, niepełnosprawność jest terminem prawnym a nie medycznym. Ponieważ w niektórych prawach amerykańskich jako definicję niepełnosprawności przyjmuje się termin medyczny, definicja niepełnosprawności ADA może się różnić. ADA definiuje osobę niepełnosprawną jako osobę z upośledzeniem fizycznym lub umysłowym, znacznie ograniczającym co najmniej jedną czynność życiową. Oznacza to również, że w świetle zapisu ADA osoba może być traktowana jako niepełnosprawna nawet jeśli została uznana medycznie za osobę w pełni sprawną. Definicja niepełnosprawności ADA zabrania również dyskryminacji osób będących w stałych związkach z osobami o określonym stopniu niepełnosprawności [7].

Ustawa prawa obywatelskiego ADA dzieli się na pięć tytułów (sekcji), które odnoszą się do różnych obszarów życia publicznego.

- Z tytułu pierwszego otrzymujemy regulacje prawne na temat zatrudnienia osób niepełnosprawnych. Dzięki temu zapisowi osoby niepełnosprawne mają mieć te same szanse na zatrudnienie oraz otrzymywanie świadczeń socjalnych co w pełni sprawni. Pracodawcy muszą zapewnić odpowiednie warunki pracy wykwalifikowanym pracownikom. Miejsce pracy musi być dopasowane do rodzaju i stopnia niepełnosprawności, aby umożliwić przeprowadzenie rozmowy kwalifikacyjnej oraz dalsze wykonywanie pracy. Ta część ustawy jest egzekwowana przez amerykańską komisję do spraw równych szans dla zatrudnienia (U.S. Equal Employment Opportunity Commission).
- Z tytułu drugiego zapisu ustawy ADA, osoby niepełnosprawne otrzymują gwarancję braku dyskryminacji we wszystkich służbach państwowych i lokalnych. Regulacja dotyczy wszystkich programów, działań i usług podmiotów publicznych. Dotyczy to rządów stanowych i lokalnych, ich departamentów i agencji. W tym tytule określono wszystkie procedury administracyjne, których należy przestrzegać.
- Sekcja trzecia jest regulacją w zakresie dostępu do miejsc publicznych. Obejmuje między innymi placówki publiczne i prywatne takie jak hotele, restauracje, sklepy detaliczne, gabinety lekarskie, szkoły, stadiony sportowe, kina i tym podobne. Określa minimalne standardy dostępności dla osób niepełnosprawnych do istniejących oraz powstających miejsc. Tytuł ten jest egzekwowany przez Wydział Sprawiedliwości Stanów Zjednoczonych (U.S. Department of Justice).
- Sekcja czwarta jest oświadczeniem o zapewnieniu usług telekomunikacyjnych osobom niepełnosprawnym, w szczególności tym z zaburzeniami mowy i słuchu.
- Ostatni wpis do ustawy, tytuł piąty zawiera różnorodne przepisy dotyczące ADA jako całości, w tym związek ustawy z innymi przepisami takimi jak immunitet, wpływ ustawy na ubezpieczycieli i świadczenia, zakres działań odwetowych i przymusowych. Z wymienionych zapisów wynika, że każda sfera życia publicznego powinna być dostępna w równym stopniu wszystkim osobom bez względu na niepełnosprawność.

1.1.2 Rola ADA w projektowaniu aplikacji internetowych.

Zapis trzeci ustawy ADA obejmuje swoim zasięgiem wszystkie miejsca publiczne, a więc również ich reprezentację w sieci. Strony i serwisy internetowe firm informacyjnych bądź oferujących usługi, z których może korzystać ogół społeczeństwa, powinny być dostępne dla wszystkich bez względu na stopień sprawności. Od stycznia 2018 roku wszystkie takie strony internetowe powinny być dostosowane do standardów zdefiniowanych przez W3C, WCAG-2.0.

Standard WCAG-2.0 oraz amerykańskie prawo ADA jest coraz wyraźniej egzekwowane zarówno na małych firmach jak i dużych korporacjach ze względu na popularność oraz skalę usług internetowych. Coraz więcej firm oferuje korzystanie ze swoich usług wyłącznie drogą internetową. Gdyby ich serwisy i aplikacje nie były tworzone zgodnie z pewnymi standardami, definiowanymi przez globalne organizacje, trudno byłoby tworzyć uniwersalne produkty dające możliwość używania aplikacji przez osoby niepełnosprawne. Przykładem mogą być tutaj czytniki ekranów, które interpretują odpowiednio przypisane znaczniki oraz atrybuty dokumentu HTML, dzięki temu umożliwiając pobieranie usług osobom niepełnosprawnym.

Brak dostosowania strony internetowej do wspomnianych standardów jest coraz częściej karane wysokimi grzywnami. Jako przykład takiej egzekucji może posłużyć amerykańska firma Winn-Dixie, która została oskarżona przez osobę niewidomą o brak możliwości skorzystania drogą internetową z usług oferowanych przez firmę. Proces trwał bardzo długo, ostatecznie firma została obciążona grzywną oraz przymusem dostosowania strony do standardów WCAG-2.0, co kosztowało ją ponad dwieście pięćdziesiąt tysięcy dolarów [8].

Firmy podążające za standardami WCAG-2.0, które są wymogiem ustawy ADA, są określane jako ADA Complient, zgodne z ustawą ADA. Rola amerykańskiego prawa w projektowaniu i programowaniu aplikacji internetowych odgrywa w dzisiejszych czasach bardzo dużą rolę. Jeśli kiedykolwiek dana firma planuje działania na amerykańskim rynku, powinna wziąć to pod uwagę już na etapie planowania projektu. Dodatkowo, standard WCAG-2.0 jest coraz częściej wymagany również na rynku europejskim. Aby tworzony produkt, strona czy aplikacja internetowa były zgodne z ustawą ADA, należy uwzględnić sześćdziesiąt jeden zasad zawartych w WCAG-2.0. Wiele z nich strony internetowe zawierają z definicji, natomiast do wielu należy się dostosować. Do najistotniejszych należą nawigacja klawiszowa, teksty alternatywne i tytuły. Te ostatnie tłumaczą czym są oraz do czego służą pola, na których znajduje się robot interpretujący.

Istotną rolę odgrywa tutaj standard WAI-ARIA (Accessible Rich Internet Applications), który udostępnia zestaw znaczników, dzięki któremu możemy definiować role poszczególnych tagów HTML. Dzięki nim możemy świadomie definiować sposób w jaki znaczniki zostaną zaprezentowane użytkownikowi przez czytnik ekranu. Strona powinna również obsługiwać powiększenie tekstu o minimum dwieście procent bez konieczności przesuwania ekranu w poziomie oraz zawierać odpowiednie kontrasty kolorystyczne. O ile wprowadzenie standardu WCAG-2.0 nie powinno stanowić dużego wyzwania w trakcie tworzenia produktu, tak modyfikacja istniejących stron może okazać się problematyczna, a często wręcz niemożliwa, zarówno z technicznego jak i biznesowego punktu widzenia.

1.2. Czytniki ekranu

Przed zaprezentowaniem najpopularniejszych czytników ekranu stosowanych do interpretacji i prezentacji głosowej stron internetowych warto zapoznać się z definicją i głównymi założeniami czytników ekranu.

Czytnik ekranu (ang. *Screen reader*) [29] to program komputerowy, który rozpoznaje i interpretuje informacje wyświetlane na monitorze. Jest odpowiedzialny za ich przetwarzanie i prezentację użytkownikowi w formie głosowej lub wysyłanie sygnału do brajlowskiego urządzenia zewnętrznego.

Jest to technologia umożliwiająca użytkowanie systemów operacyjnych i wszystkich funkcjonalności z nim związanych osobom niewidomym, niedowidzącym lub prowadzącym samochód czy obsługującym kilka urządzeń jednocześnie. Wiele systemów operacyjnych zawiera wbudowane czytniki ekranu.

Są to między innymi *Narrator* dla *MS Windows* czy *VoiceOver* dla *Mac OS X*. Najczęściej używanymi nadal są rozwiązania komercyjne, takie jak *JAWS* czy *Windows-eyes*, jednak dynamiczny rozwój programów open-source powoduje ogromny wzrost zainteresowania alternatywnymi produktami oferującymi podobne doświadczenia.

Dużą popularnością cieszą się również technologie wspomagające, działające w środowisku przeglądarek. Mowa tutaj o rozszerzeniach mających możliwość interpretacji i prezentacji kodu źródłowego strony internetowej. Dzięki takim technologiom bardzo ułatwiono programistom testy i określenie stopnia dostępności ich produktu.

W ramach pracy magisterskiej, szczególną uwagę skupiono na jednym z niekomercyjnych rozwiązań open-source jakim jest rozszerzenie przeglądarki *Google Chrome*, *ChromeVox*. Za pomocą *ChromeVox* zostały przeprowadzone badania oraz testy rozwiązania prezentowanego w ramach projektu pracy.

1.2.1 Przegląd najpopularniejszych czytników ekranu.

Według badań przeprowadzonych na ponad dwóch i pół tysiącach aktywnych użytkowników czytników ekranów przez *WebAIM (Web Accessibility in Mind)* w lipcu 2015 roku, do najpopularniejszych czytników ekranu, wspomagających interpretację kodu HTML stron internetowych, należą *JAWS (30,2%)*, *ZoomText (22,2%)*, *Window-eyes (20,7%)*, *NVDA (14,6%)*, *VoiceOver (7,6%)*, *System Access (1,5%)* oraz *ChromeVox(~3%)*, który wraz z pozostałymi rozwiązaniami zdobył ponad trzy procent głosów. Badanie było kontynuacją ankiety *WebAIM Screen Reader* przeprowadzanej w 2009 roku, w grudniu 2010, w maju 2012 oraz w styczniu 2014. Jak podaje autor ankiety, badania przeprowadzone w 2015 roku otrzymały najwyższą liczbę ważnych odpowiedzi [9].

WebAIM ***(Web Accessibility in Mind)***, od 1999 roku, jest firmą zapewniającą kompleksową edukację w zakresie dostępności stron internetowych. Doświadczenie zdobywane latami sprawiło, że firma jest wiodącym dostawcą wiedzy na temat dostępności sieci na świecie. Jest to organizacja non-profit z siedzibą w *Utah State University* [10].

JAWS *(Job Access With Speech)* to najpopularniejszy na świecie czytnik ekranu zaprojektowany z myślą o osobach, którym trwała utrata wzroku uniemożliwiła oglądanie zawartości ekranu i nawigację przy pomocy kursora myszy. Oprogramowanie zapewnia prezentację słowną oraz obsługę sygnałów brajlowskich urządzeń zewnętrznych. Czytnik ekranu zawiera dwa wielojęzyczne syntezatory mowy, *Eloquence* i *Vocalizer Expressive*. Poza czytaniem zawartości ekranu systemu operacyjnego, oferuje on odczyt dokumentów HTML (stron internetowych) oraz dokumentów tekstowych (PDF). *JAWS* jest w pełni kompatybilny z pakietem *Microsoft Office*, co daje mu ogromną przewagę nad konkurencją [11].

ZoomText Magnifier to w pełni zintegrowany z systemem operacyjnym program służący do powiększania i czytania elementów ekranu. Został zaprojektowany dla osób ze znacznie ograniczoną widocznością. Jego głównymi założeniami jest wprowadzenie odpowiednich kontrastów kolorystycznych oraz poprawienie jakości elementów znajdujących się na ekranie. Posiada funkcję odzwierciedlania głosowego wpisywanych znaków oraz automatyczne odczytywanie stron internetowych za pomocą interpretacji kodu źródłowego. Podczas powiększania tekstu jego klarowność ulega znacznemu pogorszeniu. W rezultacie wiele czcionek jest trudnych, a czasem niemożliwych do

odczytania. Opracowana i zaimplementowana przez producenta technologia *xFont* wyświetla tekst w wysokiej rozdzielczości, który jest łatwy do odczytania przy wszystkich poziomach powiększenia. Skondensowane ustawienia pozwalają precyzyjnie dostroić grubość i odstępy pomiędzy tekstami tak, aby zapewnić możliwie najlepszą czytelność [12].

Window-eyes, oprogramowanie produkowane przez firmę *GW Micro Inc*. pozwala osobie niewidomej korzystać z komputera działającego z systemem *MS Windows*. Osoba niewidoma używa klawiszowych urządzeń peryferyjnych do nawigacji i obsługi systemu komputerowego natomiast wszelkie informacje, ostrzeżenia i powiadomienia, na które napotyka użytkownik są odczytywane przez zintegrowany syntezator mowy lub przesyłane do zewnętrznego urządzenia z wyświetlaczem brajlowskim. Dzięki temu przeglądanie Internetu, odtwarzanie muzyki lub obsługa poczty przez osobę niewidomą staje się dużo łatwiejsza [13].

NVDA *(Non Visual Desktop Access)* to pierwsza z niekomercyjnych propozycji programu czytającego ekrany systemów operacyjnych i przeglądarek. *NVDA* jest projektem open-source i jest darmowy. Działa podobnie jak *JAWS* czy *Window-eyes*, oferując zbliżonej jakości usługi. Niestety, co za tym idzie, posiada również podobne problemy. Za jego pomocą możemy przeglądać strony internetowe w aplikacji *Internet Explorer* lub *Mozilla Firefox*. Umożliwia także czytanie, tworzenie i edycję dokumentów tekstowych oraz wysyłanie i odbieranie poczty za pomocą klientów pocztowych. Istotną opcją jest możliwość korzystania z wiersza poleceń [14].

VoiceOver jest programem interpretującym obraz wbudowanym dla systemu *Mac OS X*. Z pewnością ułatwi korzystanie z komputera osobom niewidomym lub słabowidzącym. Najnowsza wersja programu obejmuje obsługę gestów, urządzenia brajlowskie, wskazówki głosowe oraz opcje pomocy technicznej [15].

ChromeVox jest bezpłatnym czytnikiem ekranu firmy *Google* zaprogramowanym jako rozszerzenie przeglądarki internetowej *Google Chrome*. Czytnik został zaprojektowany do ułatwienia korzystania z serwisów internetowych osobom niewidomym i niedowidzącym. Jego czołową funkcjonalnością jest prezentacja głosowa zawartości strony internetowej. W odróżnieniu od konkurencyjnych propozycji *ChromeVox* jest zbudowany wyłącznie z użyciem technologii przeglądarkowych, *HTML5*, *CSS3* oraz *JavaScript*. Rozszerzenie od samego początku było tworzone z myślą o ułatwieniu dostępu do aplikacji internetowych wykorzystujących standardy WCAG-2.0 oraz WAI-ARIA aby zapewnić jak najlepsze doświadczenia użytkownikowi. Prosta i intuicyjna nawigacja jest łatwa do nauczenia i pozwala na szybkie rozpoczęcie korzystania ze wszystkich benefitów oferowanych przez program. *ChromVox* jest wciąż rozwijany i niestety jeszcze nie jest kompatybilny z dostępnymi na rynku czytnikami ekranu. Aby korzystać z rozszerzenia *ChromeVox* należy wyłączyć inne czytniki ekranu. Istnieje również wersja desktopowa programu *ChromeVox*, która jest wbudowanym czytnikiem ekranu Chromebook'ów. Wyróżnia się spośród innych czytników ekranu łatwością obsługi i dzięki temu jego popularność wśród użytkowników czytników ekranu sukcesywnie wzrasta [16].

1.2.2 Kompatybilność stron internetowych z czytnikami ekranu.

W postrzeganiu większości osób, dostępność stron internetowych oznacza ich kompatybilność z czytnikami ekranu. Programista musi zatem przestrzegać szeregu zasad, które niestety bardzo często są pomijane. Wiele z atrybutów wspomagających rozpoznawanie wyświetlanej treści przez oprogramowanie interpretujące obraz bądź kod źródłowy strony internetowej, jest wymagane również w standardach *SEO (Search Engine Optimization)*.

SEO według definicji *SEMTEC*[17] to proces, którego celem jest zwiększenie pozycji strony w darmowych wynikach wyszukiwania. Zajmowanie wysokich miejsc w wynikach wyszukiwania znacznie zwiększa liczbę odwiedzin strony internetowej. Ilość odwiedzin wpływa na wzrost popularności strony internetowej dając właścicielowi możliwość zbudowania niepowtarzalnej i popularnej marki i zwiększenie potencjalnego dochodu płynącego z prezentowanej treści. SEO swoim zasięgiem obejmuje optymalizację strony pod kątem silników wyszukiwarek. Za granicą, definicja optymalizacji SEO obejmuje również pozycjonowanie w wynikach wyszukiwania. W Polsce to wprowadzanie poprawek do strony i dopiero w połączeniu z pozyskiwaniem odnośników do strony składa się na pozycjonowanie. Jest to popularna strategia marketingu internetowego próbująca „przewidywać" zachowanie użytkowników oraz wykorzystująca słowa kluczowe.

Dbanie o kompatybilność naszej strony internetowej z popularnymi czytnikami ekranu może znacząco wpływać na pozycję w wynikach wyszukiwania w przeglądarkach internetowych.

Firma *WebAIM* zajmująca się profesjonalnie dostępnością stron internetowych szczególnie pod względem kompatybilności z czytnikami ekranu, opublikowała artykuł omawiający projektowanie z uwzględnieniem czytników ekranu. Czytniki ekranu konwertują tekst cyfrowy za pomocą syntezatora mowy umożliwiając użytkownikowi odsłuchanie treści. Technologia daje możliwość osobom niewidomym lub słabowidzącym korzystanie ze wszystkich funkcjonalności oferowanych przez serwisy internetowe wspierające obsługę głosową z taką samą niezależnością i prywatnością jak wszystkim pozostałym. Osoby z wadami wzroku nie są jedynymi odbiorcami czytników ekranu. Programy okazują się być bardzo przydatne osobom z pewnymi niepełnosprawnościami poznawczymi lub borykającymi się z problemami z właściwą interpretacją treści. Wielokrotnie czytniki ekranu są również wykorzystywane przez w pełni sprawne osoby, na przykład podczas prowadzenia samochodu bądź w trakcie nauki, kiedy użytkownik preferuje treści audio od tych cyfrowych. Czytniki ekranu interpretują treść zupełnie inaczej niż ludzie. Ich syntezatory zwykle brzmią bardzo monotonnie i mechanicznie. Doświadczeni użytkownicy dzięki możliwościom dostosowywania, wbudowanym w programy interpretujące, przyspieszają wielokrotnie szybkość czytania tak, aby brzmiała mniej mechanicznie[18].

Podczas projektowania strony internetowej pod względem kompatybilności z popularnymi czytnikami ekranu należy pamiętać o kluczowych elementach, którymi są język dokumentu, linearyzacja zawartości, nawigacja klawiszowa, uwzględnienie różnic pomiędzy czytnikami ekranu oraz sposób interpretacji strony internetowej przez czytnik ekranu.

Każdy język istniejący na świecie posiada swoje własne zasady wymowy. Jeśli czytnik posiada wielojęzyczny syntezator mowy należy w jakiś sposób poinformować program w jakim języku powinien czytać tekst. Strony internetowe mają możliwość zdefiniowania języka całego dokumentu za pomocą atrybutu *lang* w znaczniku *html* dokumentu. Ułatwia to również automatyczne tłumaczenie treści. Często strony internetowe zawierają treści przeplatające wyrazy lub całe zdania w różnych językach. W takich przypadkach atrybut *lang* jest stosowany do znaczników okalających takich jak *span*. Daje to sygnał czytnikowi ekranu o tymczasowej zmianie języka syntezatora mowy. Atrybut *lang* nie nadaje żadnych walorów stylistycznych znacznikowi, dla którego został zdefiniowany, jednak może stanowić odnośnik podczas budowy selektora np. *span[lang]*.

Interfejsy audio takie jak syntezatory mowy prezentują zawartość stron internetowych liniowo, po jednym elemencie na raz. Bardzo wyraźnie kontrastuje to ze sposobem w jaki większość ludzi interpretuje treści. Widzący użytkownicy są w stanie zinterpretować całą treść jako obraz, poznając ogólny styl, układ artystyczny i pozostałe aspekty wizualne pomagające w zrozumieniu oraz właściwej interpretacji treści. Liniowy postęp w treści czytników ekranu przypomina zautomatyzowany system

odczytu elementów menu telefonów komórkowych, gdzie każda z dostępnych opcji jest odczytywana kolejno. Informacja o liniowej interpretacji i prezentacji interfejsu przez czytniki ekranu może być kluczowa podczas planowania projektu oraz budowania architektury serwisu internetowego.

Pomimo liniowego charakteru interpretacji kodu źródłowego stron internetowych, czytniki ekranu dostarczają kilka sposobów szybkiej nawigacji po treści strony.

Jednym ze sposobów nawigacji klawiszowej jest nawigacja po odnośnikach strony. Za pomocą klawisza *tab* użytkownik ma możliwość przemieszczania się między kolejnymi linkami zgodnie z kolejnością w jakiej zostały umieszczone w strukturze *DOM (Document Object Model)*.

Często elementy na stronie są umiejscowione w niestandardowy sposób a ich położenie w oknie przeglądarki jest zdefiniowane za pomocą kaskadowych arkuszy styli. Taka modyfikacja stylistyczna może być powodem niezrozumienia zawartości strony oraz sposobu nawigacji. Aby uniknąć nieporozumień i zaprezentować użytkownikom jak najlepsze i intuicyjne doświadczenia, możemy użyć kolejnego sposobu definiowania nawigacji klawiszowej, atrybutów *tabindex*.

Atrybut *tabindex* jak podaje dokumentacja *MDN (Mozilla Developer Network)* [19] jest „integerem wskazującym czy dany element może zostać wyszczególniony (jest *fokusyjny*) oraz czy powinien być brany pod uwagę podczas nawigacji sekwencyjnej klawiatury i jeśli tak to na jakiej pozycji" [20]. Zwykle atrybut może przyjmować kilka wartości, w zależności od których nadaje odpowiednie właściwości elementowi, dla którego został zdefiniowany. Wartość może być negatywna (*negative value*) co oznacza, że element z przypisanym atrybutem o danej wartości jest *fokusyjny* ale nie osiągalny w nawigacji klawiszowej. Wartość zero (*0*) oznacza, że element z atrybutem o danej wartości jest osiągalny za pomocą nawigacji klawiszowej jednak jego kolejność jest zdefiniowana przez przeglądarkę. Zwykle jest to sekwencyjna kolejność znaczników w strukturze *DOM*. Wartość pozytywna atrybutu *tabindex* oznacza, że jest to element osiągalny w nawigacji klawiszowej a jego kolejność jest zdefiniowana przez wartość atrybutu.

Kolejnym sposobem ustandaryzowania sposobu nawigacji klawiszowej jest prezentacja nagłówków. Dzięki temu użytkownicy mogą usłyszeć zarys najistotniejszej zawartości znajdującej się na stronie internetowej aby następnie wybrać treść, do której chcą się cofnąć aby zgłębić dany temat. Wadą tej techniki jest poleganie na nagłówkach, których wielu stronom internetowym niestety brakuje bądź są ustawiane niezgodnie ze standardami.

Sekwencyjna nawigacja klawiaturą może być wspomagana atrybutami *ARIA Landmarks*. **Atrybuty *ARIA*** jak podaje organizacja *W3C* [21] stanowią skuteczny sposób identyfikacji organizacji i strukturyzacji strony internetowej. Informacje strukturalne przekazywane wizualnie użytkownikom powinny być reprezentowane programowo w znacznikach, przy użyciu odpowiednich ról.

Wykorzystanie atrybutów z poprawnie zdefiniowanymi rolami umożliwia nawigację klawiszową po wybranych elementach oraz zaprezentowanie zawartości strony we właściwy sposób za pomocą czytnika ekranu, który potrafi interpretować atrybuty. Zdefiniowanie oraz przypisanie odpowiednich ról strukturze strony internetowej jest zadaniem dla programisty. Kolejnymi krokami, które powinny zostać wykonane, jest podzielenie struktury strony na fragmenty, które spełniają konkretną rolę na stronie internetowej. Może być to między innymi nawigacja, baner, główny kontener czy pole wyszukiwania. Kolejnym krokiem jest przypisanie odpowiednich atrybutów do znaczników.

Istotną rolę w nawigacji klawiszowej odgrywają *skip links* (*lub skimming links*). Są to dodatkowe, niewidoczne na stronie elementy *fokusyjne*, dzięki którym użytkownik może pominąć część zawartości strony internetowej, która nie jest dla niego istotna. Przykładem może być nawigacja zawierająca kilka pozycji. Pamiętając o liniowej naturze odczytywania treści przez czytniki ekranu, za każdym razem gdy

użytkownik znajdzie się na nawigacji jest zmuszony wysłuchać wszystkich pozycji po raz kolejny. Aby umożliwić ominięcie poszczególnych fragmentów strony możemy dodać *skimming link*. Link dodany przed fragmentem strony, który chcemy ominąć zostaje uaktywniony w momencie kiedy wybierzemy go za pomocą nawigacji klawiszowej. Treść zawarta w linku, która zostanie zaprezentowana użytkownikowi, powinna sugerować swoją funkcjonalność, czyli propozycję ominięcia kolejnego fragmentu strony. Po jego użyciu zostaniemy przekierowani na kolejny *fokusyjny* element za ominiętym fragmentem.

Każdy z dostępnych czytników ekranu oferuje podobną funkcjonalność i udogodnienia. Z definicji rozwiązują ten sam problem. Niestety skróty klawiszowe, sposób prezentacji głosowej czy komend głosowych różnią się. Niezmienny pozostaje sposób interpretacji dokumentów i stron internetowych. Użytkownicy czytników ekranu posiadający swoje preferencje nie będą zbyt często zmieniać oprogramowania wspomagającego i będą przyzwyczajeni do skrótów zaimplementowanych w danym programie. Podczas projektowania strony internetowej pod względem kompatybilności z czytnikami ekranu programista może zatem nie przejmować się skrótami klawiszowymi. Istotną kwestią pozostaje podążanie za standardami dostępności stron internetowych oraz dobre praktyki programistyczne.

Przestrzeganie standardu WCAG-2.0 w trakcie tworzenia nowego produktu nie jest szczególnym wyzwaniem. Problem pojawia się w momencie kiedy należy wprowadzić zmiany ułatwiające poprawną interpretację kodu starego serwisu internetowego przez czytniki ekranu. Wieloletnie technologie, które często nie są już rozwijane to spore wyzwanie dla właściciela. Bardzo trudno jest znaleźć specjalistę chętnego podjęcia się modyfikacji *legacy code'u*. O ile jest to w ogóle możliwe i opłacalne. Niestety dla firm, które posiadają tego typu produkty jako komercyjne rozwiązania, reprezentujące ich wizerunek w sieci, kompatybilność z czytnikami ekranu jest coraz częściej wymagana a nawet egzekwowana prawnie.

Programowanie po stronie klienta, wraz z rozwojem dużych framework'ów front-end'owych zyskało w ostatnich latach ogromną popularność. Coraz więcej firm produkujących rozwiązania internetowe zwraca uwagę na poprawnie zaprojektowaną warstwę prezentacji. Niesie to ze sobą wiele korzyści poprawiających odczucia towarzyszące użytkowaniu strony. Są to między innymi szybkość ładowania oraz kompatybilność z czytnikami ekranu.

Jakkolwiek dokładnie zdefiniowane są standardy, których należy przestrzegać podczas programowania stron internetowych, niestety do niedawna nie była to szeroko stosowana praktyka. O ile nowe produkty są coraz częściej zaprogramowane w sposób, który ułatwia czytnikom ekranu rozpoznanie i prezentację treści strony, o tyle starsze produkty pozostają daleko w tyle. Niestety w czasach kiedy *JavaScript* był używany jedynie do obsługi podstawowych zdarzeń, czytniki ekranu były tematem sporadycznie pojawiającym się na ustach programistów. Co za tym idzie, ich produkty do dziś widniejące dumnie w sieci to kolejne mury stawiane na drodze do nieograniczonego dostępu do sieci. Wiele firm zarządza renowację ich sieciowej reprezentacji jednak jeszcze więcej bagatelizuje problem.

Na pomoc przychodzi amerykańska ustawa prawna ADA, która zmusza coraz więcej przedsiębiorstw do zniesienia barier jakie stawiają przed użytkownikami swoich sieciowych produktów. Często taka re-aranżacja kodu źródłowego niestety okazuje się niemożliwa. Warto zatem zadać sobie pytanie, co zrobić w sytuacji kiedy chcemy poprawić kompatybilność naszej strony internetowej z czytnikami ekranu bez konieczności modyfikacji kodu źródłowego?

2. Reorganizacja struktury strony pod względem kompatybilności z czytnikami ekranu.

Kompatybilność z czytnikami ekranu to nic innego jak przestrzeganie zasad zdefiniowanych w standardzie WCAG-2.0. Jeżeli programista od początku trwania projektu ich nie przestrzegał konieczna może okazać się re-aranżacja kodu źródłowego warstwy prezentacji.

Zanim przystąpimy do jakichkolwiek prac programistycznych mających na celu poprawienie jakości naszego oprogramowania, należy przeprowadzić dokładną analizę produktu, z którym będziemy pracować.

Fundacja Widzialni (*www.widzialni.org*) [23] zajmuje się przeciwdziałaniem wykluczaniu cyfrowemu i społecznemu. Według danych znajdujących się na stronie fundacji, aż „dwadzieścia trzy procent ludności w Polsce to osoby z niepełnosprawnościami, a nie jest to jedyna grupa, która ma kłopot ze znajdowaniem i odtwarzaniem informacji na stronach. Musimy pamiętać również o osobach starszych, użytkownikach urządzeń mobilnych, nowych użytkownikach Internetu, obcokrajowcach, czy osobach korzystających ze starego oprogramowania. Patrząc szerzej mówimy, aż o osiemdziesięciu milionach spośród ludności Unii Europejskiej.” Głównym celem fundacji jest nadanie jednakowych praw i możliwości dostępowych wszystkim użytkownikom sieci bez względu na ich wiek, niepełnosprawności, oprogramowanie czy sprzęt. Fundacja od wielu lat kieruje uwagę opinii publicznej na problem dostępności stron internetowych. Badania fundacji są niezwykle precyzyjne i miarodajne ze względu na opracowanie wraz z naukowcami Uniwersytetu Śląskiego kompleksowego sposobu badania stron internetowych ze standardem WCAG-2.0.

Jak sugerują specjaliści Fundacji Widzialni, Pierwszym elementem, na który powinniśmy zwrócić uwagę jest to czy cała zawartość strony internetowej została zamieszczona w siatce zbudowanej z tabel. Jeśli tak, musimy zwrócić uwagę na kilka istotnych kwestii z tym związanych. Powinniśmy pamiętać o odpowiedniej strukturze, którą definiują tabele. Nie możemy zapominać o nagłówkach i stopkach tabel. Dzięki temu wspomożemy czytniki ekranu we właściwej interpretacji treści strony.

Kolejną istotną rzeczą jest odpowiednio reprezentacja menu. Powinna być zakotwiczona w strukturze dokumentu. Po za tym, że samo jej umiejscowienie jest istotne ze względu na liniowy charakter prezentacji stron internetowych przez czytniki ekranów, należy pamiętać o odpowiedniej strukturze znaczników HTML, z których powinna się składać. Elementy występujące po sobie o podobnym charakterze powinny składać się z tego samego, przewidywalnego zestawu znaczników. Ponadto dla każdej z podstron zawartych w naszym serwisie, powtarzalne elementy powinny zawsze znajdować się w tym samym przewidywalnym miejscu. Mówi o tym jedna z zasad standardu WCAG-2.0 [24] a mianowicie „prezentowanie powtarzających się komponentów dokładnie w tej samej kolejności i z tą samą strukturą za każdym razem kiedy się pojawiają”. Ta technika pozwala zachować spójny układ między stronami a co za tym idzie spójny sposób prezentacji za pomocą czytników ekranu. Ta zasada świetnie wpisuje się w poruszony temat nawigacji, która jest zazwyczaj powtarzalna dla każdej z podstron.

Należy również zbadać hierarchię nagłówków, tytuły oraz pola formularzy wraz z ich etykietami i pozostałymi istotnymi atrybutami. Celem utrzymania odpowiedniej hierarchii nagłówków, według standardów WCAG-2.0 [25] jest użycie odpowiednich znaczników w celu zapewnienia semantycznego kodu nagłówków. Dzięki temu czytnik ekranu będzie w stanie rozpoznać tekst jako nagłówek i w taki sposób zaprezentować go użytkownikowi. Czytniki zwykle są w stanie nawigować pomiędzy nagłówkami, dzięki czemu użytkownik może szybciej znaleźć interesujące go treści.

Jeśli mowa o atrybutach, niezwykle istotne w związku z czytnikami ekranu okazują się teksty alternatywne załączone do każdego ze zdjęć. Według techniki implementacji tekstów alternatywnych opisanej w standardach WCAG-2.0 [26], używając elementu *img* załączającego zdjęcie do naszej strony internetowej programista powinien określić tekstową alternatywę z atrybutem *alt*. Jeśli obraz zawiera tekst, który jest istotny do zrozumienia treści strony internetowej, zalecanym rozwiązaniem jest aby tekst alternatywny również zwierał ten tekst. Umożliwi to tekstowi alternatywnemu odtworzenie tej samej funkcji na stronie jaką spełnia obraz.

Teksty alternatywne nie są jedynymi atrybutami, o które należy zbadać podczas analizy kodu strony internetowej. Istotną kwestią są również atrybuty *aria*. Tą kwestię również możemy znaleźć w opisie technik aranżacji dokumentu html znajdującym się w standardzie WCAG-2.0 [27]. Punkty orientacyjne (ang. *Aria landmarks*), definiują sekcje strony. Pomagają użytkownikowi technologii wspomagających takich jakimi są czytniki ekranu w odnalezieniu odpowiedniej zawartości strony oraz w dużej mierze ułatwiają nawigację pomiędzy komponentami. Zapewniają łatwy sposób omijania nieistotnych treści, które są powielane na wielu podstronach.

Analiza jakości kodu źródłowego pod względem kompatybilności z czytnikami ekranu to nie wszystko. Kolejnym krokiem w analizie jakości strony internetowej jest weryfikacja poprawności działania nawigacji klawiszowej. Istnieje kilka sposobów nawigacji klawiszowej jednak zazwyczaj na stronach stosuje się kompozycję kilku z nich. Istotne jest aby strona internetowa zawierała linki *fokusyjne* pozwalające na ominięcie powtarzalnych fragmentów strony internetowej. Ponadto funkcjonalność powinna umożliwić przeskakiwanie pomiędzy komponentami za pomocą klawisza *tab*.

2.1 Analiza zgodności kodu źródłowego ze standardem WCAG-2.0 według WebAIM

Firma *WebAIM* opublikowała listę zasad, które powinny zostać spełnione w ramach pełnej kompatybilności z czytnikami ekranu z podziałem na ocenę przyznawaną za konkretną implementację przez organizację *W3C* [22]. Każda osoba analizująca strukturę dokumentu HTML pod względem kompatybilności ze standardem WCAG-2.0 powinna zapoznać się z listą. Może to być bardzo pomoce podczas podejmowania decyzji o wprowadzaniu zmian oraz może stanowić wysokiej jakości materiał edukacyjny.

Artykuł dzieli się na cztery sekcje, z których każda odnosi się do innej właściwości dokumentu HTML. Na potrzeby pracy magisterskiej zostaną przytoczone niektóre z rekomendacji prezentowanych w ramach każdej sekcji.

Sekcja pierwsza standaryzuje osiągalność dokumentu HTML za pomocą zmysłów. Dokument HTML powinien zawierać treści internetowe łatwo osiągalne za pomocą zmysłu wzroku, słuchu i/lub dotyku [22]. Firma WebAIM w ramach tej sekcji rekomenduje aby wszystkie widzialne obrazy, przyciski oraz mapy graficzne posiadały równoznaczny z reprezentacją wizualną tekst alternatywny. Obrazy, których treść została poniekąd przekazana w widniejącym na stronie tekście, powinny posiadać wartość atrybutu *alt* ustawioną jako *null*. Alternatywnym sposobem może być implementacja takich obrazów jako tła za pomocą kaskadowych arkuszy styli. Kolejną rekomendacją jest nadawanie wartości opisowych wszystkim polom i przyciskom formularzy znajdujących się na stronie. Jest to niezwykle istotna kwestia ponieważ formularze zazwyczaj służą do przesyłania informacji do serwera aplikacji. Serwer jest w stanie zinterpretować zawartość formularza i wykonywać na jej podstawie konkretną operację niejednokrotnie wiążącą się z transakcją finansową.

Sekcja druga definiuje rekomendacje dotyczące wykonywalnych elementów interfejsu strony internetowej. Wszystkie formularze, kontrolki, przyciski oraz nawigacja są elementami wykonalnymi [22]. Wymienione elementy są używalne za pomocą klawiatury i wykonują konkretną akcję poprzez interakcję. Według rekomendacji każda z funkcjonalności dostępnych na stronie internetowej powinna być dostępna za pomocą klawiatury. Jedynym wyjątkiem są funkcję, które nie mogą zostać wykonane za pomocą peryferyjnego urządzenia klawiszowego, takie jak np. rysowanie. Zaimplementowane na stronie internetowej skróty klawiszowe nie powinny powodować konfliktów pomiędzy skrótami natywnie działającymi w środowisku przeglądarki. Najważniejszą z zasad odnośnie nawigacji klawiszowej proponowaną przez firmę WebAIM jest możliwość nieustannego poruszania się pomiędzy komponentami za pomocą jedynie klawiatury. Oznacza to, że użytkownik nie powinien nigdy zostać zatrzymany na jednym elemencie bez możliwości opuszczenia go z użyciem klawiatury.

Sekcja trzecia odnosi się do zawartości strony. Według rekomendacji znajdujących się w tej sekcji, wszystkie treści oraz interfejs znajdujący się na stronie internetowej powinien być zrozumiały dla każdego użytkownika [23]. Oznacza to, że bezwarunkowo język strony powinien być zdefiniowany za pomocą atrybutu *lang* znacznika *html*. Jeśli wśród treści zamieszczonych na stronie pojawiają się teksty w innych językach niż tym zdefiniowanym w znaczniku *html* język powinien być określany dla elementów blokowych bądź okalających również za pomocą atrybutu *lang*. Słowa, które mogą być niejednoznaczne powinny zawierać definicję tłumaczącą znaczenie zaimplementowaną za pomocą jednej z dostępnych metod, np. w liście definicji. Skróty zawarte w tekście powinny być rozwinięte wewnątrz znacznika *abbr* bądź znajdować się w liście definicji wspomnianej w poprzednim przykładzie.

Ostatnia sekcja czwarta mówi o dostępności strony internetowej dla innych programów, którymi mogą być zarówno urządzenia klienckie jak i urządzenia wspomagające korzystanie z Internetu [22]. Oznacza to, że kod źródłowy powinien być napisany zgodnie z zasadami semantyki HTML i wspomagać programy interpretujące.

Po dogłębnej analizie produktu wykonanej przez doświadczonego programistę pod kątem wszystkich wymienionych rekomendacji, należy zastanowić się jak dużo pracy wymaga dostosowanie istniejącej strony internetowej pod względem kompatybilności z popularnymi czytnikami ekranu oraz standardem WCAG-2.0. Zanim zostanie podjęta decyzja o rozpoczęciu pracy, należy poszukać jak najmniej inwazyjnych metod re-aranżacji, o ile takie istnieją i jest to w ogóle możliwe.

2.1. Istniejące rozwiązania

Podczas badań przeprowadzonych w ramach pracy magisterskiej znalezione zostały dziesiątki rozwiązań związanych z testowaniem aplikacji internetowych pod względem kompatybilności z czytnikami ekranu. Należą do nich między innymi popularne programy analizujące statycznie strukturę kodu HTML z uwzględnieniem implementacji technik opisanych w ramach standardu WCAG-2.0. Niestety nie znaleziono żadnych rozwiązań oferujących automatyczną bądź częściowo automatyczną re-aranżację kodu źródłowego pod względem kompatybilności z czytnikami ekranu. Projekt przygotowany w ramach pracy magisterskiej wydaje się być pionierskim rozwiązaniem nie pojawiającym się do tej pory wśród oferowanych rozwiązań.

2.1.1 Dedykowane rozwiązania

Dedykowane rozwiązania problemu z brakiem kompatybilności strony internetowej ze standardami WCAG-2.0 mogą być zaimplementowane na wiele sposobów. Niestety na rynku nie są dostępne rozwiązania ogólne pozwalające poprawić jakość kodu bez konieczności jego modyfikacji.

Firmy zmagające się z „*doprogramowaniem*” standardów WCAG-2.0 sięgają po różne metody, które są przystosowane do ich konkretnego produktu oraz specyfiki firmy. W ramach pracy magisterskiej zostanie omówiona jedna implementacja dedykowanego rozwiązania braku wparcia dla czytników ekranu w kodzie.

Ze względu na poufność danych osobowych nazwy firm zostały zmienione. **Firma X** jest firmą borykającą się z dylematem wprowadzenia uaktualnienia dla poprawy wsparcia czytników ekranu bądź zapłacenia wysokiej kary grzywny i utraty reputacji. Analitycy zatrudnieni przez *Firmę X* przeprowadzają dogłębną analizę kodu źródłowego strony za pomocą narzędzi przeznaczonych do tego celu oraz manualnie.

Po zapoznaniu się z problemem, okazuje się, że należy wprowadzić kilkadziesiąt modyfikacji do istniejącego markup’u oraz dodać kilka skryptów *JavaScript* poprawiających nawigację klawiszową. Do zalecanych modyfikacji należało dodanie odpowiednich tekstów alternatywnych, podział dokumentu na odpowiednie komponenty oraz nadanie odpowiednich ról *aria* ich kontenerom. Dodanie *skimming linków* przed elementami powtarzającymi się na podstronach. Ze względu na automatycznie generowane adresy *url* zalecona została również modyfikacja *metatagów* każdej z podstron; tytuł, opis oraz autor/właściciel. Strona prezentowała również bardzo wiele treści, które nie były bezpośrednio związane z zawartością strony przez co niezbędne było wprowadzenie odpowiednich ścieżek nawigacyjnych omijających nieistotne treści automatycznie. Wszystkie te modyfikacje nie byłyby trudne do wprowadzenia.

Niestety w przypadku *Firmy X* wprowadzanie jakichkolwiek zmian do źródła strony było bardzo ryzykowne z biznesowego punktu widzenia. *Firma X* posiadała kilka tysięcy bardzo podobnych produktów, które były prezentowane na oddzielnych stronach internetowych pod oddzielnymi adresami *url*. Wszystkie te strony korzystały z tego samego dedykowanego serwera generującego kod źródłowy strony. Wprowadzenie modyfikacji w głównym szablonie współdzielonym pomiędzy wszystkimi stronami produktów mogło skutkować ogromnymi stratami finansowymi w przypadku pomyłki czy jakiejkolwiek niezgodności. Tak duża skala modyfikacji zawsze niesie ze sobą ryzyko niepowodzenia. *Firma X* oraz jej analitycy starali się znaleźć odpowiednie rozwiązanie problemu na marne.

Z pomocą *Firmie X* przyszła *Firma Y* oferująca dynamiczne testy A/B. **Testy A/B** według *ORACLE* [28], firmy posiadającej jedną z najlepszych na rynku usługę testowania stron internetowych w ramach *OMC* (*Oracle Marketing Cloud*) to jeden z najprostszych sposobów na zwiększenie konwersji oraz zdobycie wiedzy o swoich odbiorcach. Testy A/B według *ORACLE* wciąż nie są wykorzystywane w odpowiednim stopniu przez specjalistów od marketingu internetowego. Jest tak dlatego, że każda forma testowania jest uważana za bardzo techniczną i trudną do wdrożenia. Ale tak nie jest. Biorąc pod uwagę korzyści płynące z takiego sposobu testowania witryny, nie można go zignorować.

Firma X wykorzystała usługi *Firmy Y* w dość niekonwencjonalny sposób. *Firmia Y* zaproponowała napisanie skryptu, który zostanie wstrzyknięty tuż przed wyświetleniem zawartości strony i wprowadzi odpowiednie modyfikacje kodu źródłowego bez konieczności modyfikacji istniejącego rozwiązania. Skrypt zostanie zaimplementowany jako jeden z wariantów testu A/B, który zostanie zaprezentowany dla stu procent odwiedzających.

2.1.2 Wady i zalety dedykowanych rozwiązań.

Największą zaletą dedykowanego rozwiązania opisanego wcześniej była jego skuteczność. Rozwiązanie działało i w sprzyjających warunkach spełniało doskonale swoje zadanie.

Niestety tego typu modyfikacje wykonywane po załadowaniu całej zawartości strony poprzez skrypt wstrzyknięty z zewnętrznego serwera i uruchamiany podczas procedury *document.write*. Nie

trudno domyślić się jak wiele czasu zajmuje wykonanie wszystkich tych kroków a przy wolnym połączeniu internetowym może przynieść więcej szkód niż pożytku. Wadą takiego rozwiązania jest również uzależnienie od obecnej struktury dokumentu. Jeśli cokolwiek zmieni się w zawartości kodu źródłowego strony, nawet drobna rzecz taka jak nazwa klasy czy atrybutu, konieczne jest zlecenie dostosowania testu a co za tym idzie poniesienie podwójnych kosztów modyfikacji. Przez ten zabieg *Firma X* uzależnia się od usług *Firmy Y*, która pobiera stałe opłaty za nieustanne serwowanie testu oraz za każdą niezbędną modyfikację, która pojawia się w trakcie typowego cyklu życia aplikacji internetowej.

3. Możliwe rozwiązania problemu reorganizacji kodu źródłowego

Jakkolwiek niekonwencjonalna i wadliwa okazała się propozycja rozwiązania problemu opisana w poprzednim rozdziale, stała się ona inspiracją do zgłębienia tematu dostępności stron internetowych i zaprogramowania generycznego rozwiązania. Efektem wielomiesięcznych badań i testów przeprowadzanych na kilkunastu stronach internetowych jest uniwersalna biblioteka *JavaScript,* zawierająca zestaw niezbędnych metod oraz rozszerzenie narzędzi deweloperskich przeglądarki *Google Chrome*, dzięki któremu reorganizacja kodu dla polepszenia jakości wsparcia czytników ekranu stała się dużo łatwiejsza.

Doświadczenie nabyte podczas badań nad dostępnością sieci umożliwiło wyodrębnienie najczęstszych błędów w implementacji standardu WCAG-2.0 oraz zasad najczęściej pomijanych przez programistów. Szczegóły dotyczące wspomnianych błędów, zostaną przedstawione w dalszej części pracy.

W rozbudowanej dziedzinie jaką jest inżynieria oprogramowania zostało zdefiniowanych wiele zasad i praktyk uznawanych jako właściwe. Jednymi z nich są zasady *SOLID*. **SOLID** [30] w programowaniu obiektowym jest skrótem dla pięciu zasad projektowania oprogramowania mających na celu sprawienie by kod był bardziej zrozumiały, elastyczny i łatwy w utrzymaniu. Zasady *SOLID* są podzbiorem ogólnych zasad promowanych przez Roberta Cecila Martina (Uncle Bob) w jego książce „*Czysty Kod*" [58] i „*Agile. Programowanie zwinne: zasady, wzorce I praktyki zwinnego wytwarzania oprogramowania w C#*" [59]. Chociaż zasady *SOLID* mają głównie zastosowanie w przypadku projektowania obiektowo zorientowanego kodu, mogą mieć również podłoże filozoficzne dla metodologii wytwarzania oprogramowania takich jak np. *Agile*. Pięć zasad wchodzących w skład paradygmatu *SOLID* to:

- *Single responsibility principle* [31] jest zasadą mówiącą, że klasa powinna mieć jeden i tylko jeden powód aby być zmieniona. Oznacza to, że klasa powinna być odpowiedzialna tylko za jedno zadanie.
- *Open-closed principle* [31] jest zasadą mówiącą, że obiekty bądź encje w kontekście programowania obiektowego, powinny być zamknięte na modyfikacje i otwarte na rozszerzanie. Oznacza to, że kod powinien być łatwy do rozszerzenia bez konieczności modyfikacji samego kodu.
- *Liskov substitution principle* [31] jest zasadą mówiącą, że funkcje używające wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych bez znajomości tych obiektów. Zasada została pierwszy raz zdefiniowana przez *Barbarę Liskov* w książce *Data Abstraction and Hierarchy*. Zasada podstawienia Liskova tak jak wszystkie pozostałe zasady wchodzące w skład SOLID, została spopularyzowana przez *Roberta C. Martina* w jego publikacji *„Principles of object-oriented design"*.
- *Interface segregation principle* [31] jest zasadą definiującą sposób segregacji interfejsów oraz to, że *klient* nigdy nie powinien implementować interfejsu, którego nie używa. W żadnym wypadku nie należy uzależniać klientów od metod, których nie używają.
- *Dependency inversion principle* [31] jest zasadą mówiącą o zależnościach pomiędzy encjami. Encje powinny zależeć od abstrakcji a nie od konkretnych obiektów. Według tej zasady żaden moduł wysokiego poziomu nie powinien zależeć od modułu niskiego poziomu, natomiast bazować na abstrakcjach.

Przestrzeganie zasad *SOLID* jest kluczowe w procesie wytwarzania wysokiej jakości oprogramowania. Druga zasada *SOLID, „open-close principle”* definiuje sposób projektowania oprogramowania tak aby było ono „zamknięte” na modyfikacje ale „otwarte” na rozszerzanie. *Firma X* chciała wprowadzić wsparcie dla czytników ekranu bez konieczności bezpośredniej modyfikacji ich istniejącego szablonu, a zatem rozszerzyć funkcjonalność szablonu bez jego modyfikacji. Dzięki elastyczności i możliwościom języka *JavaScript,* wszystkie niedociągnięcia związane ze wsparciem czytników ekranu mogą zostać zaimplementowane w taki sposób.

Takie rozwiązanie może być interesującą alternatywą dla natywnych modyfikacji kodu, w przypadkach kiedy te nie są bezpieczne.

3.1. Reorganizacja - cele i założenia

Przed rozpoczęciem analizy korzyści i problemów płynących z tak niekonwencjonalnego podejścia do rozwiązania problemu kompatybilności z czytnikami ekranu, należy zastanowić się nad jej długofalowym celem i założeniami, które powinna spełniać.

W trakcie pracy zawodowej dla amerykańskiej korporacji, w związku z rosnącym zainteresowaniem tamtejszych władz dostępnością sieci ze względu na ustawę *ADA*, wielokrotnie zlecano pracę nad polepszeniem bądź dodaniem wsparcia czytników ekranu w produktach klienckich. Takie prace za każdym razem stanowiły ogromne wyzwanie i wymagały dogłębnej analizy kodu źródłowego aplikacji klienckich. Doświadczenia nabyte w trakcie wykonywania tego typu zleceń, umożliwiły zbudowanie listy najczęściej popełnianych błędów oraz pomijanych technik wsparcia standardu WCAG-2.0, która może posłużyć jako lista celów i założeń reorganizacji struktury kodu źródłowego strony w celu poprawienia jakości wsparcia czytników ekranu.

Pierwszą pozycją na liście jest brak, bądź niewłaściwie zdefiniowane teksty alternatywne. Według standardu WCAG-2.0 [26], każde zdjęcie zamieszczone na stronie internetowej za pomocą tagu *img*, powinno mieć przypisany atrybut *alt* z wartością stanowiącą tekst alternatywny. Wielokrotnie dodawanie atrybutu jest bagatelizowane przez programistów lub są dodawane niedbale z treściami nie spełniającymi roli, do której zostały przypisane.

Atrybut *title*, określający dodatkowe informacje na temat elementu okalającego był rzadkością w kodzie źródłowym badanych aplikacji. Ten atrybut był regularnie pomijany przez programistów programujących strony internetowe. Ze względu na różnorodność zastosowań, zostało zdefiniowanych kilka technik WCAG-2.0 odnoszących się do atrybutu *title*. Na potrzeby pracy magisterskiej zostaną przytoczone najistotniejsze z nich:

- Technika *H64* [32], definiuje użycie atrybutu w celu opisania zawartości każdego tagu *frame* bądź *iframe*. Spełnia on rolę etykiety ramki, dzięki czemu użytkownicy mogą określić zawartość ramki i zadecydować o dalszej interakcji z elementem. Atrybut *title* nie jest wymienny z atrybutem *name*.
- Technika *H65* [33] mówi o używaniu atrybutu *title* dla kontrolek formularzy w celu identyfikacji kontrolki w przypadku kiedy nie można użyć etykiety. W przypadku braku etykiety, czytniki ekranu zaprezentują kontrolkę za pomocą wartości podanej w atrybucie.
- Technika *H89* [34] definiuje sposób użycia atrybutu w celu zapewnienia pomocy kontekstowej podczas wprowadzania danych do formularzy. Może wówczas sugerować wymagany format danych lub prezentować przykładową wartość.

- Technika *H67* [35] mówi o oznaczaniu załączonych do strony obrazów aby zostały zignorowane przez czytniki ekranu. Jeśli atrybut *title* nie zostanie w ogóle użyty a wartość atrybutu *alt* zostanie ustawiona jako *null*, czytnik ekranu pominie tak oznaczone zdjęcie.

Kolejnym częstym błędem występującym w istniejących implementacjach kodu źródłowego stron internetowych jest brak lub niewłaściwe użycie atrybutu *lang*. Zostały wyspecyfikowane dwie techniki używania atrybutu *lang* należące do standardu WCAG-2.0:

- Technika *H57* [36] definiuje sposób użycia atrybutu *lang* na głównym tagu *html*. Zadaniem tego sposobu użycia jest identyfikacja domyślnego języka dokumentu. Identyfikacja języka jest niezwykle ważna z kilku powodów. Pozwala to oprogramowaniu wysyłającemu sygnały do urządzenia brajlowskiego na wybranie odpowiedniego tłumaczenia. Również syntezatory mowy implementowane w popularnych czytnikach ekranu korzystają z atrybutu orientacji języka i dzięki niemu dostosowują wymowę. Oznaczanie języka za pomocą *lang* może przynieść korzyści w przyszłości, gdy urządzenia tłumaczące będą tłumaczyły strony internetowe w czasie rzeczywistym.
- Technika *H58* [37]. Celem tej techniki jest identyfikacja każdej zmiany języka wewnątrz zawartości strony.

Niejednokrotnie błędną implementację standardu WCAG-2.0, bądź jej brak, możemy znaleźć wśród metadanych stron internetowych. Jak podaje definicja *W3Schools*, **HTML Meta Tag** [38], inaczej *metadane*, są informacjami na temat zawartości strony internetowej i jej kodu. Znacznik *meta* definiowany wewnątrz znacznika *head* zawiera zestaw informacji, który nie będzie wyświetlany na stronie a może posłużyć programom interpretującym, takim jak czytniki ekranu, przeglądarki internetowe czy wyszukiwarki słów kluczowych. Elementy *meta* są używane do określania opisu strony (*description*), słów kluczowych (*keywords*) czy autora dokumentu (*author*). Zostało zdefiniowanych kilka technik implementacji metadanych i tagu *title* jako standard WCAG-2.0. Należą do nich:

- Technika *H58* [39], opisująca definiowanie tytułu strony internetowej za pomocą tagu *title* dodanego do tagu *head* dokumentu HTML. Tytuł powinien w prostym zdaniu zawierać główną tematykę strony internetowej. Po załadowaniu dokumentu, czytnik ekranu przeczyta zawartość tagu jako tytuł strony.
- Pozostałe techniki implementacji metadanych w sposób spełniający standardy WCAG-2.0 został opisany w dodatku do dokumentacji W3C, *Understanding Metadata* [40].

Kolejnym elementem na liście błędów popełnianych przez programistów w implementacji wsparcia czytników ekranu jest całkowity implementacja punktów orientacyjnych strony internetowej. *Aria Landmarks*, i ich rola we wsparciu programów interpretujących, zostały opisane we wcześniejszych rozdziałach pracy.

Ostatnią na liście najczęstszych przyczyn problemów z czytnikami ekranu jest nawigacja klawiszowa. Możliwość przemieszczania się pomiędzy komponentami strony internetowej za pomocą klawiatury jest niezbędną funkcjonalnością nie tylko w kontekście standardu WCAG-2.0 i czytników ekranu. Poprawnie zaimplementowana nawigacja klawiszowa w znacznej mierze ułatwia korzystanie z usług oferowanych przez aplikację. Istnieją sytuacje, w których użytkownik nie ma możliwości użycia innego urządzenia nawigacyjnego niż klawiatura. Wówczas niewłaściwie działająca lub nie w pełni funkcjonalna nawigacja klawiszowa może okazać się sporym problemem. Oprócz frustracji wywołanej u użytkownika, która może skutkować natychmiastową reakcją w sieci, w postaci negatywnych opinii na temat wirtualnej reprezentacji naszej marki, możemy napotkać na dużo poważniejszy z biznesowego punku widzenia problem. Jest nim zmniejszenie konwersji i zysków generowanych przez stronę.

Wyżej wymienione elementy listy najczęściej popełnianych błędów i zaniedbań, zostały zdefiniowane w trakcie wielomiesięcznej pracy i badań nad dostępnością stron internetowych. Powyższa lista posłużyła jako lista celów i założeń, które w innowacyjny i zarazem przystępny sposób zostaną spełnione w ramach projektu pracy magisterskiej.

3.1.1 Korzyści płynące z budowania aplikacji zgodnych ze standardem WCAG-2.0

Aby zrozumieć wszystkie korzyści płynące z uwzględniania standardów projektowania aplikacji internetowych, zdefiniowanych przez organizację *W3C* jako standard *WCAG-2.0*, warto odpowiedzieć sobie na pytanie, czym są i do czego służą standardy? Jak podaje słownik [41], standard jest określeniem zbioru zasad uważanych zgodnie za właściwe oraz uznawanych jako podstawa porównania – innymi słowy, zatwierdzony model.

Fundacja Widzialni w bardzo przystępny sposób opisuje standard WCAG-2.0, korzyści płynące z jego stosowania oraz konsekwencje jego ignorowania [42]. Od wynalezienia Internetu do momentu kiedy jego zasięg stał się globalny minęło zaledwie kilka lat. Rozwijające się w zatrważającym tempie technologie serwerowe, są w stanie pomieścić coraz więcej danych, które są udostępniane całemu światu za pośrednictwem sieci. Nie trudno się domyślić, że przeciętnemu użytkownikowi odnalezienie się pośród ogromu informacji, którym jest zalewany, może sprawiać spore problemy. Biorąc pod uwagę ilość nowych programów klienckich jakimi są przeglądarki internetowe, interpretery stron internetowych, wyszukiwarki i wszystkie łatki i uaktualnienia dla każdego istniejącego programu, bardzo trudno wyobrazić sobie jak to wszystko nadal ze sobą współpracuje. Ilość oprogramowania, które staje się coraz bardziej niezbędne nigdy się nie zmniejszy a będzie sukcesywnie wzrastać. Właśnie dlatego standaryzacja procesów wytwarzania oprogramowania jest konieczna dla utrzymania kompatybilności pomiędzy aplikacjami.

Stosowanie standardów może być błędnie odbierane jako dostosowywanie się do pewnych ograniczeń definiowanych przez globalne organizacje. Istotą implementacji standardów W3C w aplikacjach internetowych jest zmaksymalizowanie liczby osób, które będą mogły korzystać z naszego produktu. Trudno wyobrazić sobie sytuację, w której producenci sprzętu komputerowego nie stosują się do ustandaryzowanego sposobu podłączania, instalacji i użycia urządzeń elektronicznych. Gdyby nie standaryzacja, każde urządzenie peryferyjne miałoby inną wtyczkę, kabel a komputer inne gniazdo a co za tym idzie używanie dwóch urządzeń razem byłoby niemożliwe. Podobna sytuacja mogłaby pojawić się w świecie oprogramowania komputerowego i aplikacji internetowych, gdyby firmy produkujące oprogramowanie nie stosowały się do standardu przesyłania danych jakim jest protokół *http*. Dzięki temu, że ten protokół został zdefiniowany i przyjęty jako standard, możemy codziennie korzystać ze stron internetowych.

Strony internetowe stanowią warstwę prezentacji dla skomplikowanych systemów bankowych, programów analitycznych i każdego innego serwisu internetowego. Niestety osoby tworzące aplikacje internetowe nie zawsze podążają za wyznaczonymi standardami przez co budują kolejne bariery w sieci. Właśnie z tego powodu organizacja W3C, zrzeszająca największe firmy i czołowe organizacje świata utworzyła inicjatywę *WAI – Web Accessibility Initiative*, określająca standard budowania powszechnie dostępnych stron internetowych. W skład standardów zdefiniowanych przez WAI wchodzą między innymi *ARIA* oraz *WCAG-2.0*.

Dokument WCAG w swojej wczesnej wersji został podzielony na trzy główne sekcje nazywane priorytetami. Jako pierwszy priorytet („A”) zapisane zostały wszystkie zasady, których użytkownik musi bezwzględnie przestrzegać aby strona internetowa mogła być uznana za dostępną w stopniu podstawowym. Zasady zdefiniowane w drugim priorytecie („AA”) gwarantowały ponad podstawowy

stopień dostępności a rekomendacje w nim zawarte powinny być przestrzegane. W trzecim priorytecie („AAA”) zawarte zostały rekomendacje, których przestrzeganie zapewnia najwyższy poziom dostępności i mogą być przestrzegane przez programistę.

Przestrzeganie wszystkich zasad zapewnia niczym nieograniczony dostęp do usług oferowanych przez aplikację internetową. Dodatkowo umożliwia rozwój produktu poprzez programowanie rozszerzeń oraz dzięki swojej przewidywalności używanie aplikacji za pomocą programów wspomagających. Gdyby wszystkie strony internetowe były budowane zgodnie ze standardami opisanymi przez organizację W3C, rozwój Internetu byłby dużo łatwiejszy i z pewnością dużo bardziej dynamiczny niż w chwili obecnej.

3.2. HTML5 a czytniki ekranu

Jak podaje definicja na stronie *MDN (Mozilla Developer Network)* [43], **HTML5** jest najnowszą wersją standardu opisującego znacznikowy język HTML. Można zdefiniować termin na dwa sposoby:

- HTML5 jest najnowszą wersją języka wprowadzająca nowe atrybuty i zachowania oraz elementy.
- Jest to duży zestaw nowych technologii pozwalających na różnorodne i potężne tworzenie stron internetowych. Zestaw ten potocznie nazywany jest *HTML5 and friends* często skracany do nazwy *HTML5*.

Główną różnicą w kwestii wsparcia czytników ekranu za pomocą atrybutów *aria* a natywnymi tagami html jest to, że *aria* jest rozszerzeniem, atrybutem, podczas gdy tag *HTML5* jest w pełni semantycznym elementem kodu html.

HTML5 wprowadza nowe elementy semantyczne reprezentujących sekcje i komponenty aplikacji internetowych. Żaden z nowych elementów nie oferuje nowych funkcjonalności, zapewnia jednak autorom stron internetowych o wiele łatwiejszy i przyjemniejszy sposób definiowania części dokumentu. Dodatkowo daje nowe możliwości oprogramowaniu wspomagającemu, które dzięki nim może w lepszy sposób interpretować zawartość kodu źródłowego strony i wykorzystywać te informacje dla lepszej nawigacji i prezentacji strony.

Język HTML został zaprojektowany tak, aby zawsze był kompatybilny z przeglądarkami. Jeśli jakiś fragment czy znacznik kodu HTML nie jest rozpoznawany przez daną przeglądarkę, zostanie pominięty, jednak jego zawartość zostanie wyświetlona. Ponieważ żaden z nowo dodanych semantycznych znaczników HTML5 nie posiada wbudowanych funkcjonalności i nie wymaga specjalnego formatowania, powinien być kompatybilny ze wszystkimi starymi i nowymi wersjami przeglądarek internetowych.

Niestety nie wszystkie czytniki ekranów są w stanie rozpoznawać tagi HTML5 i nadal rekomendowane jest używanie atrybutów *aria* do określania sekcji dokumentu.

3.3. Przegląd możliwości koncepcyjnego rozwiązania

W poprzednich rozdziałach została zdefiniowana lista najczęściej popełnianych błędów i zaniedbań programistów w kwestii wsparcia czytników ekranu. Poruszone zostały również kwestie problemów z implementacją poprawek kodu źródłowego oraz niebezpieczeństwa jakie ze sobą niosą.

Niemniej jednak budowanie produktów dostępnych dla jak największej ilości osób w sieci jest coraz częściej egzekwowane z tytułów prawnych.

Koncepcyjne rozwiązanie powinno zatem nie tylko spełniać założenia postawione w poprzednich rozdziałach pracy magisterskiej, ale również być łatwe w implementacji i elastyczne. Lista problemów została zbudowana na podstawie subiektywnych doświadczeń, zatem koncepcja rozwiązania problemu powinna również zakładać łatwą możliwość rozszerzania funkcjonalności na potrzeby specyficznych przypadków. Powinno również posiadać wszystkie zalety oraz eliminować każdą wadę opisaną w poniższych podrozdziałach.

Rozwiązanie zaproponowane przez *Firmę X*, które polegało na wykorzystaniu platformy chmurowej firmy zewnętrznej, która oferowała wówczas usługi testowe, posiadało wiele wad. Nie można jednak odmówić skuteczności tego rozwiązania. Pytanie nasuwające się w trakcie rozważań nad decyzją *Firmy Y* o wprowadzeniu testu A/B jako rozwiązania ich problemu z kompatybilnością z czytnikami ekranu to, czy można rozwiązać ten problem w inny, może lepszy sposób?

Należy przeanalizować wszystkie wady płynące z implementacji wykorzystanej w *Firmie Y* oraz wszystkie zalety. Wspomniany skrypt wstrzykiwany i uruchamiany w trakcie procedury document.write był programem napisanym z użyciem języka *JavaScript*. Używanie tego języka dla rozwiązywania problemu interpretacji strony internetowej przez przeglądarkę, niesie ze sobą kilka istotnych korzyści.

3.3.1 Elastyczność języka JavaScript

Przede wszystkim *JavaScript* był zaprojektowany aby być uruchamianym w środowisku przeglądarki zawierającej odpowiednie interpretery. Z ogromnym prawdopodobieństwem zatem, skrypt zostanie poprawnie uruchomiony w każdej z istniejących przeglądarek. Kolejną zaletą jest elastyczność oferowana przez język.

Każda funkcja *JavaScript* dostępna z poziomu przeglądarki może być dowolnie modyfikowana z zewnątrz. Jeśli programista jest w pełni świadomy konsekwencji swoich działań, może okazać się to bardzo pomocne, natomiast w przypadkach nieświadomej modyfikacji niesie ze sobą wiele niebezpieczeństw.

3.3.2 DOM API i możliwości modyfikacji kodu HTML

Zarówno *JavaScript* jak i *HTML* są nieodzownymi elementami każdej aplikacji internetowej. Rozbudowana, wirtualna reprezentacja kodu źródłowego *HTML*, udostępniona dla języka *JavaScript* jaką jest *DOM API*, daje nieograniczone możliwości modyfikacji i rozszerzania kodu źródłowego strony w dowolnym etapie trwania aplikacji.

Wprowadzanie zmian kodu HTML po załadowaniu gotowego dokumentu, daje nam możliwość dostosowania źródła aplikacji do dowolnych standardów wymaganych w związku z czytnikami ekranu.

Rozsądnym jest zatem wybór technologii internetowych do implementacji rozwiązania, zaprezentowanego w ramach projektu pracy magisterskiej. Dzięki temu otrzymamy możliwość skorzystania ze wszystkich zalet oferowanych przez wielofunkcyjne i elastyczne języki jakimi są *JavaSctipt* i *HTML*.

3.3.3 Niezależność projektu

Największą wadą zaimplementowanego rozwiązania było uzależnienie produktu od usług firmy zewnętrznej i jej infrastruktury. Takie działanie powoduje, że w momencie gdy serwery firmy zewnętrznej zawodzą, nasza strona internetowa pomimo w pełni sprawnej infrastruktury nie działa poprawnie. Co gorsza, może przestać działać jeśli zostanie wykonana tylko część skryptów

serwowanych przez zewnętrzne usługi. Dzięki rozwiązaniom zaimplementowanym w projekcie pracy magisterskiej, możemy wprowadzać wszystkie wymagane modyfikacje pozostając niezależnymi.

Wadą implementacji użytej przez *Firmę X*, jest ograniczona możliwość wprowadzania modyfikacji strony internetowej oraz podwójne koszty z nimi związane. Niejednokrotnie w celu poprawienia jakości działania strony internetowej konieczne jest wprowadzenie małych poprawek w szablonie. Niestety w przypadku serwowania testu A/B nie są one możliwe do wprowadzenia bez konieczności dostosowania kodu skryptu testującego.

Istotną wadą używania testów A/B w tak niekonwencjonalny sposób jak ten opisany w poprzednich rozdziałach pracy magisterskiej, jest jej niska skuteczność a nawet szkodliwość w przypadkach użycia przez urządzenia o słabej jakości połączeniu internetowym. W takiej sytuacji może zostać wstrzyknięta przez zewnętrzne serwery jedynie część testu, która zaburzy działanie strony bądź zupełnie je uniemożliwi. Oprogramowanie budowane w ten sposób samo w sobie nie jest zgodne ze standardami wyznaczonymi przez W3C w standardzie WCAG-2.0 przez to, że ogranicza dostęp do strony internetowej ze względu na możliwości sprzętowe.

Propozycja *Firmy Y* rozwiązywała szczególny przypadek braku wsparcia dla czytników ekranu *Firmy X*. Niestety w przypadku kolejnych zmian koniecznych do wprowadzenia w związku z tym problemem, skrypt nie spełniał by już swojej roli. Za każdym razem wymagałby modyfikacji i dostosowania co wymaga zaangażowania osób trzecich w rozwój produktu.

Koncepcyjne rozwiązanie zaproponowane w projekcie pracy magisterskiej, powinno rozwiązywać każdą z wad w odpowiedni sposób. Musi być zatem niezależnym fragmentem oprogramowania, rozszerzającym funkcjonalność kodu HTML strony internetowej o wsparcie dla czytników ekranu.

4. Przegląd technologii wykorzystanych w projekcie

Zgodnie z założeniami zaprezentowanymi w poprzednich rozdziałach pracy, projekt został zrealizowany z wykorzystaniem technologii front-end'owych jakimi są *JavaScript, HTML* i *CSS*.

Dla wykorzystania wszystkich korzyści oferowanych przez wspomniane technologie, w ramach projektu zostaną wykorzystane najnowocześniejsze narzędzia i techniki programowania w wybranych językach. Należą do nich *Webpack*, który jest narzędziem służącym do budowania końcowych paczek *JavaScript, HTML* i *CSS*, oferujący *transpilację* popularnych preprocesorów oraz rozszerzeń języka *JavaSctipt*.

Transpilacja [61], inaczej kompilacja źródła do źródła. To proces odczytywania kodu źródłowego napisanego w jednym języku programowania i jednoczesnej produkcji równoważnego kodu innego języka. Terminy kompilacja i transpilacja są często używane zamiennie, jednak ich dokładne znaczenie jest zupełnie inne. Języki programowania przenoszone do *JavaScript'u*, są często nazywane językami transpilowanymi, których językiem wyjściowym jest *JavaScript*.

Dodatkowo w trakcie budowania paczki końcowej aplikacji za pomocą tego narzędzia, możemy w łatwy sposób zainicjować procedurę popularnie nazywaną *Tree shaking*.

Tree shaking (ang. Otrząsanie drzewa) [62], to termin powszechnie używany w kontekście języka *JavaScript*, oznaczający w pełni zautomatyzowane usuwanie *martwego*, nieużywanego kodu aplikacji poprzez statyczną analizę referencji. Opiera się na statycznej składni modułów standardu *ES2015*, tj. *import* i *export*.

Kolejną technologią wykorzystaną w projekcie jest *NPM* (*Node package manager*). Całość została zaprogramowana z użyciem języka *TypeScript* z wykorzystaniem statycznych analizatorów kodu (*Linters*). Bezpieczeństwo kodu źródłowego biblioteki, zostało zapewnione dzięki testom jednostkowym napisanym z użyciem technologii *Karma* i *Jasmime*.

Przed szerszym omówieniem technologii wykorzystanych przy budowie programu stanowiącego projekt pracy magisterskiej, warto zastanowić się czym jest „nowoczesne budowanie aplikacji front-end'owych". Uproszczając, jest to wykorzystywanie w procesie budowania aplikacji klienckich, nowoczesnych narzędzi automatyzujących budowanie aplikacji oraz korzystanie z nowoczesnego stosu technologicznego. Praca programisty warstwy prezentacji aplikacji internetowej, nie polega już tylko na przelaniu projektu na kod HTML i CSS oraz dodaniu kilku funkcji obsługujących zdarzenia kliknięcia. Dziś znaczy to o wiele więcej.

Jak wspomina *Artem Sapegin*, twórca *React Styleguidist*, we wpisie na blogu technologicznym *Medium* [45], rok 2018 jest momentem, w którym rola programisty front-end powinna zostać zdefiniowana na nowo. Ze względu na dynamiczny rozwój technologii związanych z językiem *JavaScript*, programiści poświęcają bardzo dużą część swojego dnia na poszukiwaniach nowych rozwiązań, eksperymentach i dyskusjach na ich temat w sieci. Jest to jak najbardziej właściwe podejście ale pod warunkiem, że zdążyliśmy się zaprzyjaźnić z technologiami i paradygmatami, które nie zmieniają się tak szybko. Mowa tutaj o podstawach programowania obiektowego, wzorcach projektowych oraz zrozumieniu zasad działania narzędzi związanych z językiem. Ich znajomość nie tylko znacznie poprawi jakość wykonywanych przez programistów usług ale również pomoże szybciej przyswajać nowinki technologiczne pojawiające się na rynku.

Po zapoznaniu się z podstawami programowania, niezbędnymi dla zrozumienia sposobu budowy i działania aplikacji internetowych, możemy zastanowić się nad wyborem technologii, odpowiednich

dla rozwiązania problemu braku adaptacji standardu WCAG-2.0, z uwzględnieniem wszystkich założeń postawionych w poprzednich rozdziałach.

Wszystkie technologie wykorzystane w projekcie były świadomym wyborem popartym doświadczeniem zawodowym autora. Poniżej znajduje się krótki opis najistotniejszych z nich.

4.1. Języki programowania

W ramach projektu pracy magisterskiej, wykorzystane zostały podstawowe technologie służące do programowania złożonych i wydajnych aplikacji internetowych. Są to języki programowania zaprojektowane z myślą o budowie aplikacji uruchamianych w środowisku przeglądarki, mianowicie *JavaScript, HTML, CSS*. Przeznaczenie i możliwości oferowane przez wspomniane języki, idealnie wpisują się w wymagania projektu pracy.

4.1.1 JavaScript

JavaScript [63] jest skryptowym językiem programowania, pozwalającym na implementację skomplikowanych operacji, poprzez skrypty uruchamiane przez interpretery zawarte w przeglądarkach internetowych. Każda operacja wykonywana na stronie internetowej, oferująca więcej niż tylko statyczne wyświetlanie zawartości, zapewne angażuje skrypty tego języka.

JavaScript jest wieloplatformowym, obiektowo zorientowanym językiem, który cechuje szybkość działania i lekkość jego skryptów. W środowisku, w którym jest uruchamiany, może w łatwy sposób zostać połączony z dostępnymi obiektami, w celu umożliwienia łatwej kontroli nad nimi.

Podstawowa biblioteka języka [64], zawiera struktury danych, instrukcje, obiekty i operatory. Standardowa biblioteka w łatwy sposób może zostać rozszerzona i dostosowana do potrzeb programisty.

Podstawowa implementacja języka, uruchamiana po stronie przeglądarki, została rozszerzona o obiekty udostępniające nowe funkcjonalności, takie jak zarządzanie zasobami przeglądarki jakimi są m.in. *local storage, indexedDb*, wirtualny model *DOM* oraz wiele innych.

Podstawowa implementacja języka, uruchamiana po stronie serwera, została wzbogacona m.in. o mechanizmy zapewniające komunikację i połączenie z bazą danych i manipulację plikami systemowymi.

Język *JavaScript* i *Java* często są mylnie rozpoznawane jako pochodne. Co prawda zawierają pewne podobieństwa, natomiast dzielą je fundamentalne różnice. Początkujący programista może zauważyć podobieństwa czytając kod źródłowy programów napisanych w tych językach, jednak *JavaScript*, w przeciwieństwie do *Javy*, nie jest statycznie typowany. W odróżnieniu od klasycznego modelu kompilowanych języków, na których bazuje język *Java*, *JavaScript* jest językiem interpretowanym, opartym na niewielkiej ilości podstawowych struktur i typów danych.

JavaScript oferuje programiście większą swobodę działania w tworzeniu kodu, dzięki temu, że nie jest zmuszany do statycznych deklaracji typów, zmiennych czy metod. Dodatkowo *JavaScript* jest pozbawiony słów kluczowych enkapsulacji, dzięki czemu z programisty zostaje zdjęta odpowiedzialność za nadawanie odpowiedniego stopnia dostępu do zmiennych.

4.1.2 HTML

HTML (ang. Hypertext Markup Language) [65] jest standardowym językiem znacznikowym, służącym głównie do tworzenia stron internetowych. Razem z technologiami *CSS* i *JavaScript*, stanowi triadę podstawowych technologii internetowych rekomendowanych przed W3C.

Przeglądarki internetowe interpretują dokumenty *HTML* przychodzące wraz z odpowiedziami z serwerów, bądź te wczytywane z pamięci lokalnej. Interpretery *HTML* są w stanie wyświetlić dla użytkownika przeglądarki strukturę zdefiniowaną za pomocą znaczników języka. Dzięki *HTML*, programista jest w stanie opisać semantycznymi znacznikami strukturę strony oraz dać wskazówki na temat jej wyglądu.

HTML może załączać programy napisane w językach skryptowych takich jak *JavaScript*, co może wpływać na zachowanie i funkcjonalność strony internetowej. Dzięki załączaniu arkuszy *CSS*, *HTML* może definiować wygląd strony i układ treści na niej zawartych. W3C rekomenduje używanie *CSS* dla opisywania wyglądu strony od 1997.

Niemniej jednak, specyfikacja języka *HTML* oferuje więcej niż tylko reprezentacje tekstu w dokumencie. Jest to między innymi możliwość definiowania *metadanych* strony internetowej, które są wykorzystywane przez programy asystujące, do których należą również czytniki ekranu.

W kontekście kompatybilności strony z czynnikami ekranu, znajomość struktury dokumentu opisanej przy pomocy znacznikowego języka *HTML*, jest konieczna. Program zaprezentowany w ramach projektu pracy magisterskiej, zawiera zestaw metod wykonujących operacje na dokumentach *HTML*.

4.1.3 CSS

CSS (ang. Cascading Style Sheets) [66], jest językiem opisującym prezentację stron internetowych, między innymi kolory, układ strony i czcionki. Umożliwia definiowanie różnych sposobów prezentacji w zależności od urządzenia czy rozdzielczości.

CSS jest niezależny od *HTML* i może być używany z dowolnym językiem znacznikowym opartym na *XML*.

Dzięki funkcjonalności oferowanej prze *CSS*, możemy oddzielić definicję wyglądu strony od deklaracji jej struktury (*HTML*). Ułatwia to utrzymywanie i rozwój witryn internetowych, udostępnianie i rozprzestrzenianie arkuszy styli i dostosowywanie ich do różnych środowisk.

Możliwości kaskadowych arkuszy styli oraz ich wsparcie w projektowaniu stron internetowych mają realny wpływ na strukturyzację i standaryzację kodu. Rozszerzenie narzędzi deweloperskich przeglądarki *Google Chrome*, zaprogramowane na potrzeby pracy magisterskiej, wykorzystuje w swoim szablonie, style zdefiniowane przy pomocy *CSS*.

4.2. Budowanie aplikacji i zarządzanie zależnościami

NPM według definicji zamieszczonej na stronie głównej produktu [45], jest managerem pakietów *JavaScript* i największego rejestru oprogramowania na świecie. *NPM* jest przeznaczony do instalacji, zarządzania i dystrybucji zależności w projektach *JavaScript*. Oprogramowanie *NPM* jest używane przez miliony programistów na każdym z kontynentów. Główną zaletą używania *NPM* jest pełna automatyzacja zarządzania zależnościami i pakietami naszej aplikacji. Każda z zależności jest określona

w pliku konfiguracyjnym NPM, *package.json*. Dzięki temu każda osoba rozpoczynająca pracę nad projektem, może pobrać wszystkie pakiety i zależności aplikacji przez wywołanie prostej komendy *npm install*. Ponadto plik konfiguracyjny umożliwia określenie i kontrolę wersji zależności, co daje nam możliwość „uniezależnienia" się od aktualizacji publikowanych przez autorów zależności, bez obaw o przerwanie działania naszej aplikacji.

Wyczerpujący artykuł opublikowany na stronie *smashingmagazine* [46] w przystępny sposób definiuje powody, którymi można się kierować w wyborze kolejnego narzędzia. Mowa o narzędziu do automatycznego ładowania modułów i budowania paczki końcowej aplikacji.

Webpack, jest w ostatnich latach najpopularniejszym wyborem. Aby sprawdzić dlaczego *Webpack* to dobry wybór, należy zastanowić się czy i do czego możemy go użyć w budowaniu aplikacji. W większości języków programowania (w tym w *ES6*, który jest najnowszą specyfikacją języka *JavaScript*, niestety nie wspieraną w pełni w każdej popularnej przeglądarce), możemy podzielić kod na mniejsze moduły aby następnie importować je i używać kodu w nich zawartego. Ta funkcjonalność nie była zaimplementowana natywnie w przeglądarkach internetowych, dlatego dla jej wsparcia powstały *bundler'y*. *Bundler'y* mogą asynchronicznie ładować moduły i uruchamiać je po zakończeniu, bądź budować pojedynczy plik *JavaScript*, zawierający kod wszystkich niezbędnych plików (*paczka końcowa*), który następnie może zostać załączony za pomocą znacznika *script*. Przewaga *Webpack'a* polega na tym, że jest to najnowsze rozwiązanie tego typu, zatem problemy poprzedników zostały rozwiązane wraz z pierwszą oficjalną wersją.

Rozpoczęcie pracy z tym narzędziem jest bardzo proste – jeśli chcemy połączyć (zbudować) kilka plików *.js*, nie potrzebujemy nawet pliku konfiguracyjnego. *Webpack* może być również jedynym narzędziem do kompilacji, którego potrzebujemy w procesie budowania aplikacji. Ogromnym plusem przemawiającym na korzyść tego programu, jest duża społeczność internetowa zorientowana wokół, co jest równoznaczne z ogromnym wsparciem w trakcie pracy.

4.3. TypeScript i testy jednostkowe

TypeScript był oczywistym wyborem ze względu na rozszerzenie funkcjonalności języka *JavaScript* o statyczne typowanie struktur danych. Dodatkowo *TypeSctipt* dostarcza wszystkie funkcjonalności najnowszych specyfikacji *EcmaScript*, zanim zostaną natywnie zaimplementowane w środowiskach przeglądarek. Można zastanawiać się dlaczego dodawać statyczną deklarację typów do języka typowanego dynamicznie? Odpowiedź na to pytanie została udzielona przez wiele organizacji, w tym przez firmę uważaną za autorytet w dziedzinie programowania i projektowania aplikacji, jaką jest *Google*. W artykule zamieszczonym na stronie [47] dowiadujemy się, że *TypeScript* jest oficjalnym językiem używanym w *Google* do rozwoju oprogramowania przeznaczonego dla aplikacji klienckich. Powodem takiej decyzji jest wielokrotne potwierdzenie tego, że typy poprawiają szybkość i łatwość refaktoryzacji kodu i skutecznie redukują ilość nieprzewidzianych wad oprogramowania. Dużo lepiej aby to kompilator znalazł błędy w naszej implementacji niż klient w trakcie używania opublikowanego oprogramowania. Dodatkowo statycznie typowany kod jest najlepszą formą dokumentacji, którą możemy przedstawić programistom dołączającym do projektu. Istnieje wiele innych powodów przemawiających za wyborem tej technologii w procesie tworzenia oprogramowania, jednak te wymienione wcześniej były najistotniejsze w trakcie podejmowania decyzji związanych z projektem pracy magisterskiej.

Prawidłowe działanie kodu i testy jednostkowe to nierozłączne elementy. Decyzja o utrzymaniu wysokiego poziomu pokrycia kodu biblioteki testami jednostkowymi, była naturalna. Jedyną kwestią wartą zastanowienia był wybór technologii służącej do programowania testów jednostkowych aplikacji *JavaScript.*

Wybraną technologią jest **Karma i Jasmine**. *Karma* [48], stworzona przez programistów *Google'a* pracujących nad *frameworkiem front-end'owym Angular*, jest nowoczesnym środowiskiem testowym aplikacji *JavaScript*. Nie wymaga ono skomplikowanej konfiguracji a jest raczej oprogramowaniem, w którym programista ma napisać kod i otrzymać natychmiastową odpowiedź zwrotną z testów, ponieważ informacja o poprawności kodu jest czynnikiem wpływającym pozytywnie na naszą produktywność. Dzięki testom jednostkowym pisanym w *Karma i Jasmine*, rozwój oprogramowania jest dużo przyjemniejszy i bezpieczny.

5. Wprowadzenie do projektu

Głównym założeniem projektu, jest rozwiązanie najczęstszych błędów popełnianych przez programistów w związku z implementacją standardu WCAG-2.0 i ogólnie rozumianego wsparcia czytników ekranu. Projekt składa się z kilku części:

- Biblioteki *JavaScript*, udostępniającej zestaw metod pomocnych przy poprawianiu jakości kodu pod względem kompatybilności z czytnikami ekranu. Na jej zawartość składają się dwie główne funkcjonalności opisane poniżej.

  Zestaw serwisów oferujących modyfikacje związane z implementacją standard WCAG-2.0. Należą do nich serwis udostępniający metody wspomagające nadawanie atrybutów *aria*, serwisy dostarczające funkcjonalności umożliwiające modyfikację nawigacji klawiszowej, serwis pozwalający na edycję i dodawanie atrybutów związanych ze standardem WCAG-2.0 oraz serwis zawierający metody wspomagające pracę nad metadanymi strony internetowej.

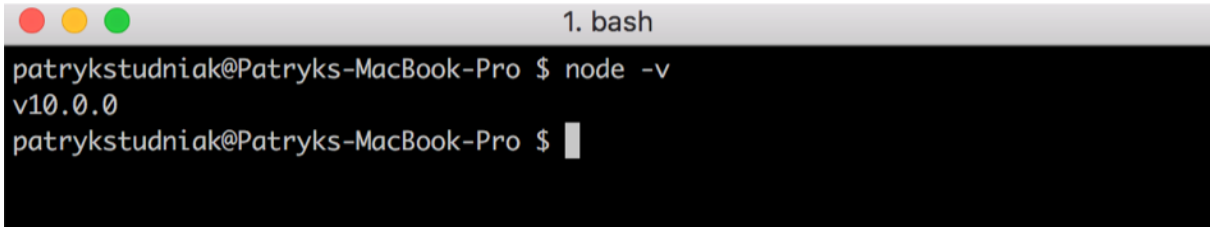
  Program interpretujący, dający możliwość wykonywania operacji na stronie internetowej przy pomocy odpowiednio przygotowanego i załadowanego do programu pliku konfiguracyjnego (instrukcji zadań) w formacie .js bądź .json. Taki sposób implementacji nadaje elastyczny charakter proponowanym rozwiązaniom i pozwala na różne sposoby użycia.
- Drugą część projektu stanowi rozszerzenie narzędzi developerskich przeglądarki *Google Chrome*, stanowiące jednocześnie przykładowy sposób użycia biblioteki. Zawiera ono przyjazny interfejs, dostępny z poziomu narzędzi developerskich przeglądarki. Ta część rozwiązania została zaprojektowana z myślą o osobach, znających techniki komputerowe i inżynierię oprogramowania w podstawowym stopniu. Za jej pośrednictwem, użytkownik ma możliwość wygenerowania i pobrania pliku konfiguracyjnego, bez konieczności bezpośredniego używania serwisów oferowanych przez bibliotekę oraz dokładnej znajomości, wymaganej struktury pliku. Wygenerowany plik konfiguracyjny może następnie posłużyć jako punkt wejścia interpretera biblioteki, który na podstawie jego zawartości wykona odpowiednie operacje na stronie internetowej.
- Trzecią, ostatnią częścią projektu jest strona internetowa pozbawiona wparcia dla czytników ekranu i standardu WCAG-2.0, która posłuży jako przykład wykorzystania biblioteki. Dzięki niej dowiemy się w jaki sposób wykonywane są modyfikacje na stronach internetowych za pośrednictwem biblioteki. Przedstawione zostaną również sposoby użycia i przeznaczenie programu zaprezentowanego w ramach pracy magisterskiej.

5.1. Implementacja rozwiązań

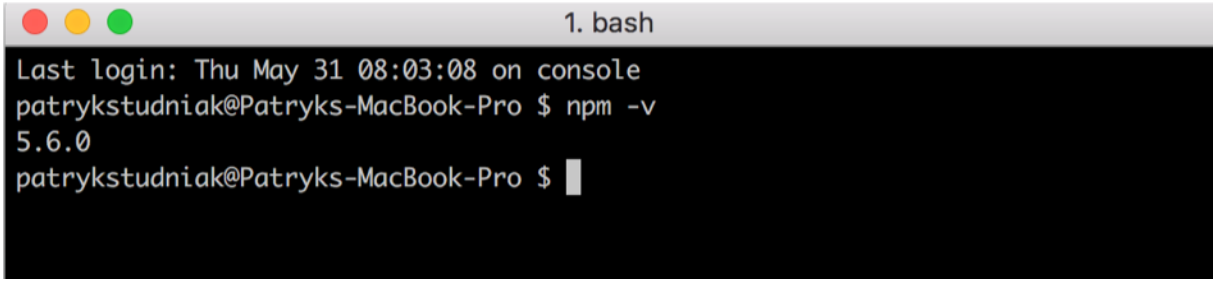
Wszystkie opisywane kroki instalacji i pliki konfiguracyjne są zgodne z systemem operacyjnym Mac OS X i mogą się różnić względem innych systemów operacyjnych. Dokładny opis implementacji rozwiązań wiąże się z głównymi elementami projektu którymi są biblioteka instrukcji JavaScript oraz rozszerzenie narzędzi deweloperskich przeglądarki *Google Chrome*.

W celu używania managera paczek *NPM* należy wykonać odpowiednie kroki. Każdy z nich jest opisany szczegółowo w dokumentacji projektu [49]. Pierwszym krokiem, który należy wykonać jest dokładne zapoznanie się z dokumentacją. Następnie należy postępować zgodnie z krokami opisanymi w dokumentacji w celu założenia konta na serwerze *npm.js*. *NPM* jest zaprogramowany za pomocą *Node.js*. Niezbędna jest instalacja najnowszej, oficjalnej wersji tego oprogramowania na naszej

maszynie. Istotnym jest aby zainstalować wersję oznaczoną jako *LTS (Long term support)*, ponieważ pozostałe wersje nie zostały przetestowane pod względem kompatybilności z *NPM*. Po zainstalowaniu, w oknie terminala, możemy sprawdzić wersję zainstalowanego oprogramowania, używając do tego komendy *node -v* (Rys.1). W celu sprawdzenia wersji *NPM* możemy wywołać komendę *npm -v* (Rys.2).



Rysunek 1 Widok konsoli po wywołaniu komendy sprawdzającej wersję oprogramowania node.js



Rysunek 2 Widok konsoli po wywołaniu komendy sprawdzającej wersję oprogramowania npm

Ponieważ oprogramowanie *Node.js* i *NPM* są zarządzane przez różne jednostki, aktualizacje i obsługa mogą stać się problematyczne i złożone. Dodatkowo *Node.js* jest instalowany w katalogu, do którego prawa dostępu posiada jedynie administrator systemu. Może to zatem powodować błędy uprawnień podczas próby instalacji pakietów globalnie. Aby rozwiązać te problemy, wielu programistów decyduje się na użycie managera wersji *Node.js* lub *NVM* (*Node Virtual Machine*) do zainstalowania *NPM* i utrzymywania najnowszej kompatybilnej wersji oprogramowania. Ponadto *NVM* umożliwia łatwe testowanie oprogramowania dla różnych wersji *Node.js*, dzięki możliwości szybkiego przełączania się pomiędzy wersjami. Na potrzeby tej pracy magisterskiej nie zostały wykorzystane technologie wspomagające proces aktualizacji managera paczek *NPM*.

Po zainstalowaniu *Node.js* i *NPM* możemy przejść do utworzenia projektu. W tym celu tworzymy nowy katalog z nazwą projektu i otwieramy wewnątrz nowe okno terminala. Inicjujemy nowy projekt wykonując komendę *npm init* i przechodzimy kolejno przez wszystkie kroki.

Po przejściu wszystkich kroków procedury, zostanie wygenerowany nowy plik konfiguracyjny *package.json*, zwierający podstawową strukturę naszej aplikacji (Listing 1). Zawiera informacje podane przez nas w momencie inicjalizacji projektu i w dalszej części pracy nad projektem, będzie interpretowany przez managera paczek. Za jego pośrednictwem będziemy mogli zarządzać wszystkimi zależnościami projektu, zarówno tymi końcowymi jak i tymi wymaganymi jedynie w procesie wytwarzania oprogramowania.

```
{
  "name": "sdk",
```

```
  "version": "1.0.0",
  "description": "Web sdk helper method for ad-hoc adjustment of html document for screen readers compatibility",
  "main": "index.js",
  "scripts": {
    "test": "test"
  },
  "keywords": [
    "screen",
    "reader",
    "aria",
    "sdk",
    "html",
    "ada",
    "disabilities",
    "web",
    "accessibility"
  ],
  "author": "Patryk Studniak",
  "license": "ISC"
}
```

Listing 1 Wygenerowany plik package.json

Aby dodać zależności do projektu, należy utworzyć dwie kolejne sekcje w pliku konfiguracyjnym *NPM*. Są nimi *dependencies* i *devDependencies*. Są to klucze pod, którymi znajdują się pary klucz-wartość, gdzie klucz jest dokładną nazwą zależności, a wartość jest jego dokładną wersją. Do projektu pracy magisterskiej dodane zostaną zależności wymagane dla technologii użytych w projekcie (Listing 2). Opisane zależności są zależnościami biblioteki instrukcji JavaScript zaprogramowanej w ramach projektu pracy magisterskiej.

```
"dependencies": {
      "ts-loader": "^2.3.7",
      "tslint": "^5.8.0",
      "tslint-loader": "^3.5.3",
      "typescript": "^2.8.3",
      "uglifyjs-webpack-plugin": "^1.2.2",
      "webpack": "^3.6.0"
   },
   "devDependencies": {
      "@types/jasmine": "^2.6.0",
```

```
    "jasmine": "^2.8.0",
    "json-loader": "^0.5.7",
    "karma": "^1.7.1",
    "karma-jasmine": "^1.1.0",
    "karma-phantomjs-launcher": "^1.0.4",
    "karma-typescript": "^3.0.12"
  }
}
```

Listing 2 Plik package.json z uwzględnionymi sekcjami dependencies i devDependencies

Zainstalowane zależności są niezbędne w projekcie dla wsparcia języka *TypeScript*, dynamicznego ładowania modułów aplikacji za pomocą *Webpack'a* oraz przeprowadzania testów jednostkowych z użyciem *Karma/Jasmine*.

TS Loader [50] jest programem załadowującym pliki z rozszerzeniem *.ts* dla kompilatora, pomagając w rozwiązaniu zależności w tych plikach (*imports*). Jest niezbędnym narzędziem do pracy z technologiami *webpack* i *typescript*.

TSLint [51] jest rozszerzalnym narzędziem do statycznej analizy kodu. Sprawdza kod *TypeScript* pod względem czytelności, łatwości utrzymania i błędów funkcjonalnych. Jest szeroko obsługiwany przez nowoczesne edytory kodu i środowiska deweloperskie oraz systemy kompilacji. Można go dostosować za pomocą dostępnych i własnych reguł, konfiguracji i formaterów, zadeklarowanych w pliku konfiguracyjnym *tslint.json*.

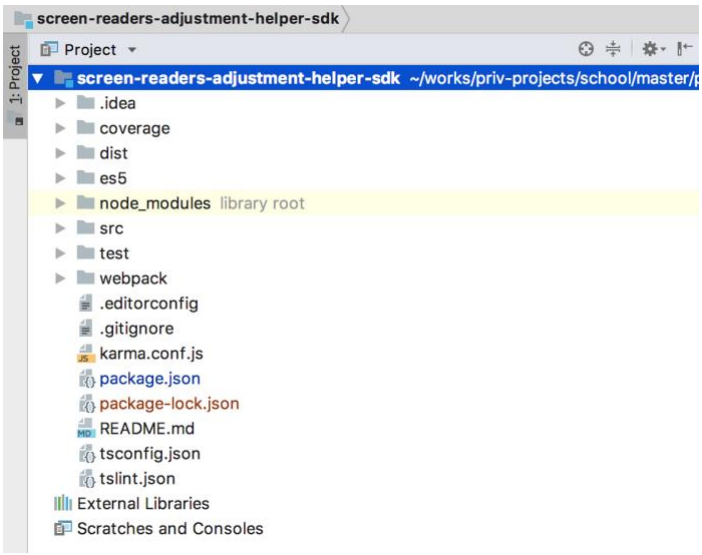
TypeScript [51] opisany w poprzednim rozdziale, jest rozszerzeniem funkcjonalności języka *JavaScript* dostarczanym przez firmę *Microsoft*. Dodaje opcjonalne typy, klasy i moduły dla *JavaScript'u*. Skompilowany *TypeScript* jest poprawnym, w pełni zgodnym ze standardami kodem języka *JavaSctipt.*

Webpack [52] jest opisanym wcześniej, narzędziem służącym do automatyzacji budowania paczek końcowych *JavaScript*. Istnieje możliwość używania niezależnych rozszerzeń oprogramowania. Jednym z nich jest używany w ramach projektu pracy magisterskiej **Uglify Webpack Plugin**. Jest to rozszerzenie służące do minifikacji kodu *JavaScript*, dla zminimalizowania rozmiaru paczki.

Wśród zależności wymaganych jedynie w trakcie programowania aplikacji, możemy wyszczególnić grupę zależności umożliwiających przeprowadzanie testów jednostkowych aplikacji. W jej skład wchodzą **Jasmine** [53], który jest framework'iem do przeprowadzania testów aplikacji *JavaScript*. Nie polega na przeglądarkach, *DOM* ani żadnej innej strukturze *JavaScript*, dzięki czemu nadaje się zarówno do testów aplikacji klienckich jak i serwerowych. Aby możliwe było przeprowadzanie testów jednostkowych za pomocą tego framework'a, należy zasymulować środowisko przeglądarki, do czego służy kolejna zależność – **PhantomJS**. Jest to wirtualnie generowane i uruchamiane środowisko przeglądarki, działające bez interfejsu (*headless*) i służące do przeprowadzania testów jednostkowych.

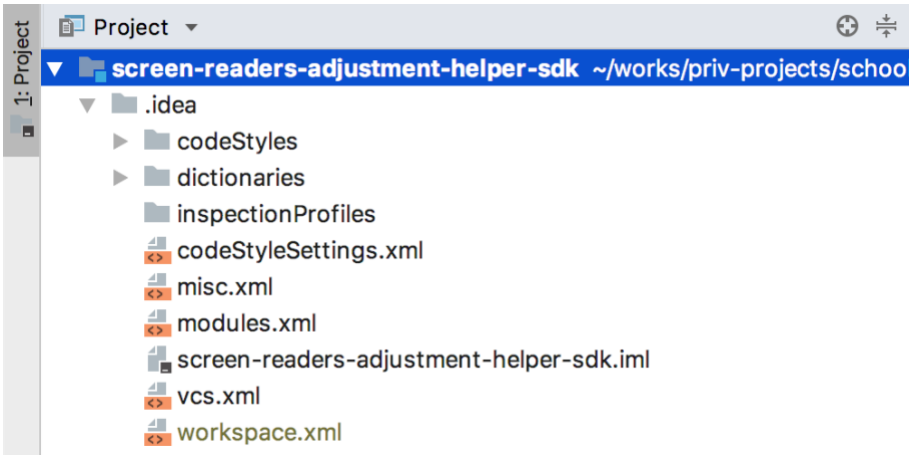
5.1.1 Struktura projektu biblioteki JavaScript

Istotną częścią każdego projektu programistycznego jest konfiguracja. Poniżej znajduje się prezentacja struktury aplikacji, stanowiąca część projektu pracy magisterskiej, biblioteka *JavaScript* (Rys.3). Widok struktury katalogów projektu biblioteki JavaScript.



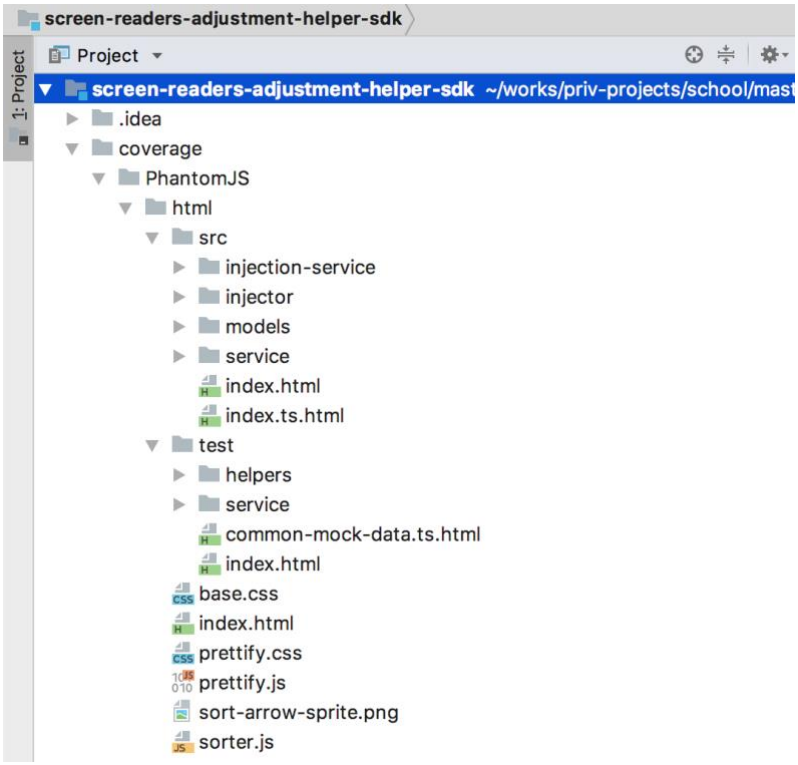
Rysunek 3 Struktura plików i katalogów projektu biblioteki JavaScript

Na strukturę tej aplikacji składa się kilka katalogów i plików, z których każdy ma ogromne znaczenie w kontekście budowy aplikacji. Folder .idea (Rys.4) zawiera pliki konfiguracyjne środowiska programistycznego firmy *JetBrains IntelliJ Idea* [57] takie jak słownik, ustawienia modułów i środowiska, wygląd (*theme*) oprogramowania oraz sposób interpretacji, kolorowania i podświetlania składni.



Rysunek 4 Widok zawartości katalogu .idea projektu biblioteki JavaScript

Folder *coverage* jest generowany automatycznie po wykonaniu testów jednostkowych aplikacji za pomocą framework'a *Jasmine* w środowisku *PhantomJS*. Dzięki tej funkcjonalności, test runner jest w stanie wygenerować pełen raport z przeprowadzonych testów w formie lokalnej strony internetowej. Wewnątrz folderu (Rys.5) znajduje się zestaw plików HTML (Rys.6, 7) odpowiadający każdemu wywołanemu testowi jednostkowemu oraz jednostce testowanej.



Rysunek 5 Widok zawartości folderu coverage projektu biblioteki JavaScript

Po uruchomieniu pliku *index.html* w głównym katalogu, możemy wyświetlić i przeanalizować wyniki przeprowadzonych testów (Rys.6, Rys.7), sprawdzić raport o procentowym pokryciu specyficznych części kodu jako całości bądź podzielonego na testowalne jednostki.

/

95.6% Statements 500/523 **85.84%** Branches 97/113 **96.3%** Functions 130/135 **95.12%** Lines 429/451

File		Statements		Branches		Functions		Lines	
src/		100%	20/20	100%	0/0	100%	0/0	100%	20/20
src/models/		100%	22/22	100%	8/8	100%	4/4	100%	22/22
src/service/aria-landmarks/		100%	26/26	100%	3/3	100%	10/10	100%	22/22
src/service/aria-landmarks/model/		100%	8/8	100%	2/2	100%	1/1	100%	8/8
src/service/attributes/		100%	26/26	100%	3/3	100%	10/10	100%	22/22
src/service/attributes/model/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
src/service/config-interpreter/		93.33%	28/30	83.33%	5/6	100%	4/4	92.31%	24/26
src/service/config-interpreter/model/		100%	7/7	100%	2/2	100%	1/1	100%	7/7
src/service/config-interpreter/service-executers/		88.89%	80/90	75%	18/24	91.67%	11/12	87.34%	69/79
src/service/config-interpreter/service-executers/decorators/		100%	48/48	100%	18/18	100%	18/18	100%	36/36
src/service/custom-operation/model/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
src/service/helpers/		94.87%	74/78	81.25%	13/16	90.91%	20/22	95.38%	62/65
src/service/helpers/decorators/		100%	32/32	100%	12/12	100%	12/12	100%	24/24
src/service/helpers/models/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
src/service/metatags/		100%	19/19	100%	3/3	100%	7/7	100%	15/15
src/service/metatags/model/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
src/service/skimming-links/		100%	27/27	100%	3/3	100%	7/7	100%	23/23
src/service/skimming-links/model/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
src/service/skimming-road-setter/		84.78%	39/46	53.85%	7/13	86.67%	13/15	82.05%	32/39
src/service/skimming-road-setter/model/		100%	0/0	100%	0/0	100%	0/0	100%	0/0

Rysunek 6 Raport z przeprowadzonych testów jednostkowych biblioteki JavaScript

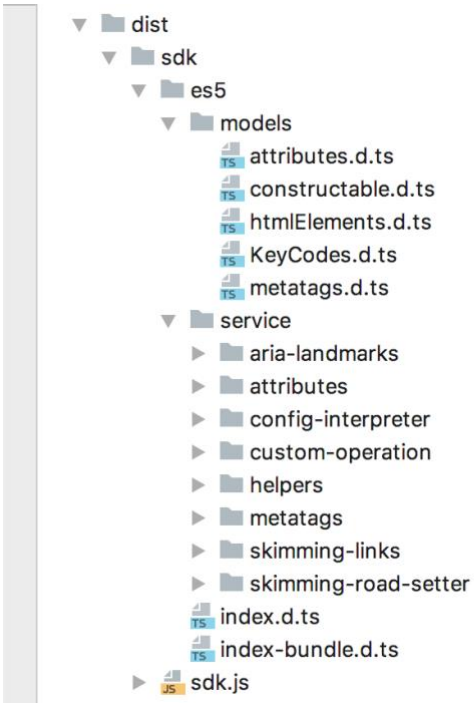
all files / src/service/aria-landmarks/ aria-landmarks-service.decorator.ts

100% Statements 8/8 **100%** Branches 3/3 **100%** Functions 3/3 **100%** Lines 6/6

```
1  1×  import { DefaultAriaLandmarksService } from './aria-landmarks.service';
2      import { AriaLandmarksService } from './model/aria-landmarks-service';
3      import { Constructable } from '../../models/constructable';
4
5  1×  const defaultAriaLandmarksService = new DefaultAriaLandmarksService();
6
7  2×  export function InjectAriaLandmarksService<C extends Constructable>(constructorClass: C): C {
8  4×      return class extends constructorClass {
9  3×          private ariaLandmarksService: AriaLandmarksService = defaultAriaLandmarksService;
10 2×      };
11     }
12
```

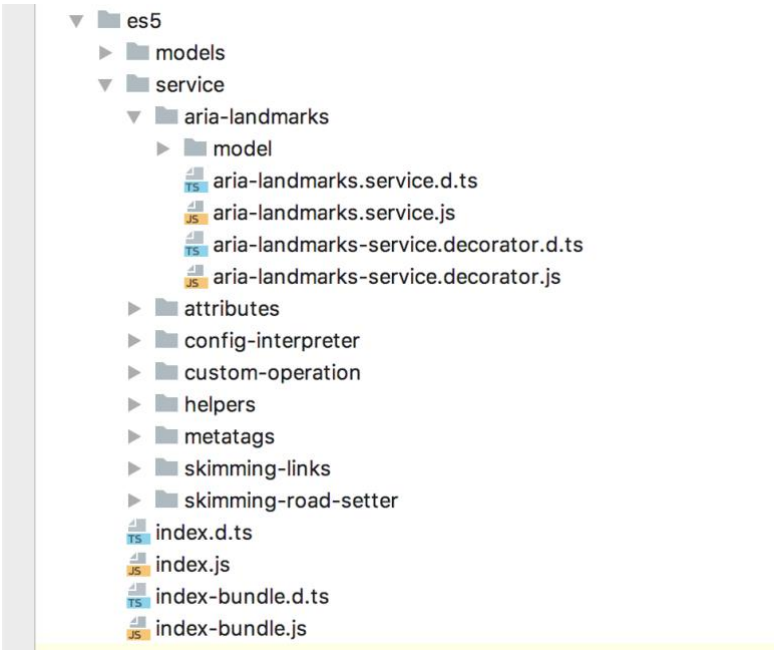
Rysunek 7 Widok szczegółowy raportu z przeprowadzonych testów jednostkowych

Folder *dist* (Rys.8), jest domyślnym folderem zawierającym wynikowe pliki *JavaScript* skompilowane za pomocą *Webpack'a* (wraz z paczką końcową biblioteki) oraz pełen zestaw plików definicji typów. Pliki definicji typów (*definition files*), zawierają deklaracje używanych typów, utworzonych bądź zadeklarowanych w ramach skompilowanego programu, napisanego w *TypeScript*.



Rysunek 8 Zawartość folderu dist, projekru bilbioteki JavaScript

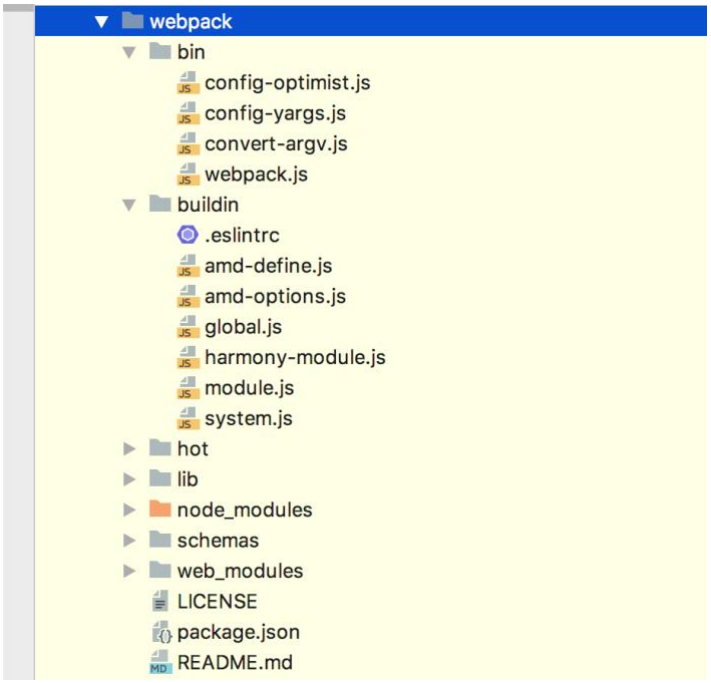
Katalog *es5* (Rys.9) jest zbiorem wszystkich skompilowanych plików *TypeScript* za pomocą kompilatora *tsc* do poprawnego formatu interpretowanego przez silniki przeglądarek, jakim jest .js. Kompilator *tsc* w przeciwieństwie do *webpack'a*, nie buduje plików wynikowych i nie obsługuje dynamicznego załadowywania zależności wewnątrz plików.



Rysunek 9 Zawartość folderu es5, projektu biblioteki JavaScrcipt

Kolejny katalog to *node_modules*. Jest to automatycznie wygenerowany folder przy pomocy managera pakietów i zależności *NPM*. Zawiera wszystkie katalogi i pliki źródłowe zależności projektu

i ich zależności. Po wywołaniu komendy *npm install* bądź *npm i*, utworzony zostaje katalog *node_modules*, gdzie zapisywane są źródła pobieranych i instalowanych zależności, które następnie mogą być używane w projekcie. Instalowane paczki (Rys.10) mogą zawierać pliki wykonywalne, pliki źródłowe z kodem *JavaScript*, skrypty powłoki i pliki definicji typów. Zawartość przykładowego pakietu zależności pobranego i zainstalowanego za pomocą managera pakietów NPM.



Rysunek 10 Przedstawiona zawartość pakietu to Webpack

Kolejne dwa katalogi to *src* zawierający wszystkie pliki źródłowe aplikacji oraz *test*, gdzie przechowywane są pliki testów jednostkowych.

W katalogu *webpack* znajduje się plik *build-bundle.js* (Listing 3), który jest plikiem konfiguracyjnym *webpack'a*.

```
const path = require('path');

module.exports = () => {
  return {
    devtool: 'source-map',
    entry: ['./src/index.ts'],
    output: {
      filename: 'sdk.js',
      path: path.resolve(__dirname, '../dist/sdk/'),
      libraryTarget: 'umd',
      library: 'ScreenReaderHelperSDK',
      umdNamedDefine: true
```

```
    },
    module: {
      rules: [{
        test: /\.json$/,
        loader: 'json-loader'
      }, {
        test: /\.ts$/,
        use: 'ts-loader'
      }, {
        test: /\.ts$/,
        enforce: 'pre',
        loader: 'tslint-loader'
      }]
    },
    resolve: {
      extensions: ['.ts', '.js', '.json']
    }
  }
};
```

Listing 3 Plik konfiguracyjny build-bundle.js

We wspomnianym pliku konfiguracyjnym znajduje się zestaw instrukcji przekazywanych do programu interpretującego (*webpack*). Znajdują się w nim informacje na temat pliku wejściowego (*entry*) i wyjściowego (*output*) oraz modułów, w których zdefiniowane zostały zasady użycia oprogramowania ładującego zależności ze względu na rozszerzenie obsługiwanego pliku. Dla rozszerzenia *json* jest to *json-loader*, dla *ts* to *ts-loader*.

W skład struktury aplikacji wchodzą również pozostałe wymagane pliki konfiguracyjne. Są to przede wszystkim *karma.config.js* (Listing 4), *tsconfig.json* oraz *tslint.json* (Listing 5).

```
module.exports = (config) => {
  config.set({
    basePath: '',
    frameworks: ['jasmine', 'karma-typescript'],
    files: [
      './test/**/*.ts',
      './src/**/*.ts'
    ],
    exclude: [],
    preprocessors: {
```

```
      'src/**/*.ts': ['karma-typescript'],
      'test/**/*.ts': ['karma-typescript']
    },
    reporters: ['progress', 'karma-typescript'],
    karmaTypescriptConfig: {
      include: [
        'src/**/*.ts',
        'test/**/*.ts'
      ],
      coverageOptions: {
        threshold: {
          global: {
            statements: 80,
            branches: 80,
            functions: 80,
            lines: 80
          }
        }
      },
      reports: {
        'html': {
          'directory': 'coverage',
          'subdirectory': 'PhantomJS'
        },
        'text-summary': ''
      }
    },
    port: 9876,
    colors: true,
    browsers: ['PhantomJS'],
    autoWatch: true,
    singleRun: false,
    concurrency: Infinity
  });
};
```

Listing 4 Plik konfiguracyjny karma.conf.js

Na konfigurację framework'a *Karma*, wykorzystywanego wraz z framework'iem *Jasmine* i językiem *TypeScript*, składa się kilka podtawowych właściwości. Jest to deklaracja używanych framework'ów i ścieżka do plików, które mają zastać wstępnie skompilowane przez rozpoczęciem

testów za pomocą odpowiednich preprocesorów (w tym przypadku to *karma-typescript*). Kolejną istotną częścią konfiguracji jest ustawienie portu, na którym mają być uruchamiane testy – bardzo istotne jest ustawienie wolnego portu, tak aby nie powodować konfliktów pomiędzy serwerami aplikacji. Ważna jest także definicja środowiska przeglądarki, w której będą przeprowadzane testy. Na potrzeby biblioteki został użyty *PhantomJS*, który jest wirtualnym środowiskiem nie wymagającym interfejsu przeglądarki. Dodatkowymi opcjami są ustawienia kompilatora *karma-typescript*, gdzie możemy zadeklarować pliki zawierające kod do przetestowania lub te, które chcemy pominąć. Minimalny wymagany poziom pokrycia testami jednostkowymi, wyrażany procentowo, również jest definiowany przy pomocy tych opcji.

```
{
  "extends": "tslint:recommended",
  "rules": {
    "indent": [
      true,
      "spaces",
      4
    ],
    "no-trailing-whitespace": [
      true,
      "ignore-blank-lines"
    ],
    "ordered-imports": false,
    "typedef": [
      true,
      [
        "arrow-call-signature",
        "call-signature",
        "parameter",
        "arrow-parameter",
        "property-declaration",
        "variable-declaration",
        "member-variable-declaration",
        "object-destructuring",
        "array-destructuring"
      ]
    ],
    "interface-name": [
      "never-prefix"
    ],
```

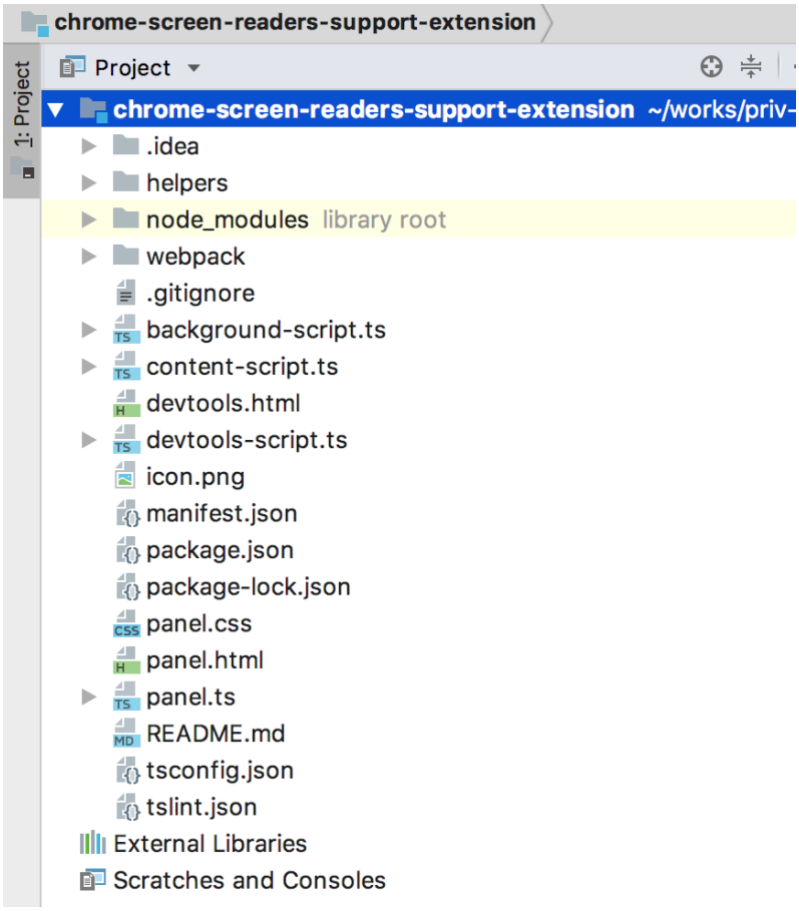
```
    "no-any": true,
    "no-bitwise": true,
    "trailing-comma": [
      true,
      {
        "multiline": "never",
        "singleline": "never"
      }
    ],
    "quotemark": [
      true,
      "single"
    ],
    "no-var-requires": false,
    "max-line-length": [
      true,
      185
    ]
  }
}
```

Listing 5 Plik konfiguracyjny tslint.json

Tslint.json (Listing 5) jest plikiem konfiguracyjnym statycznego analizatora (lintera) języka *TypeScript*. Zawiera zestaw zasad, które będą egzekwowane w trakcie pisania aplikacji. Jeżeli, któraś ze zdefiniowanych zasad zostanie złamana, edytor natychmiast powiadomi o tym programistę.

5.1.2 Struktura projektu rozszerzenia narzędzi deweloperskich

Na strukturę rozszerzenia narzędzi deweloperskich przeglądarki *Google Chrome* (Rys.11), składa się pięć podstawowych plików, którymi są *package.json, manifest.json* (Listing 6), skrypt rozszerzenia (*devtools-script.ts*), skrypt uruchamiany w tle (*background-script.ts*) oraz *content-script.ts*, działający jako skrypt zintegrowany z aktywną stroną internetową.



Rysunek 11 Struktura rozszerzenia narzędzi deweloperskich przeglądarki Google Chrome

```
{
  "name": "Screen Readers Adjuster Extension",
  "short_name": "SR Adjuster",
  "description": "Chrome devtools extension which is using screen readers adjuster SDK to allow non-programmer user to adjust website to be more screen readers compliant",
  "version": "0.0.1",
  "devtools_page": "devtools.html",
  "manifest_version": 2,
  "content_scripts": [{
    "matches": ["<all_urls>"],
    "js": ["content-script.js"]
  }],
  "background": {
    "scripts": ["background-script.js"]
```

```
 }
}
```

Listing 6 Plik manifest.json rozszerzenia narzędzi deweloperskij przeglądarki Google Chrome

Plik *manifest.json* jest najważniejszym elementem rozszerzenia narzędzi deweloperskich *Google Chrome*. Jak podaje dokumentacja projektowa rozszerzeń przeglądarki *Google Chrome* [54], każda aplikacja rozszerzająca narzędzia deweloperskie, powinna mieć sformatowany plik *JSON*, zawierający najważniejsze informacje o rozszerzeniu. W przypadku rozszerzenia zaprogramowanego na potrzeby pracy magisterskiej w pliku *manifest.json* zawarto informacje na temat nazw produktu, opis produktu oraz jego wersję.

Dodatkowo, pliki używane w ramach rozszerzenia również powinny zostać zadeklarowane w odpowiedni sposób. Strona zawierająca szablon, który będzie wyświetlany po otwarciu zakładki, dodanej przez zainstalowanie wtyczki, jest zdefiniowana za pomocą klucza *devtools_page*. Pliki działające w tle, są opisane jako *background*. Skrypty uruchamiane jako zintegrowana część aktywnej strony internetowej, są zdefiniowane jako *content_scripts*. Wewnątrz *content_scripts*, możemy zawrzeć dokładne adresy *URL*, pod którymi skrypty będą uruchamiane. Na potrzeby wymagań pracy magisterskiej, są to wszystkie adresy *URL*.

Pozostałe pliki wchodzące w skład struktury rozszerzenia narzędzi deweloperskich, są analogiczne do tych prezentowanych w poprzednim rozdziale w związku z biblioteką zaprogramowaną w ramach pracy magisterskiej. Są to *package.json, tslint.json, build-bundle.js*.

5.1.3 Przegląd funkcjonalności – biblioteka JavaScript

Biblioteka JavaScript zaprogramowana w ramach projektu pracy magisterskiej, oferuje zestaw instrukcji w formie serwisów, których głównym zadaniem jest poprawa jakości kodu HTML strony internetowej. Dzięki wielomiesięcznej pracy nad poprawą jakości stron internetowych pod względem zgodności ze standardem WCAG-2.0, udało się utworzyć listę najczęstszych błędów i zaniedbań związanych z tą kwestią. Zostały one opisane w poprzednich rozdziałach pracy. Wspomniana lista, posłużyła za listę podstawowych problemów, które powinny rozwiązywać serwisy składające się na bibliotekę.

Pierwszym z serwisów jest *aria-landmarks service* (Listing 7). Za jego pośrednictwem, biblioteka udostępnia publiczne metody związane z atrybutami *aria*. Dzięki nim, użytkownik może w łatwy sposób zaimplementować wybrane atrybuty, bez konieczności znajomości dokładnych standardów atrybutów. Serwis oferuje ułatwiony sposób precyzowania atrybutu *role, labeledBy, banner, complimentary, contentInfo* oraz *label*.

```
export interface AriaLandmarksService {
   defineRole(selector: string, value: string): void;
   defineLabeledBy(selector: string, labeledBy: string): void;
   setBanner(selector: string): void;
   setComplementary(selector: string): void;
   setContentInfo(selector: string): void;
```

```
    setLabel(selector: string, value: string): void;
}
```

Listing 7 Interfejs serwisu aria landmarks service

Drugi serwis zaprogramowany w ramach projektu to *attributes service* (Listing 8). Istotnym zaniedbaniem programistów, w związku ze standardem WCAG-2.0 są atrybuty, rekomendowane jako podpowiedzi dla czytników ekranu. Najczęściej są to atrybuty *alt* i *title*. Są one niezwykle ważne dla poprawnej prezentacji strony internetowej, co zostało opisane w poprzednich rozdziałach pracy. Za pomocą serwisu, użytkownik może w przystępny sposób, bez koniecznej technicznej wiedzy na temat atrybutów, dodać, nadpisać bądź rozszerzyć istniejące atrybuty. Serwis uwzględnia atrybuty *alt, title* i *lang*.

```
export interface AttributesService {
    setAlt(selector: string, value: string): void;
    setTitle(selector: string, value: string): void;
    setLang(selector: string, value: string): void;
    overrideAlt(selector: string, value: string): void;
    overrideTitle(selector: string, value: string): void;
    overrideLang(selector: string, value: string): void;
}
```

Listing 8 Interfejs serwisu attributes service

Kolejnym serwisem oferowanym przez bibliotekę JavaScript, jest *metatags service* (Listing 9). W trakcie badań, przeprowadzanych na stronach internetowych klientów zlecających poprawę jakości ich serwisów w związku ze wsparciem czytników ekranu, istotną obserwacją były zaniedbania wśród meta danych strony internetowej. Z opisów przedstawionych w poprzednich rozdziałach, możemy dowiedzieć się jak istotną rolę dla czytników ekranu odgrywają metadane strony internetowej. Jeden z serwisów zawartych w bibliotece przedstawionej w projekcie, zawiera zestaw kilku metod, pozwalających na dodanie, edycję bądź nadpisanie najczęściej pomijanych bądź niewłaściwie implementowanych metadanych strony internetowej. Są to *meta title, meta description* oraz *meta author*.

```
export interface MetatagsService {
    setDocumentTitle(value: string): void;
    setDocumentDescription(value: string): void;
    setDocumentAuthor(value: string): void;
}
```

Listing 9 Interfejs serwisu metatags service

Serwis *skimming links service* (Listing 10) jest odpowiedzią na częsty brak linków umożliwiających pominięcie elementów powtarzających się na stronach. Opisany w poprzednich rozdziałach pracy sposób użycia *skimming linków* oraz jego rola w poprawieniu jakości nawigacji klawiszowej, stał się inspiracją do utworzenia serwisu wspomagającego tworzenie linków. Metody

oferowane przez serwis, pozwalają na automatyczne utworzenie i dodanie poprawnie skonfigurowanego linku, w odpowiednie miejsce w strukturze strony bądź tylko dodać podany link bądź tylko takowy link wygenerować i dać możliwość programiście dalszej pracy z nim.

```
export interface SkimmingLinksService {
   generateSkimmingLink(text: string, jumpTo: string): HTMLAnchorElement;
   appendSkimmingLink(parentNodeSelector: string, link: HTMLAnchorElement): void;
   generateAndAppendSkimmingLink(parentNodeSelector: string, text: string, jumpTo: string): void;
}
```

Listing 10 Interfejs serwisu skimming links service

W związku z nawigacją klawiszową oraz problemami związanymi z jej niewłaściwą implementacją, niezbędną funkcjonalnością jest możliwość ponownego ustawienia szczegółowej ścieżki nawigacyjnej. Serwis *skimming road seter* (Listing 11) jest odpowiedzią na tę potrzebę. Za jego pomocą możemy ustawić dokładną ścieżkę, po której przeprowadzany będzie użytkownik nawigujący przy pomocy klawiszy *tab* i *shift*.

```
export interface SkimmingRoadSetterService {
   generateSkimmingRoad(selectors: string[]): void;
}
```

Listing 11 Interfejs serwisu skimmin road seter service

Serwisy zaprezentowane w ramach biblioteki, udostępniają funkcjonalności rozwiązujące kilka istotnych problemów, najczęściej pojawiających się w związku ze wsparciem czytników ekranu. Niestety nie jest możliwe pełne pokrycie wszystkich przypadków. Aby biblioteka pozostała uniwersalna i mogła być używana w każdej sytuacji, należy zaprojektować ją w sposób nadający jej elastyczność i możliwość dostosowywania do pojawiających się wymagań.

Aby osiągnąć pożądany efekt, biblioteka została wyposażona w generyczny serwis *custom operation service* (Listing 12), który daje możliwość wywołania dowolnego, poprawnego kodu *JavaScript* bądź wykonania, załadowanego za pomocą *require*, pliku *JavaScript* zawierającego odpowiednie instrukcje.

```
export interface CustomOperationService {
   invoke(params: CustomParams): void;
}
```

Listing 12 Interfejs serwisu custom operation service

```
export const customServiceExample = {
   invoke: (customParams) => testMethod(customParams)
};

export const testMethod = (params) => {
```

```
    const body = document.querySelector('body');
    const newElement = document.createElement('h1');
    newElement.textContent = params.text;
    body.appendChild(newElement);
};
```

Listing 13 Przykład użycia custom operation service – deklaracja metody

```
{
    useService: RegisteredServices.customService,
    params: {
        code: customServiceExample,
        customParams: {
            text: 'TEST CUSTOM PARAMS'
        }
    }
}
```

Listing 14 Przykład użycia custom operation service – deklaracja w pliku konfiguracyjnym

Metody rozwiązujące podstawowe problemy z implementacją wsparcia standardu WCAG-2.0 oraz uniwersalność i elastyczność, osiągnięta poprzez implementację serwisu umożliwiającego wywołanie dowolnego kodu *JavaScript*, to nie wszystko co oferuje biblioteka.

Aby rozwiązanie oferowane przez bibliotekę, spełniało wszystkie założenia postawione w poprzednich rozdziałach pracy, powinna być łatwa w użyciu i możliwa do wykorzystania przez osoby znające technologie komputerowe i inżynierię oprogramowania w podstawowym stopniu. Gdyby implementacja funkcjonalności oferowanych przez bibliotekę kończyła się na serwisach, nie byłoby to możliwe. Do jej wykorzystania potrzebna byłaby znajomość technologii internetowych znacząco wykraczająca poza standardy określane mianem podstawowej wiedzy z danej dziedziny.

Oczywiście jest to również jeden z możliwych sposobów użycia biblioteki. Aby jednak umożliwić korzystanie z biblioteki osobom nie będącym specjalistami w dziedzinie aplikacji internetowych i rozwiązań klienckich, udostępniona została możliwość konfiguracji wsparcia czytników ekranu strony internetowej, za pomocą pliku konfiguracyjnego.

Przy pomocy zaprogramowanego na potrzeby pracy magisterskiej interpretera (Listing 15), użytkownik może w pełni wykorzystać możliwości oferowane przez bibliotekę, bez konieczności programowania i bezpośredniego używania biblioteki a jedynie poprzez załadowanie odpowiednio przygotowanego pliku konfiguracyjnego.

```
export interface ConfigInterpreterService {
    executeConfigFile(config: ConfigFile): void;
}
```

Listing 15 Interfejs interpretera plików konfiguracyjnych biblioteki

5.1.4 Przegląd funkcjonalności – rozszerzenie Chrome DevTools

Z punktu widzenia zasad wyznaczanych przez organizację *WAI* i standard WCAG-2.0, udostępnienie możliwości korzystania z biblioteki jak największej liczbie osób jest niezwykle ważne. Aby osiągnąć zamierzony cel, należy w jak największym stopniu zminimalizować konieczność programowania podczas implementacji rozwiązań z użyciem biblioteki. Opisana w poprzednim rozdziale pracy magisterskiej, możliwość automatycznego wykorzystywania możliwości biblioteki za pomocą plików konfiguracyjnych, otwiera wiele nowych możliwości dla wykorzystania biblioteki.

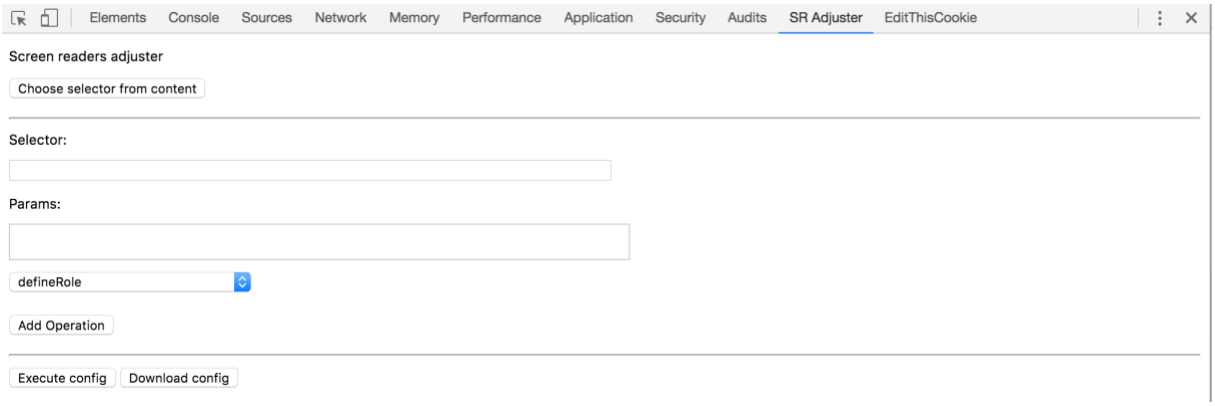
Maksymalizacja liczby osób mogących korzystać z projektu, wiąże się z minimalizacją poziomu skomplikowania, sposobu tworzenia pliku konfiguracyjnego. Zakładając, że plik konfiguracyjny może zostać utworzony przez osobę z podstawową znajomością technik komputerowych, bądź wygenerowany automatycznie przez oprogramowanie wspomagające ten proces, możemy spodziewać się kilku korzyści.

Pierwszą i najistotniejszą korzyścią jaką niesie za sobą maksymalne uproszczenie użycia biblioteki, jest minimalizacja kosztów. Właściciel produktu z brakami w implementacji standardów WCAG-2.0, nie będzie zmuszony zatrudniać osoby w pełni technicznej, posiadającej doświadczenie w pracy z technologiami internetowymi, a będzie mógł przejść przez cały proces osobiście, nie ponosząc dodatkowych kosztów.

Kolejną korzyścią, niemniej ważną od poprzedniej, jest ilość potencjalnych użytkowników rozwiązania. Dzięki temu, że implementacja i sposób użycia są łatwe i przystępne dla każdego użytkownika, liczba potencjalnych klientów wzrasta.

Wymienione wcześniej korzyści, skłaniają do podjęcia decyzji o dołożeniu wszelkich starań do tego, aby biblioteka była łatwa w użyciu. Efektem rozważań nad sposobem takiej implementacji, jest druga część projektu pracy magisterskiej, a mianowicie rozszerzenie narzędzi deweloperskich przeglądarki *Google Chrome*.

Implementacja rozszerzenia narzędzi deweloperskich daje możliwość wygenerowania pliku konfiguracyjnego, interpretowanego przez bibliotekę z projektu pracy. Proces generacji jest łatwy i nie wymaga znajomości języków programowania i technik programistycznych. Za pomocą przyjaznego interfejsu (Rys.12), użytkownik może wygenerować każdą operację dostępną w ramach implementacji biblioteki.



Rysunek 12 GUI rozszerzenia narzędzi deweloperskich przeglądarki Google Chrome

Możliwości interfejsu sięgają ponad manualne wpisywanie selektorów i parametrów funkcji oferowanych przez bibliotekę. Każd użytkownik musiałby znać dokładną implementację interfejsów każdego z serwisów oraz posiadać wiedzę umożliwiającą utworzenie odpowiedniego selektora elementu. Dzięki możliwościom oferowanym przez interfejs programistyczny przeglądarki *Google Chrome*, zaimplementowana została funkcjonalność tworzenia selektora elementu, poprzez kliknięcie na niego. Aby wybrać selektor, należy zablokować interfejs strony internetowej przyciskiem *choose selector from content*, a następnie kliknąć wybrany element na stronie.

Po wybraniu odpowiedniej operacji, automatycznie zostaną wyświetlone pola, odpowiadające parametrom wymaganym dla danej metody (Rys.13). Po wykonaniu tych kroków, możemy użyć przycisku *add operation* aby dodać operację do pliku konfiguracyjnego. Możemy zobaczyć jej sformatowaną reprezentację pojawiającą się w formie przypisu (Rys.14).

Rysunek 13 Interfejs rozszerzenia narzędzi deweloperskich przeglądarki *Google Chrome*, po wybraniu odpowiedniego selektora i operacj

Rysunek 14 Interfejs rozszerzenia narzędzi deweloperskich przeglądarki *Google Chrome*, po dodaniu operacji do pliku konfiguracyjnego

Po wygenerowaniu pliku konfiguracyjnego, możemy go uruchomić na aktywnej stronie internetowej, za pomocą przycisku *execute config*. Taka implementacja będzie działała do momentu pierwszego odświeżenia strony.

Aby zaimplementować wygenerowaną konfigurację na stałe, należy pobrać plik konfiguracyjny (Listing 16) za pomocą przycisku *download config* i załączyć go do strony używając interpretera biblioteki.

```
[
  {
    "index": 0,
    "selector": "body > div > #topnav > #fatnav > ul > li:nth-child(3) > ul > li:nth-child(3) > ul > li:nth-child(1) > a",
    "useService": "ariaLandmarksService",
    "operation": "setComplementary",
    "params": {
      "value": "test value",
      "labeledBy": "test labeled by"
    }
  }
]
```

Listing 16 Zawartość wygenerowanego pliku konfiguracyjneg

5.1.5 Przegląd wybranych fragmentów kodu i technik programistycznych

Implementacja programów składających się na projekt pracy magisterskiej była przeprowadzana z ogólnie przyjętymi zasadami i dobrymi praktykami programistycznymi. Opisywany w poprzednich rozdziałach paradygmat, określany skrótem *SOLID*, był głównym wyznacznikiem wpływającym na wysoki standard wytworzonego oprogramowania. Oprócz odpowiednio zastosowanej segregacji interfejsów, rozwiązaniem poprawiającym jakość, testowalność oraz czytelność kodu, jest

wykorzystanie refleksji metadanych i koncepcyjnych dekoratorów języka *TypeScript* (Listing 17, Listing 18).

Przy pomocy dekoratorów, odpowiedzialność tworzenia instancji zależności wewnątrzklasowych została zdjęta z klas. Dekoratory okazały się również pomocne w implementacji wzorca projektowego *singleton* (Listing 19), szeroko stosowanego w kodzie biblioteki.

```
const defaultAttributesService = new DefaultAttributesService();

export function InjectAttributesService<C extends Constructable>(constructorClass: C): C {
  return class extends constructorClass {
    private attributesService: AttributesService = defaultAttributesService;
  };
}
```

Listing 17 Kod źródłowy dekoratora wstrzykującego attributes service

```
@InjectAttributesHelper
export class DefaultAttributesService implements AttributesService {
  private attributesHelper: AttributesHelper;

  public overrideAlt(selector: string, value: string): void {
    this.attributesHelper.setAttributeOnCollection(selector, Attribute.alt, value, true);
  }

  public setAlt(selector: string, value: string): void {
    this.attributesHelper.setAttributeOnCollection(selector, Attribute.alt, value);
  }
```

Listing 18 Fragment kodu źródłowego attributes service, wraz z przykładem użycia dekoratora attributes helper

```
const defaultAriaLandmarksService = new DefaultAriaLandmarksService();

export function InjectAriaLandmarksService<C extends Constructable>(constructorClass: C): C {
  return class extends constructorClass {
    private ariaLandmarksService: AriaLandmarksService = defaultAriaLandmarksService;
  };
}
```

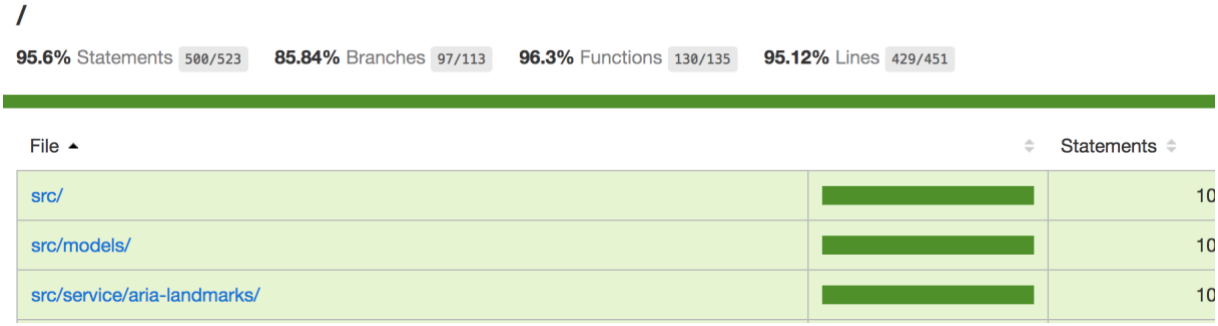
Listing 19 Kod źródłowy dekoratora aria landmarks service jako przykład implementacji wzorca projektowego singleton

Łatwa implementacja i możliwość rozwoju testów jednostkowych z użyciem framework'ów *Karma/Jasmine* (Listing 20), w dużym sopniu przyczyniły się do utrzymania wysokiej jakości kodu

źródłowego. Nie wymagający skomplikowanej konfiguracji framework, pozwolił na utrzymanie ponad 90% pokrycia testami jednostkowymi linii kodu, funkcji i instrukcji warunkowych, co możemy zaobserwować w wynikach testów (Rys.15).

```
describe('DefaultAriaLandmarksService', () => {
  const service = new DefaultAriaLandmarksService();
  const testAttributeValue = 'test value';
  const testAdditionalValue = 'test value';
  let element: HTMLElement;

  beforeEach(() => {
    element = document.querySelector('body');
  });

  it('should have a defineRole method which overrides role attribute of given elements', (done) => {
    service.defineRole('body', testAttributeValue);
    service.defineRole('body', testAdditionalValue);
    expect(element.getAttribute(AriaLandmark.role)).toEqual(testAdditionalValue);
    done();
  });
```

Listing 20 Fragment implementacji testu jednostkowego z użyciem framework'a Karma i Jasmine



Rysunek 15 Wyniki testów jednostkowych, z uwzględnieniem procentowego pokrycia testami danych części kodu

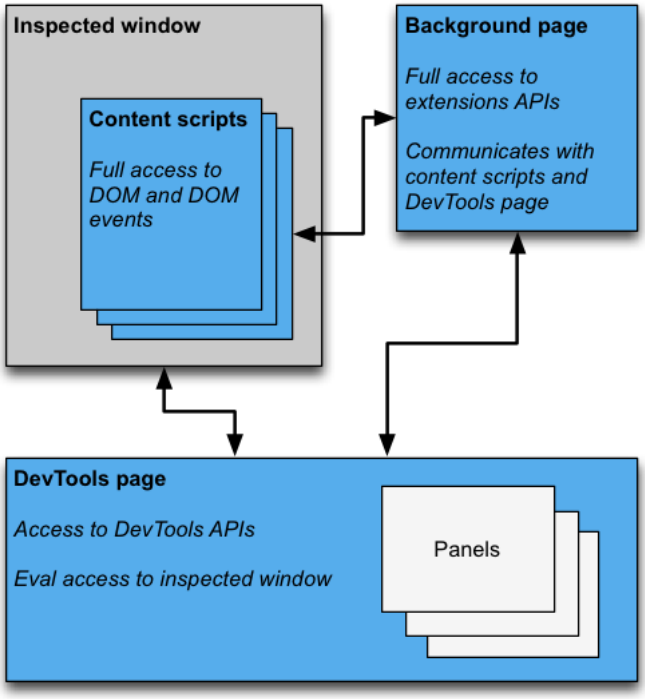
Praca nad implementacją rozszerzenia narzędzi deweloperskich przeglądarki internetowej *Google Chrome*, pozwala poznać ciekawą technikę komunikacji pomiędzy komponentami wchodzącymi w skład wtyczki. Dokumentacja projektowa rozszerzeń przeglądarki, zawiera dokładny opis funkcjonalności oferowanych przez wtyczki oraz sposobu komunikacji komponentów (Listing 21), z której się składa [55].

```
const port = chrome.runtime.connect(null, {name: 'panel'});
const tabId = chrome.devtools.inspectedWindow.tabId;
```

```
const selectorInput = document.getElementById('selector');
const addOperation = document.getElementById('add-operation');
const chooseSelector = document.getElementById('choose-selector');
const executeConfig = document.getElementById('execute-config');
const downloadConfig = document.getElementById('download-config');
const operationDefiner = new OperationDefiner();

operationDefiner.generateOperationsDropdown();

port.onMessage.addListener((message, sender) => {
    switch (message.action) {
        case 'choose selector':
            (selectorInput as HTMLInputElement).value = message.selector;
            break;
    }
});

const post = (message) => {
    message.tabId = tabId;
    port.postMessage(message);
};

addOperation.addEventListener('click', event => {
    operationDefiner.addOperation();
});

executeConfig.addEventListener('click', event => {
    chrome.runtime.sendMessage({
        action: 'execute config',
        target: 'content',
        config: configFileModel,
        tabId
    });
}, false);

chooseSelector.addEventListener('click', event => {
    chrome.runtime.sendMessage({
        action: 'set choose selector event',
```

```
        target: 'content',
        tabId
    });
}, false);

downloadConfig.addEventListener('click', (e) => {
    operationDefiner.downloadConfig((e.target as HTMLButtonElement)).click();
});

post({action: 'init'});
```

Listing 21 Fragment kodu źródłowego skryptu rozszerzenia narzędzi deweloperskich, wraz z uwzględnieniem systemu wiadomości



Rysunek 16 Komunikacja pomiędzy komponentami wtyczki przeglądarki Google Chrome [55]

Z dokumentacji możemy dowiedzieć się że do komunikacji pomiędzy skryptem rozszerzenia (*panel.ts* (Listing 21)) i skryptem działającym jako integralna część aktywnej strony (*content-script.ts* (Listing 22)) konieczny jest skrypt uruchamiany w tle (*background-script.ts* (Listing 23)). Zarówno *content-script* jak i *panel* mają dostęp do skryptu w tle. Każdorazowa komunikacja pomiędzy tymi dwoma komponentami, musi przebiegać za pośrednictwem skryptu *background-script*. Istotnym jest, aby skrypt działający w tle, nasłuchiwał na połączenia utworzone przez każdy ze skryptów i odpowiednio przekierowywał odpowiedzi w celu uzyskania poprawnej komunikacji.

```
const interpreter = new DefaultConfigInterpreterService();
const pathResolver = new RelativePathResolver();
let communicationPort;

const setupPortIfNeeded = () => {
    if (!communicationPort) {
        communicationPort = chrome.runtime.connect(null, {name: 'content'});

        communicationPort.postMessage({action: 'init'});
        communicationPort.onDisconnect.addListener(function () {
            communicationPort = null;
        });
    }
};

chrome.runtime.onMessage.addListener(function (message) {
    switch (message.action) {
        case 'execute config':
            executeConfig(message.config);
            break;
        case 'set choose selector event':
            attachChooseSelectorEvent();
            break;
    }
});
```

Listing 22 Fragment kodu źródłowego pliku content-script.ts z uwzględnieniem komunikacji pomiędzy koponentami

```
let connections = {};

chrome.runtime.onConnect.addListener((port) => {
    const isNotPanelOrContentConnection = port.name !== 'panel' && port.name !== 'content';

    if (isNotPanelOrContentConnection) {
        return;
    }

    const extensionListener = (message) => {
```

```
    const tabId = port.sender.tab ? port.sender.tab.id : message.tabId;

    if (message.action == 'init') {
      connections[tabId] = connections[tabId] ? connections[tabId] : {};
      connections[tabId][port.name] = port;

      return;
    }

    if (message.target) {
      const conn = connections[tabId][message.target];
      conn ? conn.postMessage(message) : void(0);
    }
  };

  port.onMessage.addListener(extensionListener);

  port.onDisconnect.addListener((port) => {
    port.onMessage.removeListener(extensionListener);

    const tabs = Object.keys(connections);
    for (let i = 0, len = tabs.length; i < len; i++) {
      if (connections[tabs[i]][port.name] === port) {
        console.log('background-script onDisconnect connections cleanup',
          {tabId: tabs[i], portName: port.name});
        delete connections[tabs[i]][port.name];

        if (Object.keys(connections[tabs[i]]).length === 0) {
          console.log('background-script onDisconnect remove connection object',
            {tabId: tabs[i]});
          delete connections[tabs[i]];
        }
        break;
      }
    }
  });
});
```

```
chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {
    if (request.target == 'content' && request.tabId) {
        chrome.tabs.sendMessage(request.tabId, request);
    }

    return true;
});
```

Listing 23 Kod źródłowy pliku background-script.ts z uwzględnieniem komunikacji pomiędzy komponentami

Ciekawą implementacją jest również algorytm przeszukiwania BFS (Listing 24), który posłużył do określania dokładnego selektora za pomocą kliknięcia na wybrany element. **Algorytm BFS** [56] to przeszukiwanie struktur grafowych, drzewa bądź wykresu. Zaczyna od korzenia drzewa (lub dowolnego węzła grafu, określanego jako klucz wyszukiwania) i przeszukuje wszystkie sąsiadujące wierzchołki na tej samej głębokości przed przejściem do kolejnego poziomu.

DOM (ang. Document Object Model) jest strukturą *drzewową*, a kliknięty element może stanowić klucz wyszukiwania.

```
export class RelativePathResolver {
    public resolvePath(element): string {
        const stack = [];

        while (element.parentNode != null) {
            let sibCount = 0;
            let sibIndex = 0;

            for (let i = 0; i < element.parentNode.childNodes.length; i++) {
                let sib = element.parentNode.childNodes[i];
                if (sib.nodeName == element.nodeName) {
                    if (sib === element) {
                        sibIndex = sibCount;
                    }
                    sibCount++;
                }
            }

            if (element.hasAttribute('id') && element.id != '') {
                stack.unshift( '#' + element.id);
            } else if (sibCount > 1) {
```

```
            stack.unshift(element.nodeName.toLowerCase() + ':nth-child(' + (sibIndex + 1) + ')');
        } else {
            stack.unshift(element.nodeName.toLowerCase());
        }
        element = element.parentNode;
    }

    return stack.slice(1).join(' > ');
  }
}
```

Listing 24 Implementacja algorytmy przeszukiwania drzewa BFS

5.2. Prezentacja użycia

Prezentacja rozwiązania zaprogramowanego w ramach pracy magisterskiej, została przeprowadzona na nieopublikowanej stronie internetowej, zawierającej spore błędy w związku z implementacją standardu WCAG-2.0.

Implementacja dostosowania kodu źródłowego, została wykonana za pomocą pliku (Listing 26) konfiguracyjnego, wygenerowanego automatycznie przez rozszerzenie przeglądarki *Google Chrome*. Wprowadzone, wymagane poprawki to:

- Tytuł dokumentu
- Język dokumentu
- Nawigacja klawiszowa (w tym *skimming links*)
- Nadanie poprawnych atrybutów *aria*
- Atrybuty *alt*, *title*

Do inicjacji zmian zdefiniowanych w pliku konfiguracyjnym, została załączona biblioteka zaprogramowana w ramach pracy magisterskiej oraz wygenerowany plik konfiguracyjny. Ostatnią wymaganą operacją było wywołanie serwisu interpretującego (Listing 25).

```
<script type="text/javascript" src="./sr-adjuster.config.js"></script>
  <script type="text/javascript" src="./sdk.js"></script>

<script>
  (function initSRConfig() {
      const interpreter = new ScreenReaderHelperSDK.DefaultConfigInterpreterService();
      interpreter.executeConfigFile(window.SRConfig);
    })();
</script>
```

Listing 25 Załączenie i inicjalizacja pliku konfiguracyjnego

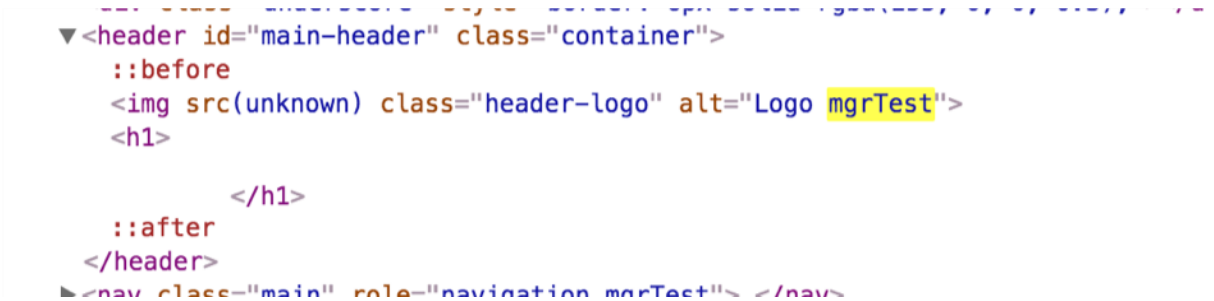
```
{
  'index': 3,
  'selector': 'body > footer > div > div > div:nth-child(1) > p:nth-child(1)',
  'useService': 'skimminLinksService',
  'operation': 'generateAndAppendSkimmingLink',
  'params': {
    'parentNodeSelector': '.carousel-section',
    'text': 'test skip carousel mgrTest',
    'jumpTo': '#main-footer',
    'link': ''
  }
}
```

Listing 26 Fragment wygenerowanego automatycznie za pomocą wtyczki, pliku konfiguracyjnego

Każda operacja zadeklarowana w pliku konfiguracyjnym, zostanie odpowiednio zinterpretowana przez interpreter biblioteki i wykona odpowiadające mu operacje, mające na celu poprawienie jakości wsparcia czytników ekranu strony internetowej. Jako przykład użycia oraz dowód działania rozwiązania zaprezentowanego w ramach pracy magisterskiej, przedstawiony zostanie kod źródłowy strony internetowej z wyszczególnionymi zmianami wprowadzonymi za pomocą pliku konfiguracyjnego.

Rysunek 17 Wynik operacji przeprowadzonych na dokumencie HTML przy pomocy wtyczki – alt text i title

Rysunek 18 Wynik operacji przeprowadzonych na dokumencie HTML przy pomocy wtyczki – aria role



Rysunek 19 Wynik operacji przeprowadzonych na dokumencie HTML przy pomocy wtyczki – alt text

Rysunek 20 Wynik operacji przeprowadzonych na dokumencie HTML przy pomocy wtyczki – alt text i title

Rysunek 21 Wynik operacji przeprowadzonych na dokumencie HTML przy pomocy wtyczki – metatag title

Jak dowodzą powyższe rysunki (Rys.17,18,19,20,21), dzięki funkcjonalnościom zaprogramowanym w ramach projektu pracy magisterskiej, użytkownik z minimalną wiedzą z zakresu inżynierii oprogramowania, jest w stanie wprowadzić zmiany poprawiające jakość wsparcia czytników ekranu przez stronę internetową. Prostota użycia, brak wymogu znajomości technicznych szczegółów implementacyjnych i funkcjonalność umożliwiająca rozwiązywanie najczęściej pojawiających się błędów i zaniedbań związanych z implementacją standardu WCAG-2.0, sprawia, że proponowane rozwiązanie staje się atrakcyjne dla wielu firm i prywatnych użytkowników, chcących łamać bariery stawiane osobom niepełnosprawnym w sieci. Każde z założeń i celów postawionych w poprzednich rozdziałach została spełniona.

Pomimo wysokiej jakości kodu i ogromu możliwości jakie daje biblioteka JavaScript w połączeniu z rozszerzeniem narzędzi deweloperskich przeglądarki *Google Chrome*, istnieje możliwość polepszenia rozwiązania. W kolejnym rozdziale, przedstawione zostaną możliwości rozwoju oraz wnioski płynące z wielu tygodni pracy poświęconej projektowi.

6. Podsumowanie

Kiedy strony internetowe i pokrewne narzędzia są programowane w odpowiedni sposób, z zachowaniem najwyższych standardów jakości kodu, osoby niepełnosprawne są w stanie korzystać z nich w przystępny sposób, za ewentualną pomocą programu asystującego. Jednak wiele witryn, nadal tworzona jest bez uwzględnienia standardów projektowania i budowy sieci, przez co utrudniają bądź uniemożliwiają im korzystanie z nich. Udostępnianie Internetu i jego zasobów, jak największej liczbie osób przynosi korzyści każdemu, bez względu na to czy jest dużą korporacją czy prywatnym użytkownikiem.

Organizacje pracujące nad standardami dostępności sieci, wciąż ulepszają i aktualizują swoje rekomendacje podążając za dynamicznie rozwijającymi się technologiami internetowymi. Naturalną sytuacją, jaka może wystąpić, jest „starzenie się" systemów informatycznych. Najlepszym rozwiązaniem tej sytuacji, jest wymyślenie sposobu na utrzymanie najwyższej jakości standardów, bez konieczności ciągłego wprowadzania modyfikacji i ulepszeń kodu.

Programy powstające każdego dnia, nie są w stanie na bieżąco implementować wszystkich wytycznych, zatem konieczne jest ich systematyczne odświeżanie pod względem jakości kodu.

Generyczne rozwiązanie zaprogramowane na potrzeby pracy magisterskiej, może być uniwersalnym sposobem nieustannego utrzymania wysokiej jakości implementacji standardu WCAG-2.0. Oczywistym jest, że musi ono być w odpowiedni sposób utrzymywane i rozwijane aby mogło spełniać wszystkie założenia najnowszych standardów dostępności sieci.

Niemniej jednak, program przedstawiony jako projekt pracy może stanowić źródło inspiracji, jako odświeżone spojrzenie na sposób wspierania standardów W3C, w związku ze wsparciem czytników ekranu i trzon aplikacji rozwijających możliwości rozwiązania.

6.1. Propozycje rozwoju projektu

Istotną kwestią jest nieustanny rozwój oprogramowania w celu utrzymania pełnej funkcjonalności kodu i wysokiej jakości oferowanych przez niego usług. Projekt pracy magisterskiej stanowi jedynie jądro oprogramowania, które można zbudować na jego podstawie, dzięki inspiracji czerpanej z nowatorskiego podejścia do rozwiązania problemów z implementacja standardu WCAG-2.0. W ramach projektu, możemy wyszczególnić kilka istotnych możliwości rozwoju oprogramowania.

Pierwszym z nich jest rozszerzenie funkcjonalności oferowanych przez bibliotekę. Implementuje ona kilka serwisów, z których każdy jest odpowiedzialny za konkretne funkcje. W trakcie używania, z pewnością pojawią się niestandardowe sytuacje, wymagające specjalnego sposobu użycia. Można wówczas użyć generycznego serwisu. Jeśli jednak sytuacja się powtórzy kilkukrotnie, należałoby rozszerzyć kod źródłowy biblioteki o kolejny, dedykowany serwis.

Plik konfiguracyjny, zawierający zestaw instrukcji możliwych do wykonania przez bibliotekę JavaScript projektu pracy, może w przyszłości okazać się bardzo skomplikowany i trudny w użyciu, wygenerowaniu i zrozumieniu. Jeśli liczba serwisów i oferowanych przez nie funkcjonalności zacznie rosnąć, struktura pliku konfiguracyjnego powinna zostać zmieniona na bardziej ustandaryzowaną, zrozumiałą i dającą możliwość łatwej aktualizacji. Dzięki obiektowo zorientowanemu projektowi biblioteki, wprowadzenie zmian w kreacji pliku konfiguracyjnego, nie powinny stanowić dużego problemu.

Interfejs rozszerzenia narzędzi deweloperskich przeglądarki *Google Chrome*, stanowiącej część pracy magisterskiej, również mógłby zostać poprawiony. Specjalista w dziedzinie *User Experience*, z pewnością zaplanowałby każdy element interfejsu w bardziej przystępny i intuicyjny sposób. W momencie rosnącej popularności wtyczki, jej interfejs powinien zostać przeprojektowany przez osobę wykwalifikowaną.

Layout wtyczki, również jest potencjalnym elementem wymagającym odpowiedniego dostosowania do najnowszych standardów panujących w sieci. Inspiracją do tej zmiany może być popularny w nowoczesnym projektowaniu sieci *Material Design* [60], uznawany przez wielu programistów i projektantów jako standard.

W przyszłości rozszerzenie wraz z biblioteką, mogą zawierać metody wspomagające implementację innych standardów sieci i dawać użytkownikowi możliwość wyboru sposobu implementacji wybranego standardu.

Są to jedynie przykłady możliwości rozwoju oprogramowania zaprezentowanego w ramach pracy magisterskiej. Istnieje wiele więcej zastosowań tego typu rozwiązania i możliwości rozwoju są ograniczone jedynie naszą wyobraźnią. Nowoczesny sposób projektowania aplikacji internetowych i możliwości jakie dają rozszerzenia narzędzi deweloperskich to niekończąca się skarbnica pomysłów.

6.2. Wnioski płynące z przeprowadzonych badań i nabytego doświadczenia

Doświadczenie zebrane podczas pracy nad projektem, pozwoliło na wzbogacenie wiedzy na temat sposobów budowania nowoczesnych aplikacji klienckich, oraz najczęstszych błędów i zaniedbań programistów tychże aplikacji. Wnioski płynące z badań nad dostępnością sieci, pozwalają na zrozumienie ignorancji, wciąż panującej w przestrzeni Internetu.

Większość błędów programistów, popełnianych w związku z implementacją wsparcia czytników ekranu, jest trywialna ale niezwykle istotna dla zapewnienia najwyższej jakości produktu. Aby zrozumieć przyczynę wszystkich zaniedbań, należy zastanowić się kim są programiści budujący sieć.

Od niedawna, zawód programisty, zyskał na popularności. Otworzono wiele szkół programowania i organizacji „przyspieszających" proces kształcenia nowego programisty. Ten zabieg powoduje, że rynek pracy jest zalewany niedoświadczonymi programistami w jeszcze większej skali niż do tej pory. Często zaniedbania i błędy nie wynikają z ignorancji programistów, a z ich niewiedzy.

W społeczności programistycznej, panuje przeświadczenie, że praca programisty aplikacji klienckich (front-end), jest dużo łatwiejsza i wymaga mniejszej dyscypliny i wiedzy niż programistów pozostałych specjalizacji. Do niedawna, wielu doświadczonych pracowników mogłoby zgodzić się z tym odważnym stwierdzeniem. Dlatego wielu początkujących programistów zaczyna swoją karierę w programowaniu warstwy prezentacji aplikacji internetowych i to z pewnością jest jedną z przyczyn powstawania kolejnych barier w sieci. Tymczasem w dzisiejszych czasach, praca programisty front-end została wyniesiona na zupełnie inny poziom. Niestety przeświadczenie osób zaczynających w świecie programowania na temat „łatwości" i istoty tej specjalizacji, pozostaje niezmieniona.

Odpowiednia edukacja programistów, rozpoczynająca się u podstaw programowania i projektowania aplikacji może okazać się zbawienna dla rozwoju sieci. Programiści wyedukowani od podstaw we właściwy sposób, dużo bardziej respektują zasady i standardy projektowania, definiowane przez globalne organizacje.

Największym błędem popełnianym przez osoby odpowiedzialne za tworzenie i publikację witryn internetowych jest tolerancja zaniedbań i brak dyscypliny w związku z panującymi standardami. Należy

dbać o to, aby każdy produkt był dostępny dla każdej osoby w takim samym stopniu, bez względu na płeć, rasę i możliwości fizyczne. Nieograniczone zmysłami.

Internet jest wirtualnym światem, nieustannie budowanym i rozwijającym się z ogromną prędkością. Dbanie o standardy i dostępność każdego z elementów nowego świata, jest kluczowa w procesie jego budowania. Gdyby rzeczywisty świat był zaprojektowany we właściwy sposób u swoich podstaw, a potrzeby osób niepełnosprawnych byłyby respektowane od zawsze, zapewne nie mielibyśmy dziś schodów i problemu z dobudowywaniem wind, tylko jedno rozwiązanie dostępne dla wszystkich. Jako programiści, mamy szansę zbudowania nowego, wirtualnego świata we właściwy sposób. Należy jedynie zadbać o swoje potrzeby i o siebie nawzajem, tak aby pojęcie „bariery w sieci" było tylko legendą opowiadaną z przerażeniem.

Prace cytowane

[1]. W3C. *Accessibility*. https://www.w3.org/standards/webdesign/accessibility/, June 2018

[2]. W3C. *Accessibility in Context*. https://www.w3.org/WAI/fundamentals/accessibility-intro/, June 2018

[3]. ADA.gov. *ADA Standards for Accessible Design*. https://www.ada.gov/2010ADAstandards_index.htm, June 2018

[4]. W3C. *Web Content Accessibility Guidelines*. https://www.w3.org/TR/WCAG20/, June 2018

[5]. Adata.org. *Learn about ADA*. https://adata.org/learn-about-ada, June 2018

[6]. Adata.org. *Definition of Disability under ADA*. https://adata.org/faq/what-definition-disability-under-ada, June 2018

[7]. Adata.org. *Definition of Disability under ADA*. https://adata.org/faq/what-definition-disability-under-ada, June 2018

[8]. Forbes. *First-of-its-kind Trial Goes Plaintiff's Way; Winn-Dixie must update website for the blind*. 13th Jun 2017. https://www.forbes.com/sites/legalnewsline/2017/06/13/first-of-its-kind-trial-goes-plaintiffs-way-winn-dixie-must-update-website-for-the-blind/#2bbfdf641b38, June 2018

[9]. WebAIM. *Screen Reader User Survey*. https://webaim.org/projects/screenreadersurvey5/, June 2018

[10]. WebAIM. *About WebAIM*. https://webaim.org/about/, June 2018

[11]. JAWS. *documentation*. https://www.freedomscientific.com/Products/Blindness/JAWS, June 2018

[12]. ZoomText Magnifier. *Documentation.* https://www.zoomtext.com/products/zoomtext-magnifierreader/, June 2018

[13]. Window-eyes. *Documentation*. https://www.gwmicro.com/, June 2018

[14]. NVDA. *Documentation*. https://www.nvaccess.org/, June 2018

[15]. Voiceover. *Documentation*. https://www.apple.com/lae/accessibility/mac/vision/, June 2018

[16]. ChromeVox. *Documentation*. http://www.chromevox.com/, June 2018

[17]. Cube Group S.A. *SEO*. https://www.semtec.pl/slownik-seo/seo/, June 2018

[18]. WebAIM. *Designing for Screen Reader Compatibility*. https://webaim.org/techniques/screenreader/, June 2018

[19]. Mozilla Developer Network, *tabindex*, https://developer.mozilla.org/pl/docs/Web/HTML/Global_attributes/tabindex, June 2018

[20]. Mozilla Developer Network. *Tabindex*. https://developer.mozilla.org/pl/docs/Web/HTML/Global_attributes/tabindex, June 2018

[21]. W3C. *Aria Landmarks Principles*. https://www.w3.org/TR/wai-aria-practices/examples/landmarks/index.html, June 2018

[22]. WebAIM. *WebAIM's WCAG-2.0 checklist for HTML documents*. https://webaim.org/standards/wcag/checklist, June 2018

[23]. Fundacja Widzialni, *www.widzialni.org*, http://www.widzialni.org/index.php?p=m&idg=mg,103,43

[24]. W3C Techniques for WCAG-2.0. *repeatable components*, https://www.w3.org/TR/WCAG20-TECHS/G61.html, June 2018

[25]. W3C Techniques for WCAG-2.0. *headings hierarchy*, https://www.w3.org/TR/WCAG20-TECHS/H42.html, June 2018

[26]. W3C Techniques for WCAG-2.0. *alt text*, https://www.w3.org/TR/WCAG20-TECHS/H37.html, June 2018

[27]. W3C Techniques for WCAG-2.0. *aria landmarks*. https://www.w3.org/TR/WCAG20-TECHS/ARIA11.html, June 2018

[28]. ORACLE. *A/B testing*. https://www.oracle.com/marketingcloud/resources/ab-testing.html, June 2018

[29]. Wikipedia, *Screen Reader definition*, https://en.wikipedia.org/wiki/Screen_reader

[30]. Coderbust. *Understanding SOLID principles*. https://codeburst.io/understanding-solid-principles-open-closed-principle-e2b588b6491f, June 2018

[31]. Scotch.io *SOLID the first 5 principles of object-oriented design*. https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design, June 2018

[32]. W3C Techniques for WCAG-2.0. *Using title attribute of the frame and iframe elements.* https://www.w3.org/TR/WCAG20-TECHS/H64.html, June 2018

[33]. W3C Techniques for WCAG-2.0. *Using the title attribute to identify form controls when the label element cannot be used*. https://www.w3.org/TR/WCAG20-TECHS/H65.html, June 2018

[34]. W3C Techniques for WCAG-2.0. *Using the title attribute to provide context-sensitive help*. https://www.w3.org/TR/WCAG20-TECHS/H89.html, June 2018

[35]. W3C. *Using null alt text and no title attribute on img entities for images that AT should ignore*. https://www.w3.org/TR/WCAG20-TECHS/H67.html, June 2018

[36]. W3C Techniques for WCAG-2.0. *Using language attributes on html element*. https://www.w3.org/TR/WCAG20-TECHS/H57.html, June 2018

[37]. W3C Techniques for WCAG-2.0. *Using language attribute to identify changes in human language*. https://www.w3.org/TR/WCAG20-TECHS/H58.html, June 2018

[38]. W3Schools. *Meta Tag.* https://www.w3schools.com/tags/tag_meta.asp, June 2018

[39]. W3C Techniques for WCAG-2.0. *Providing title using title element.* https://www.w3.org/TR/WCAG20-TECHS/H25.html, June 2018

[40]. W3C Appendix C. *Understanding Metadata*. https://www.w3.org/TR/UNDERSTANDING-WCAG20/appendixC.html, June 2018

[41]. Dictionary. *Standard*. http://www.dictionary.com/browse/standard, June 2018

[42]. Fundacja Widzialni. *Standard WCAG*. http://wcag20.widzialni.org/standard-wcag,m,mg,148, June 2018

[43]. MDN. *HTML5*. https://developer.mozilla.org/pl/docs/HTML/HTML5, June 2018

[44]. Artem Sapegin. *2017 is the year when front-end developers should go back and master basics, Medium.* https://medium.freecodecamp.org/what-to-learn-in-2017-if-youre-a-frontend-developer-b6cfef46effd, June 2018

[45]. NPM. *Documentation*. https://www.npmjs.com/, June 2018

[46]. Smashingmagazine. *Webpack – a detailed introduction*. https://www.smashingmagazine.com/2017/02/a-detailed-introduction-to-webpack/, June 2018

[47]. Dartlang. *Dart, Typescript and official languages at Google*. https://news.dartlang.org/2017/04/dart-typescript-and-official-languages.html, June 2018

[48]. Github.io. *Karma test runner*. https://karma-runner.github.io/2.0/index.html, June 2018

[49]. NPM. *Documentation*. https://docs.npmjs.com/getting-started/, June 2018

[50]. NPM. *ts-loader*. https://www.npmjs.com/package/ts-loader, June 2018

[51]. NPM. *Tslint*. https://www.npmjs.com/package/tslint, June 2018

[52]. NPM. *Typescript*. https://www.npmjs.com/package/typescript, June 2018

[53]. Github. *Jasmine*. https://github.com/jasmine/jasmine, June 2018

[54]. Developer Chrome. *manifest file – documentation*. https://developer.chrome.com/apps/manifest, June 2018

[55]. Developer Chrome. *Extending DevTools*. https://developer.chrome.com/extensions/devtools, June 2018

[56]. Robin J. Wilson. *Wprowadzenie do teorii grafów*. PWN, Warszawa 1998

[57]. JetBrains. *IntelliJ Idea*. https://www.jetbrains.com/idea/, June 2018

[58]. Robert C. Martin. *Clean Code*. Pearson, 2008

[59]. Robert C. Martin. *Agile. Programowanie zwinne: zasady, wzorce I praktyki zwinnego wytwarzania oprogramowania w C#.* Helion, 2016

[60]. Materia.io. *Material Design.* https://material.io/design/, June 2018

[61]. Scotch.io. *JavaScript transpilers. What they are and why we need them.* https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them, June 2018

[62]. Webpack.js.org. *Tree Shaking*. https://webpack.js.org/guides/tree-shaking/, June 2018

[63]. Mozilla Developer Network. *What is JavaScript*. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript, June 2018

[64]. Mozilla Developer Network. *Introduction*. https://developer.mozilla.org/pl/docs/Web/JavaScript/Guide/Introduction, June 2018

[65]. Wikipedia. *HTML*. https://en.wikipedia.org/wiki/HTML, June 2018

[66]. W3C. *HTML & CSS*. https://www.w3.org/standards/webdesign/htmlcss.html, June 2018

Spis rysunków

Spis listingów