Politechnika Wrocławska

Wydział Informatyki i Zarządzania
Kierunek studiów: Informatyka
Specjalność: Projektowanie systemów informatycznych

Praca dyplomowa – magisterska

Technologie implementacji interfejsów wielojęzykowych systemów webowych

Marcin Łukasz Rosiński

Promotor:
dr hab. inż. Janusz Sobecki, prof. PWr

Recenzent:
dr inż. Maciej Piasecki

Słowa kluczowe:
lokalizacja, internacjonalizacja, wielojęzyczność,
interfejs użytkownika, systemy webowe

Krótkie streszczenie:
Celem pracy jest przegląd i porównanie technologii tworzenia interfejsów wielojęzykowych systemów webowych. Problemy lokalizacji i strategie internacjonalizacji zostaną omówione na bazie rzeczywistego systemu webowego z ogłoszeniami motoryzacyjnymi. Wybrane metody zostaną w nim zaimplementowane, opisane, a następnie zweryfikowane i ocenione.

Wrocław 2018

Streszczenie

Udział osób anglojęzycznych wśród użytkowników sieci Web maleje każdego roku. Globalizacja biznesu, migracje ludności i nowe wymogi prawne zmuszają twórców stron internetowych do rozszerzania funkcjonalności swoich systemów o obsługę wielojęzycznej zawartości. Rosnąca skala zjawiska stała się przyczyną poruszenia tematu implementacji takich mechanizmów w niniejszej pracy magisterskiej. Opisane zostały powszechnie stosowane standardy i zalecenia organizacji normalizacyjnych, a także rozwiązania popularnych problemów pojawiających się podczas tworzenia interfejsów dostosowanych do wymogów różnych kultur. W dalszej części pracy zaprezentowano tabelaryczne zestawienie strategii przygotowania aplikacji umożliwiających jednoczesną obsługę kilku języków. Ich ocena porównawcza została wykonana na podstawie kilkunastu kryteriów zdefiniowanych przez autora pracy. Wybrane techniki wielojęzyczności wdrożono w rzeczywistym systemie z ogłoszeniami motoryzacyjnymi *Motomi*, a następnie zweryfikowano na bazie porównania z podobnymi serwisami. Zaproponowane rozwiązanie okazało się nowoczesnym i skutecznym podejściem do problemu.

Abstract

English-speaking users share in the World Wide Web decreases every year. Business globalization, population migrations and new legal requirements forces website developers to extend their systems' functionality with a multilingual content support. The growing scale of the phenomenon has become the reason for discussing such implementations in this master's thesis. Common standards and recommendations of standardization organizations have been described, as well as solutions to common problems appearing during user's interface adaptation to various cultures requirements. The further part of the paper presents a tabular summary of strategies for preparing applications that support several languages simultaneously. Their comparative analysis was based on a dozen or so criteria defined by the author. Selected multilingualism techniques were implemented in *Motomi*, a real system with automotive advertisements, and then verified on the basis of comparison with similar websites. The proposed solution turned out to be a modern and effective approach to the problem.

Spis treści

1. Wstęp

W latach 90. autorytet językowy David Crystal w swojej pracy *English as a Global Language* [Cry97] opisał język angielski jako ugruntowany na pozycji uniwersalnego, globalnego języka (wł. *lingua franca*), za pomocą którego porozumiewają się różnojęzyczne grupy ludzi na całym świecie. Tamże wskazał jego znaczenie i popularność w ówczesnych mediach, nauce i kulturze, sugerując podobną, dominującą rolę w rozwoju popularyzującego się wtedy Internetu. Geneza standardu WWW sprzyjała tak formułowanym tezom, potwierdzając ich prawdziwość w początkowych latach istnienia globalnej sieci, jednak obecnie, wbrew przewidywaniom, obserwowany jest trend odwrotny do prognozowanego kilkanaście lat temu. Udział anglojęzycznych użytkowników Internetu zmalał i jest szacowany, zależnie od źródła, nawet poniżej 30%.

Opierając się na danych statystycznych dotyczących dynamiki zmian liczebności poszczególnych grup językowych użytkowników Internetu zebranych przez Enrique de Argaez w *Internet World Stats* [Dea17] w ciągu ostatnich 17 lat, można przypuszczać, że w przyszłości użytkownicy anglojęzyczni utracą swoją dominującą pozycję na rzecz użytkowników m.in. chińskojęzycznych. Na lingwistyczny obraz globalnej sieci istotny wpływ będą mieć również języki takie jak arabski czy rosyjski, które w minionych latach charakteryzowały się największym przyrostem liczby użytkowników, kilkukrotnie przewyższając język angielski. Jako przyczyny tych zmian Michael Oustinoff w artykule *English Won't Be the Internet's Lingua Franca* [Ous12] wymienia coraz lepszą dostępność łącz internetowych, a także czynniki ekonomiczne, geopolityczne i demograficzne. Wobec pojawiania się kolejnych języków uniwersalnych funkcjonujących równolegle na tym samym obszarze, zauważa on wzrost znaczenia treści tłumaczonych (również automatycznie) i sugeruje odejście od dotychczasowego zwyczaju tworzenia monolitycznych treści w języku angielskim na rzecz **wielojęzyczności** (ang. multilingualism).

Opisane powyżej przemiany mają szczególny wpływ na kierunek rozwoju podejmowany przez współczesne przedsiębiorstwa poszukujące nowych rynków zbytu w Internecie. Kanclerz Republiki Federalnej Niemiec Willy Brandt miał powiedzieć [Ede10]: *„If I'm selling to you, I speak your language. If I'm buying, dann müssen Sie Deutsch sprechen"* (z ang. gdy sprzedaję mówię w twoim języku, gdy kupuję musisz mówić po niemiecku), wskazując wyższą konkurencyjność handlu prowadzonego z kontrahentem w jego ojczystym języku, nawet jeśli komunikacja byłaby możliwa w pewnym wspólnym języku handlowym. Aktualność tych słów potwierdza Ingela Bel Habib w opracowaniu *Multilingual Skills provide Export Benefits and Better Access to New Emerging Markets* [Hab11], w którym pokazuje pozytywne ekonomicznie skutki otwarcia językowego małych i średnich przedsiębiorstw niemieckich oraz francuskich, zestawiając je ze szwedzkimi, które z powodu zamknięcia na swoją grupę językową straciły najwięcej potencjalnych klientów. Wiążąc sukces na rynkach międzynarodowych m.in. z dostępnością wielojęzycznej wersji strony webowej, należy oczekiwać rosnącego zapotrzebowania biznesu na takie rozwiązania.

Poruszone kwestie dotyczą nie tylko sfery działalności gospodarczej. Zjawiska demograficzne zachodzące na świecie, w połączeniu ze wzrostem znaczenia Internetu jako podstawowego (coraz częściej jedynego) źródła informacji [Usc15], powodują że wielojęzykowe interfejsy systemów webowych stają się istotne także dla administracji państwowej, szkolnictwa wyższego czy turystyki. Jak podaje *Rocznik statystyczny Rzeczypospolitej Polskiej 2016* [Gus16], w ciągu ostatnich 10 lat udział obcokrajowców

w ogóle studentów uczelni wyższych w Polsce wzrósł prawie dziesięciokrotnie, a według oceny instytucji związanych z internacjonalizacją szkolnictwa trend ten powinien być stymulowany i utrzymany [Cza17]. Szybki dostęp do oficjalnych informacji w rozumianym języku w oczywisty sposób zwiększa atrakcyjność uczelni na tle innych, ułatwia podjęcie studiów, odciąża pracowników administracyjnych i zmniejsza ryzyko nieporozumień. W analogicznej sytuacji znajduje się polski rynek pracy, który w 2015 roku zasilało już niemal milion obcokrajowców [Nbp16]. Zapewnienie im wiarygodnego źródła informacji urzędowych w kilku językach to problem typowy szczególnie dla społeczeństw historycznie niejednorodnych (np. Belgia) lub charakteryzujących się wysokim udziałem imigrantów (np. Wielka Brytania). W przypadku Unii Europejskiej tworzenie wielojęzycznych interfejsów stron internetowych jest nie tylko kwestią dobrej praktyki usprawniającej funkcję informacyjną i procesy administracyjne, ale także regulowanym prawnie obowiązkiem nałożonym na własne urzędy [Som10].

Zważywszy na opisane powyżej szerokie znaczenie problemu wielojęzyczności dla współczesnych interfejsów, w niniejszej pracy postanowiono poruszyć temat implementacji stosownych mechanizmów w istniejącej aplikacji internetowej. W tym celu podjęto współpracę z firmą *Pixel Office* z Chojnic[1], dysponującą rzeczywistym systemem webowym *Motomi*[2], który umożliwia swoim użytkownikom zamieszczanie w Internecie ogłoszeń motoryzacyjnych. System został przygotowany w języku polskim i w dalszej części pracy zostanie zbadany pod kątem przystosowania do komunikacji w wielu językach. Na jego przykładzie zostaną omówione i porównane różne techniki implementacji wielojęzykowych interfejsów webowych. Wybrana metoda zostanie zaimplementowana w systemie, szczegółowo opisana, a następnie zweryfikowana i oceniona.

W rozdziale 2 wyjaśnione zostaną główne terminy i zalecenia związane z tworzeniem wielojęzykowych interfejsów użytkownika. Omówione zostaną też powszechnie wykorzystywane standardy ułatwiające realizację tego zadania. W rozdziale 3 przedstawione zostaną problemy występujące podczas dostosowywania systemu do potrzeb konkretnego języka. Zaprezentowane zostaną również techniki pozwalające na zniwelowanie każdej z tych barier. W rozdziale 4 opisane zostaną strategie umożliwiające przygotowanie systemu do równoległej obsługi wielu różnych języków. Metody te zostaną ze sobą porównane oraz ocenione ze względu na kilkanaście kryteriów. W rozdziale 5 ukazane zostanie wdrożenie wybranych technik wielojęzyczności w będącym przedmiotem badań systemie *Motomi*. Rozwiązanie zostanie porównane z podobnymi systemami webowymi i zweryfikowane. W zakończeniu zawarte zostanie podsumowanie efektów wykonanej pracy.

1 Zob. http://pixeloffice.pl/.
2 Zob. https://motomi.pl/.

2. Podstawowe zagadnienia i standardy

W tym rozdziale wyjaśnione zostaną podstawowe pojęcia z jakimi można spotkać się podczas tworzenia wielojęzykowych interfejsów użytkownika. Opisane zostaną również najpopularniejsze standardy i wytyczne usprawniające pracę twórców takich aplikacji.

2.1. Wielojęzyczność a internacjonalizacja i lokalizacja

Terminy wielojęzyczności, internacjonalizacji i lokalizacji bywają błędnie używane w języku potocznym jako zamienne, co zapewne wynika z ich ścisłego powiązania. Nie są one jednak równoważne i opisują inne aspekty tworzenia systemu komunikującego się z użytkownikiem w kilku językach. Organizacja *World Wide Web Consortium* (skr. W3C) zajmująca się przygotowywaniem standardów dla sieci Web proponuje definicje przedstawione poniżej [Ish15].

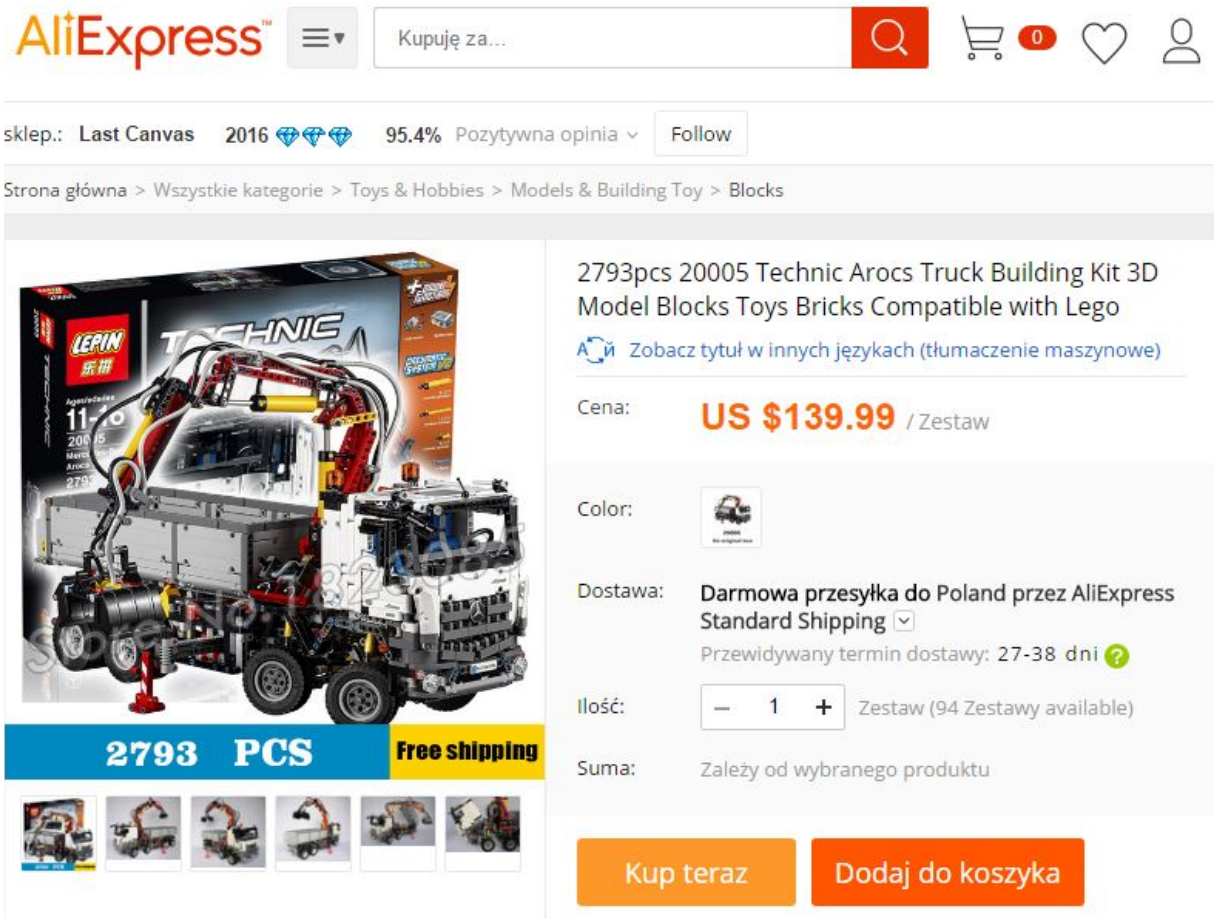
2.1.1. Lokalizacja

Przez lokalizację (ang. *localization*) rozumieć należy dostosowanie interfejsu aplikacji (oraz towarzyszących jej dokumentów) do określonych wymagań językowych i kulturowych, tak aby uwzględnione zostały wymogi konkretnej grupy docelowej lub rynku na którym produkt ma funkcjonować. W szczególności definicję tę spełnia każdy system, nawet jeśli powstaje z myślą o działaniu tylko w jednym regionie. Lokalizacja może obejmować m.in.:

1. odpowiednie kodowanie znaków narodowych i wybór fontów (podstawowy alfabet łaciński składa się z 26 znaków, natomiast japoński *kanji* – z kilku tysięcy),
2. tłumaczenie tekstu (także jego kierunek i sposób wprowadzania znaków specjalnych),
3. sposób sortowania (reguły stosowane dla pisma fonetycznego nie działają dla pisma ideograficznego),
4. format danych numerycznych (np. rodzaj separatora dziesiętnego, grupowanie cyfr),
5. reprezentację daty i czasu (np. kolejność podawania dni, miesięcy i lat, użycie zegara 12- lub 24-godzinnego),
6. walutę (klienci sklepu internetowego najchętniej będą płacili we własnej),
7. symbole i kolory (np. kolor biały kojarzony w Europie z niewinnością i czystością, w Chinach jest kolorem żałobnym),
8. grafiki (w różnych kulturach pewne idee mogą być niezrozumiałe lub zostać zinterpretowane jako obraźliwe),
9. rozkład elementów (np. inne położenie głównych elementów nawigacyjnych, gdy zawartość jest czytana od lewej do prawej, a inne gdy od prawej do lewej),
10. wymogi prawne (np. informacja o użyciu ciasteczek na terenie Unii Europejskiej).

Można zauważyć, że obszary te swoje odzwierciedlenie znajdują w heurystykach Jakoba Nielsena opisujących zasady interakcji człowiek-komputer. Przywołać należy zwłaszcza zasadę drugą, która mówi *„zachowaj zgodność pomiędzy systemem a rzeczywistością”* [Nie93]. Reguła ta obejmuje komunikację z użytkownikiem w zrozumiały dla niego sposób, czego podstawą jest przetłumaczony tekst, odpowiedni format danych czy wreszcie czytelna symbolika.

Lokalizacja może dotyczyć także innych części systemu niż wymienione powyżej, ale w praktyce, zwłaszcza gdy budowana jest na bazie innej, wcześniej istniejącej, ogranicza się do zastosowania odpowiedniego kodowania znaków narodowych i przetłumaczenia podstawowych tekstów. Dobrym przykładem może być widoczna na Rys. 2.1 polska lokalizacja sklepu internetowego *AliExpress*. Jego twórcy przetłumaczyli najważniejsze napisy (niektóre tylko częściowo, np. *94 Zestawy available*), proponują też użycie automatycznego tłumaczenia tytułu, jednak wiele tekstów (w tym nawigacja między kategoriami produktów) pozostało w języku angielskim. Zdjęcie przedmiotu również posiada nieprzetłumaczone napisy – pokazuje to, że o ile autorzy serwisu mają wpływ na lokalizację stałych elementów witryny, o tyle nie mają wpływu na treści zamieszczane przez swoich użytkowników. W takich przypadkach można stosować, w miarę możliwości technicznych, mechanizmy tłumaczenia maszynowego. Brakuje także możliwości płacenia w złotych lub przynajmniej orientacyjnego przeliczenia ceny z dolarów amerykańskich. Takie minimalistyczne podejście do lokalizacji, mimo widocznych niedoskonałości, znacząco zwiększyło dostępność systemu dla polskich użytkowników, którzy wcześniej zmuszeni byli do samodzielnego tłumaczenia interfejsu z języka chińskiego.



Rys. 2.1. Lokalizacja serwisu AliExpress.

2.1.2. Internacjonalizacja

Internacjonalizacja (ang. *internationalization*), nazywana również globalizacją, odnosi się do sposobu projektowania i wytwarzania aplikacji w sposób, który umożliwi łatwą *lokalizację* dla wybranych kultur, regionów lub języków. W odróżnieniu od lokalizacji

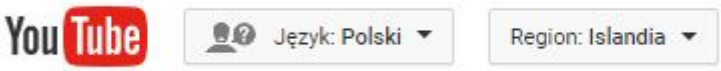
dotyczy zatem nie tyle przygotowania odpowiednich treści i sposobu ich prezentacji, co mechanizmów pozwalających oddzielić te charakterystyczne cechy od właściwego kodu źródłowego. Internacjonalizacja może obejmować m.in.:

1. uniwersalne kodowanie znaków narodowych (zamiast specyficznego),
2. przechowywanie elementów podlegających lokalizacji w zewnętrznej bazie (np. przez zapisywanie w interfejsie samych wskaźników do właściwych napisów),
3. stosowanie szablonów umożliwiających formatowanie danych (np. przez użycie identyfikatorów pozwalających na zmianę kolejności zmiennych w zdaniu lub zastosowanie kursywy),
4. obsługę funkcji specyficznych dla wybranych lokalizacji (np. dodanie znaczników identyfikujących język, na podstawie których przeglądarka automatycznie ustali kierunek tekstu),
5. obsługę ustawień regionalnych (np. kalendarz uwzględniający święta narodowe).

Należy zwrócić uwagę, że najlepszym momentem na wprowadzenie internacjonalizacji jest początek projektu. Zmiany w zlokalizowanym systemie będą dużo bardziej czasochłonne, choćby z uwagi na konieczność ręcznego wydzielenia wszystkich dotychczas wprowadzonych napisów poza aplikację czy głęboko zakodowany sposób przetwarzania czasu. W zamian kolejne lokalizacje będą mogły być przeprowadzane szybciej, w dużym stopniu przez samego tłumacza, bez konieczności posiadania kompetencji pozwalających na ingerencję w kod źródłowy, a aktualizacja istniejących języków będzie możliwa bez ponownego wdrażania systemu na serwer.

2.1.3. Wielojęzyczność

Przez wielojęzyczność systemu webowego należy rozumieć połączenie lokalizacji i internacjonalizacji w sposób, który umożliwia **jednoczesne** korzystanie z aplikacji przez użytkowników używających różnych ustawień regionalnych [Hil02, s. 1-2]. Witryna taka umożliwia wybór języka lub stara się wykryć go automatycznie i na tej podstawie dopasowuje swoją treść.



Rys. 2.2. Wielojęzyczność w serwisie YouTube.

Warto zauważyć, że twórcy dzisiejszych systemów webowych coraz częściej odchodzą od utożsamiania języka z konkretnym regionem świata. Nadal zakłada się pewne domyślne powiązanie, istnieje jednak możliwość jego zmiany. W ten sposób prezentowana treść może oznaczać współistnienie polskich elementów interfejsu i islandzkich treści proponowanych użytkownikowi. Przykładem niech będzie umożliwiający przesyłanie strumieniowe wideo serwis *YouTube*, którego kontrolki widoczne na Rys. 2.2 pozwalają na niezależne sterowanie wspomnianymi parametrami.

2.1.4. Popularne skróty

W żargonie technicznym często można spotkać enigmatyczne skróty: *l10n*, *i18n* oraz *m17n*. Są to numeronimy oznaczające odpowiednio: lokalizację, internacjonalizację

i wielojęzyczność. Powstały pod koniec lat 80. przez zastąpienie środkowej części swoich anglojęzycznych odpowiedników liczbą liter wewnątrz wyrazu. Więcej na temat ich historii można przeczytać w artykule *Origin of the Abbreviation i18n* autorstwa Texa Texina [Tex10].

2.2. Wytyczne organizacji W3C

Mnogość pomysłów na techniczną realizację założeń wielojęzyczności opisywanych w literaturze powoduje, że początkowe rozważania najlepiej oprzeć na wytycznych organizacji W3C mającej największy wpływ na standardy wprowadzane do sieci Web. Na swoich stronach opublikowała zbiór podstawowych dobrych praktyk [Wor16], w skład których wchodzą m.in.:

1. unikanie czystego tekstu nieujętego w żadne znaczniki HTML pozwalające na dodanie metadanych opisujących sposób prezentacji wielojęzycznego tekstu (zalecane jest użycie przynajmniej znaczników *span*),
2. oznaczanie dokumentów i fragmentów tekstu atrybutem HTML *lang* definiującym wariant językowy (dla automatycznego przetwarzania, wyszukiwania, syntezy mowy, itd.), stosując przy tym format tagów językowych określony w specyfikacji *BCP 47* (więcej o specyfikacji można przeczytać w [Phi09]),
3. oznaczanie tekstu atrybutem HTML *dir* definiującym jego kierunek,
4. zapisywanie znaków przy użyciu standardu zgodnego z Unicode (więcej w dalszej części rozdziału),
5. wykorzystywanie znaczników semantycznych zamiast znaczników wizualnych (np. wzmocnienia *strong* zamiast pogrubienia *b*, które może być niedostępne dla niektórych języków),
6. uwzględnienie dodatkowej przestrzeni dla znaków wyższych niż łacińskie i różnej długości napisów (kontenery zawierające tekst muszą zmieniać swój rozmiar dynamicznie w taki sposób, żeby zachować czytelność tekstu, a jednocześnie nie zniszczyć układu wizualnego),
7. dodawanie opisu zawartości do zasobów multimedialnych (informacja wielomodalna może być prostsza do przygotowania niż tworzenie nowych wersji filmów czy ilustracji, a wystarczająca do zrozumienia treści witryny),
8. bazowanie na standardzie *Universal Time Coordinated* (skr. UTC) w implementacji daty i czasu.

Wytyczne te są dość ubogie, ale stanowią dobry punkt wyjścia dla planowania architektury aplikacji i pozwolą uniknąć podstawowych błędów popełnianych podczas internacjonalizacji interfejsu.

2.3. Normy ISO

Budując system wielojęzyczny, w wielu miejscach kodu źródłowego zachodzi konieczność określenia języka lub regionu. Aby uniknąć problemu kompatybilności z innymi systemami oraz bibliotekami należy posługiwać się kodami publikowanymi przez Międzynarodową Organizację Normalizacyjną (ang. *International Organization for Standardization*, znana jako ISO). Zrzesza ona ponad 160 narodowych komitetów normalizacyjnych (Polskę reprezentuje Polski Komitet Normalizacyjny), a ustalane przez nią

standardy są szeroko akceptowane na świecie i w Internecie [Iso17].

2.3.1. ISO 639 – kody języków

ISO 639 jest międzynarodowym standardem zapisu kodów języków. Większość języków używanych na świecie jest reprezentowana przez kody dwuliterowe, ale kolejne wersje standardu rozszerzały go też o kody trzy- i czteroliterowe (głównie dla określania całych grup i rodzin językowych lub języków wymarłych i antycznych) [Iso10]. Dla języka polskiego są to odpowiednio oznaczenia *pl* i *pol*. Na potrzeby internacjonalizacji stosuje się najczęściej kody dwuliterowe, podczas gdy dłuższe warianty pozostają domeną księgoznawstwa. Aktualną listę dostępnych kodów można znaleźć na stronach odpowiedzialnej za ich utrzymanie Biblioteki Kongresu (ang. *Library of Congress*)[1].

2.3.2. ISO 3166 – kody krajów

Analogicznie, do zapisu rozpoznawalnych na całym świecie oznaczeń krajów (także terytoriów zależnych, specjalnych obszarów geograficznych i jednostek administracyjnych) należy używać standardu ISO 3166. Warto podkreślić, że definiuje on jedynie uniwersalne kody, a nie pełne nazwy, ponieważ te różnią się w zależności od języka [Iso13a]. Kody te zastosowanie mają np. w domenach najwyższego poziomu. Dla Polski są to ponownie oznaczenia *PL* i *POL* (tym razem pisane wielką literą, choć nie jest to bezwzględnie wymagane). Aktualna lista kodów jest dostępna odpłatnie w formie papierowej lub przez specjalną wyszukiwarkę organizacji: *Online Browsing Platform*[2].

2.4. Kodowanie znaków

W ubiegłym wieku użytkownicy sieci Internet często spotykali się z dokumentami i stronami, na których występowały tzw. *krzaczki*, czyli z pozoru przypadkowe znaki widoczne w miejscu poszczególnych liter. W przypadku stron polskojęzycznych podmienione były zwykle litery diakrytyczne, co utrudniało, ale nie uniemożliwiało zrozumienie tekstu. W gorszej sytuacji byli internauci nieposługujący się alfabetem łacińskim, ponieważ w skrajnych przypadkach, bez podjęcia dodatkowych kroków, takich jak konwersja i użycie specjalnych fontów, treść strony mogła być dla nich całkowicie nieczytelna. Z przyczyn historycznych i z racji swoich korzeni, w Internecie najlepiej obsługiwany był język angielski. Działo się tak, ponieważ nie istniał uniwersalny system zapisu znaków pokrywający wszystkie żywe języki świata i stosowany przez wszystkie systemy operacyjne.

Obecnie możliwe jest przeglądanie obcojęzycznych witryn i wymienianie się dokumentami pochodzącymi z różnych stron świata bez żadnych dodatkowych zabiegów umożliwiających ich prawidłowe odczytanie. Wiele z wcześniejszych problemów zostało rozwiązane dzięki wprowadzeniu standardu *Unicode*, który zostanie omówiony w dalszej części tego rozdziału. Wyczerpująco o historii i sposobach zapisu tekstu w komputerach przeczytać można w dwóch pierwszych rozdziałach pozycji *Fonts & Encodings* autorstwa Yannisa Haralambousa [Har07].

2.4.1. Zestaw znaków a kodowanie znaków

Dla prawidłowego zrozumienia sposobów kodowania znaków opisanych w kolejnych punktach, należy zdefiniować towarzyszące im w dokumentacji podstawowe terminy.

1 Zob. http://www.loc.gov/standards/iso639-2/php/code_list.php.
2 Zob. https://www.iso.org/obp/ui/, zakładka C*ountry codes*.

Uwspółcześniając definicje zaproponowane przez [Mac80] otrzymujemy:

1. glif (ang. *glyph*) to kształt przedstawiający symbol alfanumeryczny lub specjalny,
2. znak (ang. *character*) to wzorzec bitowy o nadanym znaczeniu graficznym lub sterującym,
3. zestaw znaków (ang. *character set*) to kolekcja znaków możliwych do użycia w dokumencie (np. Unicode),
4. kodowanie (ang. *encoding*) to sposób zapisu w pamięci znaków z zestawu znaków, w taki sposób, że jeden kod (pozycja) oznacza jeden znak (np. UTF-8),
5. strona kodowa (ang. *code page*) to sposób przedstawienia kodowania w formie tabeli,
6. font to kolekcja glifów o spójnym kroju pisma, będąca graficzną reprezentacją zestawu znaków (np. Times New Roman).

2.4.2. ASCII

Mające swoje początki w latach 60. ASCII (ang. *American Standard Code for Information Interchange*) nie było pierwszym sposobem kodowania znaków w systemach komputerowych, jednak zyskało największa popularność i na wiele lat zdominowało Internet. W sieci Web funkcjonuje do dziś, a wiele późniejszych standardów było tylko jego modyfikacją lub rozszerzeniem. Miejsca najpopularniejszego kodowania znaków na stronach internetowych ustąpiło kodowaniu UTF-8 (notabene kompatybilnemu z ASCII) dopiero na przełomie lat 2007/2008 [Dav08].

ASCII jest kodowaniem 7-bitowym, które pozwala na zapisanie 128 znaków. Specyfika działania sprzętu komputerowego z czasów wprowadzania standardu spowodowała, że nie wszystkie znaki są graficznymi – pierwsze 32 pozycje zawierają znaki sterujące [np. formatowaniem, jak powrót karetki (ang. *carriage return*, skr. CR*)* czy komunikacją, jak koniec transmisji (ang. *end of transmission*, skr. EOT)]. Na dalszych pozycjach, oprócz liter alfabetu łacińskiego (w wariancie dużym i małym), znajdują się cyfry, znaki interpunkcyjne charakterystyczne dla krajów anglosaskich i podstawowe symbole matematyczne (takie jak operatory, nawiasy). Stronę kodową ASCII w wersji z 1967 roku przedstawiono poniżej w Tab. 2.1.

Tab. 2.1. Strona kodowa ASCII-1967.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DCI	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Źródło: opracowanie własne na podstawie [Har07, ASCII].

Pozycja danego znaku w powyższej tabeli określona jest przez sklejenie numeru wiersza i kolumny w kodzie szesnastkowym, np. znak zapytania znajduje się na pozycji $3F_{(16)}$. Na jej podstawie można zauważyć, że w kodowaniu ASCII brakuje wielu charakterystycznych znaków narodowych. Akcenty wprowadza się przez specjalne kombinacje znaków graficznych i kontrolnych (np. kombinacja *a BS ‘* daje *á*), jednak rozwiązanie to nie jest ani wygodne, ani nie pokrywa w pełni bogactwa ortograficznego języków europejskich (nie wspominając o azjatyckich, dla których możliwość zapisania 128 znaków jest niewystarczająca).

Na fali popularności ASCII powstawały jego zlokalizowane wersje, które różniły się znakami na pojedynczych pozycjach, np. stosowany w Wielkiej Brytanii *BS4730*, który zmieniał symbol dolara amerykańskiego na pozycji $24_{(16)}$ na symbol funta szterlinga. Można się tylko domyślać do jakich problemów i nieporozumień dochodziło, gdy plik z cennikiem towarów utworzony na komputerze amerykańskim był otwierany na brytyjskim, a jedyną różnicą była waluta. Ostatnia wersja ASCII została opublikowana w 1991 roku i została zatwierdzona jako norma *ISO/IEC 646:1991* [Iso91].

2.4.3. ISO 8859

Rozwiązaniem problemów ze znakami diakrytycznymi i lokalizowanymi wersjami ASCII miała być rodzina kodowań ISO 8859 zaproponowana w roku 1987. Są to kodowania 8-bitowe, które pozwalają na zapisanie 256 znaków. Dla zachowania wstecznej kompatybilności kody te pokrywają się z ASCII na pierwszych 128 pozycjach, można je zatem traktować jako swoiste rozszerzenie dotychczasowego standardu. Zestaw dodatkowych znaków jest zależny od wybranego wariantu kodowania [Gou09]: ISO 8859-1 (Europa Zachodnia), ISO 8859-2 (Europa Środkowa i Wschodnia), ISO 8859-3 (Europa Południowa), ISO 8859-4 (Europa Północna), ISO 8859-5 (cyrylica), ISO 8859-6 (arabski), ISO 8859-7 (grecki), ISO 8859-8 (hebrajski), ISO 8859-9 (turecki), ISO 8859-10 (języki nordyckie), ISO 8859-11 (tajski), ISO 8859-12 (nieużywany), ISO 8859-13 (języki bałtyckie), ISO 8859-14 (języki celtyckie), ISO 8859-15 (zmodyfikowany ISO 8859-1 ze znakiem euro), ISO 8859-16 (zmodyfikowany ISO 8859-2 ze znakiem euro). Słusznie można domniemać, że niektóre z przytoczonych kodowań będą pokrywać te same języki (np. ISO 8859-4, ISO 8859-10 i ISO 8859-13, które są dedykowane dla krajów bałtyckich). Przyczyną jest dopracowywanie standardu na przestrzeni lat z zachowaniem kompatybilności wstecznej (tworzono nowy wariant zamiast modyfikować istniejący).

Najważniejszym kodowaniem z tej rodziny jest ISO 8859-1, ponieważ przyjęto, że wariant ten będzie domyślnym dla Unixa oraz stron webowych tworzonych zgodnie ze standardem HTML4 [Tit05]. Warto wspomnieć, że w systemach Windows stosowany jest jego bliźniaczy odpowiednik, czyli Windows-1252 (określany również jako CP1252). Jedyną zmianą jaką wprowadził Microsoft było dodanie kilku dodatkowych znaków graficznych w miejsce nieużywanych znaków sterujących [Tex11]. Jeżeli nie będą one wykorzystywane, to obu kodowań można używać wymiennie.

Jawne określenie regionu połączone z zestawem dodatkowych znaków miało rozwiązać problem powstawania wielu lokalnych standardów bazujących na ASCII, brakujących znaków i niewygodnego sposobu wprowadzania akcentów. W praktyce pełna obsługa wszystkich języków nadal nie była możliwa, a 256 dostępnych do zakodowania pozycji pozostawało poza skalą potrzeb wielu języków azjatyckich. Istotnym problemem z perspektywy wielojęzykowych systemów webowych było w dalszym ciągu nierozwiązane zagadnienie prostego użycia znaków z różnych języków w ramach jednego dokumentu.

2.4.4. ISO 2022 (JIS, GB i KS)

Wobec widocznych ograniczeń jakie na rozmiar zestawu znaków narzucały kodowania ASCII i ISO 8859, opracowany został kolejny standard, ISO 2022 (ECMA-35), który pozwala na użycie większej liczby bajtów do zakodowania pojedynczego znaku. Jego specyfikacja pozwala on na uzyskanie 94^n lub 96^n pozycji, gdzie n – liczba bajtów użytych na jeden znak. Wewnętrzna organizacja zestawu znaków w tymże kodowaniu polega na podzieleniu go na kilka grup, między którymi można się przełączać używając odpowiedniej sekwencji znaków sterujących [Ecm94]. Konieczność utrzymywania bieżącego stanu dla każdego bloku tekstu spowalnia jednak i komplikuje operacje na tekście, takie jak wyszukiwanie czy wstawianie.

W oparciu o ISO 2022 powstały głównie kodowania języków dalekiego wschodu, które funkcjonowały równolegle do ISO 8859 (ten skupiał się przede wszystkim na językach świata łacińskiego), takie jak: ISO 2022-JP, ISO 2022-CN, ISO 2022-KR. Opierają się one na mieszance czerpiącej znaki z różnych zestawów, m.in.: japońskich JIS X 0201 (ang. *Japanese Industrial Standard*), chińskich GB 2312 (chiń. *Guojia Biaozhun*, oznaczający standard narodowy), koreańskich KS X 1001 (ang. *Korean Standard*), itd. Więcej na temat historii rozwoju kodowania języków dalekowschodnich przed erą Unicode można przeczytać w sekcji *The Far East*, w rozdziale *Before Unicode* z pozycji [Har07].

Standard ISO 2022, podobnie jak ISO 8859, nie zapewnia wygodnego mechanizmu wielojęzyczności w ramach jednego dokumentu ani nie rozwiązuje problemu współistnienia mnogich, niekompatybilnych ze sobą sposobów kodowania. Pozwolił jednak azjatyckim użytkownikom komputerów na komunikację w językach narodowych, bez konieczności upraszczania jej do niedokładnych form kompatybilnych ze standardami tworzonymi w duchu innego kręgu kulturowego.

2.4.5. UTF-8 (Unicode)

Potrzeba wprowadzenia spójnego i uniwersalnego zestawu znaków oraz jego kodowania spowodowała założenie, z inicjatywy firm Apple i Xerox, konsorcjum Unicode, które miało ten ambitny cel osiągnąć. Równocześnie organizacja ISO pracowała nad standardem ISO 10646, który miał rozwiązać ten sam problem. W wyniku współpracy między oboma projektami, w roku 1993 wydano pierwszą wersję Unicode – uniwersalnego zestawu znaków. W momencie pisania tej pracy Unicode 10.0.0 obejmuje 136690 znaków pochodzących ze 139 rodzajów pisma, a jego kolejne wersje są wydawane regularnie każdego roku [Uni17, s. 3]. Problem wieży Babel został po latach z sukcesem rozwiązany, pozwalając na wielojęzykowe dokumenty i strony webowe bez skomplikowanej implementacji obsługi języka, konieczności śledzenia i przełączania między różnymi kodowaniami czy konwersji.

Zapis znaków Unicode jest możliwy za pomocą jednego z kilku zdefiniowanych w standardzie sposobów kodowania. Najlepszym wyborem dla twórcy wielojęzykowego systemu webowego jest obecnie UTF-8 (ang. *Unicode Transformation Format*). Dwoma głównymi argumentami stojącymi za taką rekomendacją jest jego obecna penetracja sieci Web (w maju 2018 korzystało z niego 91,3% wszystkich stron internetowych o znanym kodowaniu [Web18]) oraz zalecanie UTF-8 jako domyślnego kodowania znaków przez specyfikacje języków XML 1.0 i HTML5 [Ish14a].

UTF-8 jest kodowaniem 8-bitowym o zmiennej długości od 1 do 6 oktetów (w praktyce używane jest do 4 oktetów, co pozwala na zapisanie wszystkich obecnie istniejących znaków Unicode). W postaci pojedynczego oktetu pokrywa się z ASCII, co zapewnia

kompatybilność z wcześniejszym standardem i ułatwia migrację na UTF-8 [Jur15]. W postaci kilku oktetów, pierwszy informuje o liczbie wykorzystanych bajtów (równą liczbie najwyższych bitów w stanie wysokim), a kolejne oznaczają kontynuację za pomocą najwyższych bitów ustawionych na *10*. Pozostałe bity (oznaczone *x*) kodują znak:

```
0xxxxxxx
110xxxxx 10xxxxxx
1110xxxx 10xxxxxx 10xxxxxx
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Przykładowo, chcąc zakodować literę *ń* (nazwa Unicode: *Latin Small Letter N With Acute*) o numerze *U+0144* można zauważyć, że potrzeba do tego co najmniej 9 wolnych bitów kodowych *x*, ponieważ $144_{(16)}=\mathbf{101000100}_{(2)}$. Używając pojedynczego oktetu dostępne jest ich tylko 7, należy zatem użyć dwóch oktetów, które kodują do 11 bitów (niewykorzystane bity uzupełnia się zerami). Otrzymane kodowanie to $11000\mathbf{101}\ 10\mathbf{000100}_{(2)}=C584_{(16)}$.

Dzięki przyjętemu formatowi zapisu uzyskano łatwość identyfikacji początku i końca znaku w strumieniu danych, zachowując jednocześnie zgodność z ASCII i unikając problemu z oktetem *00000000*, kończącym w wielu systemach transmisję (ang. *null terminated string function*). Wadą takiego zapisu, dotyczącą głównie języków azjatyckich, jest pewna nadmiarowość, np. w stosunku do bliźniaczego UTF-16, który ten sam znak koduje na 2 bajtach zamiast na 3, itp. Więcej o Unicode i sposobach jego kodowania przeczytać można w opracowaniu *Unicode Demystified* autorstwa Richarda Gillama [Gil02, Unicode Storage and Serialization Formats].

2.4.6. UTF-16 i UTF-32 (Unicode)

UTF-16 i UTF-32 są odpowiednio 16- i 32-bitowym kodowaniem zestawu znaków Unicode. Mimo bliźniaczej nazwy, długość słowa maszynowego nie jest ich jedyną różnicą w stosunku do UTF-8. Dla przykładu, UTF-32 ma zawsze stałą długość, a numery Unicode koduje w sposób bezpośredni. Formaty te nie będą jednak szczegółowo omawiane w niniejszej pracy, ponieważ są używane przede wszystkim przez systemy operacyjne do wewnętrznego przechowywania napisów i mają drugorzędne znaczenie dla twórcy systemów webowych, który w większości przypadków powinien wybrać kodowanie UTF-8. Więcej informacji o obu standardach można znaleźć w [Gil02].

2.5. Format pliku z tłumaczeniami

Nieodłączną częścią procesu tworzenia interfejsów wielojęzykowych systemów webowych jest lokalizacja, z którą wiąże się tłumaczenie napisów (strategie ich przygotowywania zostaną szerzej omówione w kolejnych rozdziałach). Niezależnie od wybranego podejścia, tłumaczenia własnego czy wykonanego przez zewnętrzną firmę, napisy należy zapisać w formacie uwzględniającym wygodę zarówno programisty, jak i tłumacza. Często istotna jest również kompatybilność z oprogramowaniem komputerowym wspierającym tłumaczenie (ang. *computer aided translation*, skr. CAT). Formaty opisane w kolejnych punktach to, wg firmy Transifex zajmującej się profesjonalnie dostarczaniem tłumaczeń dla firm trzecich, jedne z częściej stosowanych [Gle16].

Schemat budowy typowego rekordu opiera się zwykle na powiązaniu tłumaczenia z pewnym identyfikatorem, który swoją nazwą określa miejsce wystąpienia napisu lub w inny logiczny sposób przyporządkowuje go do widoku (np. *views.offerDetails.header*). Innym stosowanym sposobem tworzenia identyfikatora jest streszczenie w jego nazwie

tłumaczonego napisu (np. *views.offerDetails.selectedCarDetailsAndReports*) – ten może się jednak z czasem zmienić. Możliwe jest również użycie w tej roli pełnego pierwotnego tekstu, zwykle angielskiego (np. *Here are selected car's details and reports provided by our partners*). Dobrą praktyką ułatwiającą i podnoszącą jakość pracy tłumacza jest dołączanie komentarzy określających kontekst, w jakim napis występuje.

2.5.1. CSV

Format CSV (ang. C*omma-Separated Values*) to jeden z prostszych sposobów przechowywania tłumaczeń (i danych w ogóle). Każdy wiersz oznacza rekord, który opisany jest grupą atrybutów rozdzielonych przecinkami. Za firmą PhraseApp można przyjąć konwencję[3], w której pierwszy atrybut (kolumna) oznacza identyfikator (klucz) tłumaczenia, drugi tłumaczenie, a trzeci komentarz (opcjonalnie):

```
title,"Ogłoszenia samochodowe","Tytuł na karcie strony i w wyszukiwarce"
views.offerDetails.header,"Szczegóły wybranego samochodu i raporty
przygotowane przez naszych partnerów","Główny nagłówek w widoku
szczegółowym ogłoszenia, ten między zdjęciem oferty i naszym logiem"
```

Zaletą CSV jest jego kompaktowość, czytelność i możliwość łatwego dołączania kolejnych danych. Wadą jest ograniczenie do płaskiej struktury w najprostszej formie.

2.5.2. JSON

Format JSON (ang. *JavaScript Object Notation*) to lekki i otwarty standard wymiany danych ogólnego przeznaczenia, oparty na podzbiorze języka JavaScript (kompatybilny jednak z wieloma innymi). Jego struktura jest budowana na bazie par klucz-wartość oraz kilku prostych typów, takich jak obiekty, tablice, napisy i liczby [Ecm17]:

```
{
  "title": "Ogłoszenia samochodowe",
  "views.offerDetails.header": "Szczegóły wybranego samochodu i raporty
  przygotowane przez naszych partnerów"
}
```

Innym sposobem przygotowania pliku w formacie JSON jest głębokie zagnieżdżanie elementów w stylu obiektowym:

```
{
  "title": "Ogłoszenia samochodowe",
  "views": {
    "offerDetails": {
      "header": "Szczegóły wybranego samochodu i raporty przygotowane
      przez naszych partnerów"
    }
  }
}
```

Zaletą JSON jest możliwość tworzenia hierarchicznych struktur danych bez istotnego narzutu składniowego i z zachowaniem wysokiej czytelności dla człowieka. Sporą wadą z punktu widzenia tłumacza jest brak obsługi komentarzy. JSON będzie stanowił silną konkurencję dla XML głównie w sytuacjach, gdy nie jest konieczna możliwość walidacji struktury danych względem narzuconego schematu [Pat17].

3 Zob. https://phraseapp.com/docs/guides/formats/csv/.

2.5.3. XLIFF (XML)

Format XLIFF (ang. *XML Localization Interchange File Format*) to standard przemysłowy zbudowany w oparciu o XML (ang. *Extensible Markup Language*), który powstał specjalnie z myślą o przechowywaniu lokalizowanych danych oraz ich wymianie między różnymi programami CAT. Rozwijany jest przez konsorcjum OASIS (ang. *Organization for the Advancement of Structured Information Standards*), a jego bieżąca wersja to 2.1 z lutego 2018 [Fil18]. Zestaw tłumaczeń analogiczny do wcześniejszych przykładów można zapisać w XLIFF następująco:

```
<xliff xmlns="urn:oasis:names:tc:xliff:document:2.1" version="2.1"
srcLang="en-US" trgLang="pl-PL">
  <file id="views">
    <unit id="common">
      <notes>
        <note id="note1" appliesTo="target">
          Tytuł na karcie strony i w wyszukiwarce
        </note>
      </notes>
      <segment id="title">
        <source>
          Car offers
        </source>
        <target>
          Ogłoszenia samochodowe
        </target>
      </segment>
    </unit>
    <unit id="offerDetails">
      <notes>
        <note id="note2" appliesTo="target">
          Główny nagłówek w widoku szczegółowym ogłoszenia, ten między
          zdjęciem oferty i naszym logiem
        </note>
      </notes>
      <segment id="header">
        <source>
          Selected car’s details and reports provided by our partners
        </source>
        <target>
          Szczegóły wybranego samochodu i raporty przygotowane przez
          naszych partnerów
        </target>
      </segment>
    </unit>
  </file>
</xliff>
```

Wszystkie unikalne elementy struktury (niebędące kontenerami na inne) muszą posiadać swój identyfikator *id*. Rekordy zebrane są w pliki identyfikowane znacznikiem *file* (np. według modułów systemu), które zawierają podjednostki grupujące *unit* (np. według widoków). Każda z nich może posiadać wiele segmentów właściwego tekstu między znacznikami *segment* oraz komentarze do nich w sekcji *notes*. Można zauważyć, że w tym przypadku struktura jest dużo bardziej formalna i rozbudowana. Tekst źródłowy oraz docelowe tłumaczenie są przechowywane razem, co nazywane jest bitekstem (ang. *bitext*).

Ułatwia to korektę tekstu i kontrolę jakości, zwłaszcza gdy nad dokumentem pracuje wiele osób. Pozwala także na dopasowanie wyrazów między oboma napisami, co polepsza skuteczność tłumaczenia maszynowego [Bur17].

Ze względu na swoją złożoność, XLIFF znajdzie zastosowanie głównie w największych projektach, nad których lokalizacją pracują wieloosobowe zespoły profesjonalistów posiłkujących się narzędziami CAT [Fil17]. Jeżeli w projekcie nie będą używane narzędzia potrafiące wykorzystać szeroki zestaw metadanych, jakimi można opisać tłumaczenia w formacie XLIFF, to jego zastosowanie będzie niepotrzebnym narzutem na rozmiar pliku.

3. Główne problemy lokalizacji

W tym rozdziale przedstawione zostaną najczęstsze problemy pojawiające się podczas pracy nad lokalizacją interfejsów wielojęzykowych aplikacji webowych. Opisane zostaną również wybrane techniki pozwalające na ich rozwiązanie.

3.1. Aspekty kulturowe

Przywołując ponownie drugą heurystykę Jakoba Nielsena [Nie93] mówiącą o zachowaniu spójności systemu komputerowego ze światem rzeczywistym można zauważyć, że dobra lokalizacja wymaga podjęcia szerszego spektrum czynności niż samo przetłumaczenie napisów. Dostosowując interfejs do potrzeb innej narodowości ważne jest, aby przynajmniej częściowo ją poznać i nauczyć się myśleć o rozwiązaniach, które nie narzucają schematów specyficznych dla kręgu kulturowego twórców aplikacji. W odbiorze będzie mieć to wpływ zarówno na elementarne zagadnienia, jak zrozumienie treści (np. wyświetlanie arabskich znaków zakodowanych przy użyciu UTF-8 nie wystarczy, jeżeli będą prezentowane w kierunku od lewej do prawej, zamiast od prawej do lewej), jak i detale, które zwiększą przystępność witryny (np. nazwa domeny internetowej zapisana cyrylicą, zamiast literami alfabetu łacińskiego).

3.1.1. Ustawienia regionalne

Zbiór zasad definiujących zachowanie systemu w przypadku konkretnej lokalizacji jest określany jako ustawienia regionalne (ang. *locale*) [Rau17]. Opisują one konwencje dotyczące kalendarza, czasu, walut, liczb, sortowania, porównywania, formatowania, itd. Mogą też zawierać informację o domyślnym kodowaniu znaków. Ustawienia regionalne są wykorzystywane przez system operacyjny, przeglądarkę i różne biblioteki do dostarczenia gotowych mechanizmów, z których może skorzystać programista podczas pracy nad internacjonalizacją [Ibm06]. Podstawowy format w jakim należy podać ustawienia jest następujący:

```
język[_terytorium]
```

Język podany zgodnie z normą ISO 639 (zob. punkt 2.3.1) jest jedynym wymaganym parametrem. Po nim następuje opcjonalne terytorium podane zgodnie z normą ISO 3166 (zob. punkt 2.3.2) i poprzedzone znakiem podkreślenia (w systemach webowych używa się myślnika). Terytorium podaje się z uwagi na różnice jakie mogą występować między krajami używającymi tego samego języka (np. angielskiego, używanego w Stanach Zjednoczonych, Wielkiej Brytanii i Australii, odpowiednio: *en_US*, *en_GB*, *en_AU*). Jeżeli region jest jednoznacznie definiowany przez język, zaleca się pominąć podawanie terytorium (np. w przypadku Polski wystarczy tag *pl* zamiast *pl_PL*). Istnieją również inne, opcjonalne modyfikatory stosowane w rzadkich przypadkach. Pełna specyfikacja tagów językowych dla systemów webowych została szczegółowo opisana w standardzie BCP 47 [Phi09].

3.1.2. Projekt interfejsu

Podczas badań przeprowadzonych wśród pracowników międzynarodowej korporacji IBM przez Geerta Hofstede, zdefiniowano cztery mierzalne wymiary kulturowe charakteryzujące społeczeństwa [Hof10]:

1. stosunek do władzy i autorytetów, rozumiany także jako pogląd na nierówności między różnymi grupami ludzi; społeczeństwa o wysokiej wartości tego współczynnika będą mocno hierarchiczne, podczas gdy te o niskiej charakteryzują się spłaszczoną strukturą organizacji,
2. indywidualizm i kolektywizm, oznaczający orientację na cele własne lub grupy; kultura indywidualistów charakteryzuje się luźnymi relacjami w małym gronie, podczas gdy kolektywna to ścisłe relacje powiązanych ze sobą dużych, silnych koterii, w których dobro ogółu jest ważniejsze niż własne,
3. męskość i kobiecość, zdefiniowane jako podejście do tradycyjnych ról społecznych; poza tym społeczeństwo męskie będzie cenić wartości takie jak asertywność i opanowanie, podczas gdy dla społeczeństwa kobiecego ważne będą emocje i troska o innych,
4. unikanie niepewności, wyrażające skłonność do podejmowania ryzyka i poziom akceptacji nieznanego.

Dwa uzupełniające wymiary to:

1. myślenie długoterminowe a krótkoterminowe, czyli hierarchia wartości, która powoduje ukierunkowanie działań na nagrodę i zysk w przyszłości lub skupienie na sprawach bieżących i wypełnianiu swoich obowiązków,
2. subiektywne zadowolenie z życia, na które składa się ogólne szczęście, poziom kontroli nad własnym życiem i stosunek do wypoczynku.

Wartości powyższych wymiarów, zwłaszcza dla kultur egzotycznych i odległych, stanowią praktyczne wyznaczniki, którymi powinni kierować się także projektanci interfejsów wielojęzykowych systemów webowych [Sub12]. Znaczenie poszczególnych wymiarów można zilustrować na przykładzie portalu producenta samochodów, który chce zachęcić do zakupu swoich produktów. Lokalizując taki system dla społeczeństwa o wysokim uznaniu dla władzy i autorytetów należy wyeksponować opinie inżynierów, którzy projektowali pojazd oraz prywatną rekomendację prezesa firmy. Właściwe będą również informacje o znanych osobistościach, które dany model użytkują. Tymczasem dla klientów o narodowości z silnie zaznaczonym indywidualizmem ważniejsze będzie zdanie kierowców, którzy dzięki reklamowanemu samochodowi odnieśli osobisty sukces w rajdzie lub powodzenie swojej podróży na drugi koniec świata. Ważne będą też odczucia zwykłych użytkowników, można zatem rozważyć udostępnienie im mechanizmu komentarzy. Dla społeczeństwa unikającego ryzyka liczyć będzie się podanie na wstępie jak największej ilości szczegółowych, wręcz tabelarycznych danych technicznych, które zbudują poczucie profesjonalizmu firmy i pomogą wybrać optymalny model. Dla klientów z kultury silnie kobiecej specyfikacja nie będzie mieć znaczenia. Najważniejsze będą emocje towarzyszące posiadaniu samochodu podkreślone zdjęciami zakochanej pary jadącej wybranym modelem w stronę zachodzącego słońca oraz bezpieczeństwo najbliższych gwarantowane symbolami firm certyfikacyjnych. Tabele zawierające wartości opisywanych wymiarów dla różnych krajów można znaleźć w [Hof10].

Szczególną uwagę podczas projektowania komponentów lokalizowanego systemu należy zwrócić także na metafory i symbole, sposób myślenia wpływający na organizację treści i nawigację, metody interakcji oraz wygląd [Mar01]. Dobrym przykładem kompleksowego uwzględnienia wszystkich tych aspektów jest projekt telefonu *Wukong PDA* przeznaczonego na rynek chiński. Przeprowadzone badania wykazały, że zamiast abstrakcyjnego, liniowego sposobu prezentacji kontaktów w sposób typowy dla kultury

zachodniej, lepiej zorganizować je w sposób podkreślający relacje społeczne i więzi biznesowe. Metafory i nawigacja zostały zbudowane wokół ludzi, zgodnie z narodową filozofią Guang-xi skupiającą się na podtrzymywaniu znajomości, kluczowych w społeczeństwie kolektywnym. Wprowadzanie znaków odbywa się za pomocą rysika i komend głosowych, ponieważ taki sposób interakcji z urządzeniem w przypadku pisma ideograficznego jest wygodniejszy niż klasyczna klawiatura. Ostatecznego szlifu telefonowi nadał odpowiedni wygląd. Chińczycy traktują takie urządzenia jak eksponowaną biżuterię, która musi dobrze wyglądać w ręku czy na pasku [Yul03].

Wdrożenie przytoczonych zasad w systemie webowym można zaobserwować porównując witrynę przedsiębiorstwa motoryzacyjnego Honda w wersjach przeznaczonych na rynek amerykański i japoński (Rys. 3.1). Stronę dla klienta z zachodu wypełniają w całości grafiki o dużej rozdzielczości, które zachęcają do dalszej, głębszej eksploracji serwisu ułożonego hierarchicznie według kategorii pojazdów. Ilość tekstu ograniczono do minimum, nawet menu zostało zwinięte do postaci pojedynczego przycisku. Nawigacja jest realizowana według ściśle zaplanowanej ścieżki. Stoi to w opozycji do wersji przeznaczonej dla klienta ze wschodu, który preferuje natychmiastowy dostęp do pełnej treści [Mcg17]. Ekran wypełnia głównie gęsto ułożony tekst, a rozbudowane menu pozwala na bezpośrednią nawigację do większości podstron. Jeszcze wyraźniejsze różnice są widoczne po wejściu w szczegóły wybranego modelu. Podczas gdy amerykańska witryna zapewnia nas, że samochód po prostu świetnie przyśpiesza, japoński odpowiednik prezentuje wykres z przebiegiem krzywej momentu obrotowego i opis mechanizmów zastosowanych w skrzyni biegów.

Rys. 3.1. Amerykańska (po lewej) i japońska (po prawej) witryna Hondy.

Przypadek Hondy jest jednak dość subtelny. Europejczyk prawdopodobnie dozna szoku kulturowego serfując po azjatyckim Internecie. Popularne wśród minimalistów hasło *mniej znaczy więcej*, nie ma w nim racji bytu. Wiele stron charakteryzuje się archaicznym układem z mnóstwem kolumn, gęsto rozmieszczonym tekstem przeplatanym licznymi grafikami w niskiej rozdzielczości, kontrastowymi kolorami oraz nawiązaniami do *uroczych*

(z jap. *kawaii*) postaci z kreskówek i komiksów [Gil13]. Interfejsy kilku popularnych w Azji portali zestawiono na Rys. 3.2.

Rys. 3.2. Projekty interfejsów popularnych azjatyckich portali.

Wpisanie się w lokalne, być może bardzo odległe, trendy lub narzucenie spójnej, globalnej wizji portalu jest kwestią biznesową i dotyczy przyjętej przez firmę strategii. W przypadku pierwszej opcji inspirację dla projektu lokalizowanego interfejsu można czerpać z codziennego środowiska danej narodowości, tego jak wyglądają główne ulice dużych miast, witryny sklepowe i okładki gazet – można tam znaleźć pewne analogie widoczne później w Internecie. Wiele o upodobaniach może powiedzieć też styl 500 najpopularniejszych w danym kraju stron webowych. Raporty takie są publikowane przez serwis Amazon Alexa[1]. Jeżeli różnice kulturowe są tak wyraźne, że dopasowanie istniejącego interfejsu do potrzeb nowej lokalizacji powoduje duże rozbieżności i problemy techniczne, należy rozważyć utworzenie osobnego, samodzielnego wariantu witryny. Temat strategii budowania takich rozwiązań zostanie omówiony w rozdziale 4.

Niepomiernym zadaniem byłoby przeanalizować w niniejszej pracy wszystkie istniejące kultury świata czy opisać choćby jedną w sposób wyczerpujący. Każdy kraj, naród i region charakteryzuje się unikalnym wachlarzem cech, którego znajomość istotnie wpływa na jakość przeprowadzonej lokalizacji. Punktem wyjścia do analizy tworzonego interfejsu pod kątem konkretnego społeczeństwa może być metodyka zaproponowana w opracowaniu *Handbook of Global User Research* [Sni10]. Pozycja ta zawiera również wartościowy przegląd społeczeństw z 21 krajów wykonany przez ekspertów w dziedzinie użyteczności z każdego z nich. Czytelnik ściśle zainteresowany kulturą dalekowschodnią powinien sięgnąć po przewodnik *Access to Asia* [Sch15]. Jest to opracowanie nietechniczne, jednak autorzy odwołują się w nim do przedstawionych na początku tego punktu wymiarów Hofsteda.

3.1.3. Ograniczenia techniczne

Istotny wpływ na pełne zrozumienie aspektów kulturowych będzie mieć świadomość realiów technologicznych występujących w danym regionie. Średnia prędkość Internetu (kwiecień 2018) w Korei Południowej wynosi 26,7 Mbps, podczas gdy w Indiach zaledwie 2,8 Mbps[2]. Dziesięciokrotna różnica w przepustowości łącza powinna znaleźć odzwierciedlenie w lokalizacji o zredukowanym wymiarze multimedialnym, w której treść przedkłada się nad formę. Etykiety i elementy nawigacyjne w postaci grafik powinny zostać zastąpione tekstowymi odpowiednikami. Optymalizacja rozmiaru witryny będzie polegać

1 Zob. https://www.alexa.com/topsites/countries.
2 Zob. https://www.fastmetrics.com/internet-connection-speed-by-country.php.

również na rezygnacji z części komponentów i skryptów lub wręcz przygotowaniu uproszczonej wersji tekstowej z zasobami graficznymi dostępnymi wyłącznie na żądanie [Mar03, s. 446].

Oprócz szybkości transmisji danych ważne jest sprawdzenie najpopularniejszej w danym regionie platformy. Według danych z kwietnia 2018, wenezuelscy internauci przeglądali sieć głównie za pomocą komputerów stacjonarnych (76% wszystkich użytkowników[3]). Tymczasem w Indiach rozkład ten był odwrotny, bo 78% udziału w rynku stanowiło serfowanie za pomocą urządzeń mobilnych[4]. Tak znaczące różnice mogły powstać z przesłanek kulturowych, ekonomicznych, politycznych lub innych. Ważne jest, aby w sytuacji, gdy dominują smartfony, stosować techniki projektowania responsywnego (ang. *responsive web design*, skr. RWD), które potrafią w locie dopasować interfejs i zawartość strony do mniejszych wyświetlaczy telefonów i tabletów [Tal16].

Przed premierą konsoli do gier Xbox w Japonii, firma Microsoft zlokalizowała nie tylko jej interfejs, ale także sam kontroler. Polegało to na przesunięciu przycisków bliżej siebie, aby lepiej dopasować ich rozkład do przeciętnego rozmiaru dłoni Azjaty, który jest mniejszy niż u gracza ze Stanów Zjednoczonych czy Europy [Gai02]. Analogiczne zmiany mogłyby dotyczyć układu głównych elementów nawigacyjnych w interfejsie strony uruchamianej na urządzeniu mobilnym, które powinny pozostawać w zasięgu kciuka (jednocześnie mogą być nieco mniejsze i gęściej ułożone).

Ostatnim elementem analizy ograniczeń technicznych jest zapewnienie kompatybilności wybranych do implementacji technologii z najpopularniejszymi w danym kraju przeglądarkami internetowymi[5]. Znając przeglądarkę, zgodność konkretnej wersji z poszczególnymi elementami specyfikacji HTML, CSS i JavaScript można szybko sprawdzić na stronie *Can I use...?*[6].

3.1.4. Rola tłumacza

Wkład tłumacza w tworzenie wielojęzykowego systemu webowego bywa ograniczany do ostatniej fazy pracy nad lokalizacją, gdy pozostało jedynie przetłumaczenie gotowej aplikacji. Tymczasem zaangażowanie go w roli eksperta dziedzinowego na wczesnym etapie prac może zapobiec wielu kosztownym poprawkom w przyszłości. Dobrego tłumacza cechuje nie tylko znajomość słownictwa i gramatyki, ale i kultury kraju, którego językiem włada. Idealnie jeżeli jest to jego język ojczysty, a ponadto posiada doświadczenie w branży dla której przygotowywany jest system. Taka osoba będzie silnym wsparciem dla projektantów i programistów, zwłaszcza biorąc pod uwagę liczbę czynników kulturowych, które dotąd przedstawiono, a jakie należy uwzględnić podczas lokalizacji. Pierwsze uwagi mogą się pojawić już na – z pozoru trywialnym – etapie tworzenia konta użytkownika, gdy okaże się, że imię w języku japońskim należy bezwzględnie zapisywać na trzy sposoby, a hiszpańskie nazwiska bywają nawet potrójne [Wit11]. Korygowanie takich błędów na wszystkich warstwach gotowej aplikacji byłoby nieporównywalnie bardziej czasochłonne.

W przypadku rozbudowanych projektów realizowanych w kilkunastu językach, zaleca się tworzenie wyspecjalizowanych zespołów lokalizacyjnych, nad którymi czuwa wewnętrzny manager globalizacji lub specjalista z firmy zewnętrznej, dbający o przepływ

3 Zob. http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/venezuela.
4 Zob. http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/india.
5 Zob. http://gs.statcounter.com/browser-market-share.
6 Zob. https://caniuse.com/.

i przedmiotowość prac. O strategiach organizacji takich grup w zależności od wielkości firmy, taktyce pozwalającej zatrudnić kompetentnego tłumacza i towarzyszących procesach w ujęciu korporacyjnym można przeczytać więcej w [Yun03] (rozdziały *Web Globalization Workflow* i *Translation Management*).

3.2. Tekst

Pomimo postępu technologicznego umożliwiającego wzrost udziału treści multimedialnych wśród materiałów dostępnych w Internecie, stanowią one jedynie uzupełnienie dla podstawowej formy przekazu jaką jest tekst. Nawet systemy o charakterze typowo rozrywkowym, opartym o strumieniowanie wideo, nadal wymagają elementów tekstowych pozwalających na nawigację, zarządzanie kontem czy interakcję z innymi użytkownikami. W podrozdziale 2.4 przedstawiono sposoby zapisywania tekstu w pamięci komputera, natomiast w tym opisane zostaną aspekty dotyczące jego wizualnej prezentacji człowiekowi.

3.2.1. Język strony

Aby ustawić język tekstu strony webowej należy użyć atrybutu *lang* [Ish14b]. Nie posiada on wartości domyślnej, jego brak jest traktowany jako język nieokreślony (alternatywnie można go dostarczyć w nagłówku HTTP). Parametr podaje się w formacie zgodnym z regułami przedstawionymi w punkcie 3.1.1. Ustawienie globalne dotyczące całego dokumentu definiuje atrybut znajdujący się w tagu *html*:

```
<html lang="fr">
  ...
</html>
```

Wartość tę można przesłonić lokalnie w obrębie pojedynczego tagu:

```
<p>
  <span lang="pl">Witaj na mojej wielojęzycznej stronie domowej!</span>
  <span lang="en-US">Welcome to my multilingual homepage!</span>
</p>
```

Warto stosować to proste ustawienie, ponieważ jego wdrożenie niesie ze sobą szereg wydatnych korzyści [Ish16a]. Po pierwsze, pozwala na ostylowanie fragmentów strony w zależności od języka. Po drugie, pomaga przeglądarce wybrać prawidłową strategię łamania tekstu i dopasować fonty. Po trzecie, usprawnia indeksowanie przez wyszukiwarki internetowe, co może zaowocować lepszym pozycjonowaniem strony w wynikach. Po czwarte, umożliwia prawidłową pracę wtyczek do sprawdzania pisowni i gramatyki. Po piąte, wspomaga pracę programów syntezy mowy i innych narzędzi ułatwień dostępu. Wreszcie, po szóste – ułatwia działanie algorytmów przetwarzania i tłumaczenia maszynowego.

Oprócz języka wymagane jest określenie kodowania znaków. Najnowszym i najwygodniejszym sposobem jego ustawienia jest użycie deklaracji *meta* w połączeniu z atrybutem *charset*. Zważywszy na optymalizację działania silników przeglądarek internetowych, informacja o kodowaniu powinna znajdować się w pierwszym kilobajcie pliku, dlatego zaleca się aby była to pierwsza deklaracja w ramach bloku *head* [Ish14a]:

```
<head>
  <meta charset="utf-8">
  ...
</head>
```

3.2.2. Fonty

Określenie języka i kodowania strony nie wystarczy, aby wyświetlić wielojęzykowy tekst. Oprócz nich potrzebna jest wizualna reprezentacja znaków, którą zapewnia font. Twórcy przeważnie nie zapewniają obsługi wszystkich rodzajów pisma i skupiają się na przygotowaniu glifów odpowiadających pewnemu podzbiorowi zestawu znaków Unicode. Konieczne jest zatem dobranie całego zestawu fontów, który obsłuży wszystkie wymagane języki.

Do czasu zdefiniowania przez organizację W3C standardów WOFF i WOFF2 (ang. *Web Open Font Format*), popularnymi formatami zapisu fontów były TTF (ang. *TrueType Font*) i jego rozwinięcie OTF (ang. *OpenType Font*) tworzone na przemian we współpracy między firmami Apple, Microsoft i Adobe. Choć oba rozwiązywały problemy swoich czasów, to nie były dostosowane do potrzeb współczesnych systemów webowych, ponieważ nie obsługiwały kompresji ani zarządzania prawami autorskimi. W uproszczeniu można przyjąć, że WOFF i WOFF2 stanowią opakowanie TTF i OTF dla potrzeb sieciowych, rozszerzając ich możliwości o wspomniane, brakujące cechy [Kew12]. Zalecanym formatem jest obecnie obsługiwany przez wszystkie przeglądarki (również mobilne) WOFF, który z czasem zostanie zastąpiony przez swojego następcę o wydajniejszej kompresji – WOFF2. Jeżeli fonty w tym formacie nie są dostępne, można przejściowo stosować równie dobrze wspierane TTF i OTF [Dea18]. Odradza się natomiast użycie fontów w formatach PS (ang. *PostScript*), EOT (ang. *Embedded OpenType*) i SVG (ang. *Scalable Vector Graphics*) z uwagi na znikomą adaptację przeglądarek internetowych.

Dobierając fonty najlepiej zdać się na profesjonalne bazy, które zapewnią zgodność ze standardami webowymi, wysoką jakość glifów i kompletny zestaw prawidłowych znaków dla wybranego języka. Tym samym uwolnią programistów i projektantów od samodzielnego rozwiązywania typowych problemów [Fin17]. Najpopularniejsze bazy to Fonts[7], Adobe Typekit[8] i Google Fonts[9]. Wyróżnia się zwłaszcza ostatnie rozwiązanie, ponieważ jest darmowe i nie wymaga rejestracji. Google udostępnia przyjazną wyszukiwarkę, która pozwala filtrować fonty według języka, stylu, grubości, szerokości, nachylenia, itd. Oprócz tego każdy font posiada swoją kartę, na której zebrano informacje na temat jego popularności, historii, bliźniaczych krojów oraz piaskownicę umożliwiającą podgląd przy różnych ustawieniach. Najciekawszą funkcją jest jednak generator reguł CSS *@font-face* służących do deklarowania fontów używanych w dokumencie. Pozwala on wybrać krój, grubość i zestaw obsługiwanych znaków, a następnie zaimportować font za pomocą pojedynczej linii CSS:

```
@import url("https://fonts.googleapis.com/css?family=Itim&subset=thai");
```

Dzięki powyższej linii, system webowy zyska możliwość wyświetlania znaków alfabetu tajskiego. W importowanym linku kryje się wygenerowane automatycznie kilkadziesiąt linii kodu, które inaczej należałoby wprowadzić ręcznie w ramach reguły *@font-face* (nazwy, ścieżki, zakresy znaków, wagi, formaty odpowiednie dla wszystkich przeglądarek, itd.). Dodatkową zaletą jest przechowywanie fontów na szybkich serwerach Google, usługę można zatem traktować jako bezpłatny CDN (ang. *Content Delivery Network*).

Osobną klasę problemu stanowią fonty przeznaczone dla rodziny pism ideograficznych wywodzących się z kręgu CJKV (ang. C*hinese Japanese Korean Vietnamese*). Zagadnienie to wraz z pochodnymi zostało szeroko i szczegółowo omówione

7 Zob. https://www.fonts.com/web-fonts.
8 Zob. https://typekit.com/fonts.
9 Zob. https://fonts.google.com/.

przez Kena Lunde w książce *CJKV Information Processing* [Lun09]. Na potrzeby tej pracy wystarczy zauważyć, że tradycyjne pismo chińskie składa się z kilkudziesięciu tysięcy glifów, które porównując do kilkudziesięciu liter w alfabecie łacińskim stanowią znaczącą różnicę o bezpośrednim wpływie na duży rozmiar pliku z fontem. Ten może przekroczyć nawet 100MB[10], do tego prawdopodobnie potrzebne jest kilka krojów oraz pozostałe używane języki. W efekcie same fonty mogłyby zajmować wiele megabajtów i znacząco opóźnić czas ładowania strony webowej. Pierwszym rozwiązaniem jest używanie fontów systemowych zainstalowanych na komputerze użytkownika. Współczesne systemy operacyjne posiadają wbudowaną obsługę najbardziej egzotycznych języków świata, ich aktualną listę można znaleźć w Internecie[11]. Ryzykowne jest jednak założenie, że odbiorca posiada na swoim komputerze zasoby, z których będzie można zawsze skorzystać. Drugie rozwiązanie opiera się na obserwacji, że w pismach ideograficznych powszechnie wykorzystuje się tylko niewielką część znaków. Rząd Chiński opublikował tzw. tabelę standardowych znaków chińskich [Bnu14], czyli zestawienie w którym podaje, że w codziennej komunikacji używa się od 3500 do 6500 logogramów, co pozwala na pełne zrozumienie np. treści gazet. Również w specyfikacji Unicode określono podzbiór około 20000 logogramów uniwersalnych dla CJKV [Uni17, s. 683]. CSS udostępnia mechanizm, który pozwala podać zakres znaków jaki ma zostać pobrany w ramach fontu. Służy do tego właściwość *unicode-range*:

```
@font-face {
  font-family: "Noto Sans";
  unicode-range: U+4E00-9FFF;
  src: url("NotoSans.woff2") format("woff2");
}
```

W ten sposób można ograniczyć rozmiar pobieranych danych do kilkudziesięciu procent rozmiaru pierwotnego. Technikę tę można stosować również do zmniejszania rozmiaru fontów pozostałych pism. Przykładowo, jeżeli zachodzi potrzeba wykorzystania dodatkowego kroju do ozdobienia nagłówka strony w języku angielskim, to nie ma potrzeby pobierać pełnego fontu z literami greckimi i kompletem symboli matematycznych, wystarczy podstawowy zakres ASCII (*U+0020-007F*).

W punkcie 3.2.1 przedstawiono atrybut *lang* używany do określania języka w obrębie danego tagu HTML. Praktycznego zastosowania nabiera w połączeniu z analogicznym selektorem CSS, który pozwala użyć innych fontów w zależności od języka [Har07, s. 321]:

```
*:lang(fr) {
  font-family: "Fournier MT", serif;
}
*:lang(de) {
  font-family: "DS-Normal-Fraktur", serif;
}
```

Dzięki powyższemu ustawieniu Francuz i Niemiec będą mogli oglądać stronę w swoich ulubionych krojach pisma – delikatnym lub gotyckim, a programista nie będzie musiał ręcznie przełączać fontów między akapitami czy stronami. Poza samymi fontami można w ten sposób zmienić interlinię, rozmiar tekstu lub dowolny inny styl, co czyni z atrybutu HTML *lang* i selektora CSS *:lang* ważne narzędzia internacjonalizacji.

10 Bezpłatny font Noto Sans przeznaczony dla języka koreańskiego zajmuje 115MB, podczas gdy w wersji pokrywającej 582 pisma nieideograficzne w 72 stylach waży jedynie 16MB.

11 Zob. https://r12a.github.io/scripts/fontlist/ (dla Windows i macOS).

Ostatnią z wartych zaprezentowania technik jest użycie właściwości CSS *text-emphasis*. Stosuje się ją jako graficzną alternatywę dla metod typograficznych takich jak pogrubienie, kursywa czy kapitaliki, które mogą nie być dostępne w niektórych fontach (lub pismach). Zamiast nich pozwala wyświetlić wybrane kształty lub znaki, nad albo pod wyróżnionymi tekstem (w przykładzie kropki):

```
em {
  text-emphasis: dot;
}
```

Pełen zestaw możliwych parametrów jest dostępny w dokumentacji na stronach *Mozilla MDN Web Docs* [Mdn18a].

3.2.3. Długość tekstu

Pojedyncze glify używane w językach azjatyckich często oznaczają słowa lub całe wyrażenia (w odróżnieniu od liter np. alfabetu łacińskiego, które oznaczają dźwięki). Tak zwane *logogramy* zmieniają podejście do prezentacji treści - z jednej strony można dzięki nim przekazać więcej informacji za pomocą mniejszej liczby znaków, z drugiej strony znaki te powinny być nieco większe, aby zachowały czytelność. Portal społecznościowy Twitter uwzględnił te różnice wydłużając maksymalną dozwoloną długość wpisów publikowanych w językach innych niż chiński, japoński i koreański dwukrotnie, ze 140 do 280 znaków [Ros17]. Jest to przykład dobrej praktyki lokalizacyjnej – zamiast ograniczać zwięzły sposób pisma i karać jego użytkowników, różnicę skompensowano w górę w pismach mniej zbitych, tak aby w każdym języku długość wpisu umożliwiała przekazanie tak samo treściwej wiadomości. Decyzję motywowano prostą heurystyką, zgodnie z którą dominanta długości wpisu wykonanego za pomocą pisma logograficznego była dwukrotnie niższa niż w przypadku pisma sylabicznego. Do podobnych wniosków doszli dziennikarze *The Washington Post* [Ing17], którzy badali długość tekstu *Powszechnej deklaracji praw człowieka* przetłumaczonej na większość języków świata. Skrajne pozycje zajęły: język wietnamski (tekst dłuższy od angielskiego o 25%) i chiński (tekst czterokrotnie krótszy!).

Tab. 3.1. Przyrost długości tekstu tłumaczonego z angielskiego na inny język europejski.

Liczba znaków w tekście angielskim	Dodatkowa przestrzeń wymagana dla tekstu przetłumaczonego
Do 10	100% do 200%
11 do 20	80% do 100%
21 do 30	60% do 80%
31 do 50	40% do 60%
51 do 70	31% do 40%
Ponad 70	30%

Źródło: opracowanie własne na podstawie [Ibm16a].

W przypadku języków europejskich można przyjąć, że język angielski, zwłaszcza w krótkich formach, będzie najbardziej zwięzłym, a jego tłumaczenia będą wymagały większej ilości wolnej przestrzeni. Jako praktyczną zasadę, firma IBM w swoich wytycznych zaleca dostosowanie kontenerów na tekst do możliwości powiększenia o wartości podane

w Tab. 3.1. Oczywiście nie sprawdzą się one we wszystkich przypadkach, stanowią jednak dobry punkt wyjścia, który można zmodyfikować do własnych potrzeb. W przypadku zbitych języków dalekowschodnich przedstawione reguły przypuszczalnie spowodują powstawanie dużych, niewypełnionych przestrzeni. Najlepiej jest zatem przygotować stronę w taki sposób, aby umożliwić automatyczne dopasowywanie do treści, swobodne zawijanie tekstu i przepływ elementów [Ish07]. Unikać należy natomiast nieelastycznych szablonów opierających układ w HTML o tabele (<*table*>) czy bezwzględnego podawania wymiarów w arkuszach stylów CSS. Przykładowo, zamiast ustawiać wymiary kontenera *text-container* na sztywno pod najdłuższe tłumaczenie (przyjęto 200px):

```
.text-container {
  width: 200px;
}
```

lepiej zrobić to następująco:

```
.text-container {
  max-width: 200px;
}
```

Zamiana właściwości *width* na *max-width* pozwoli uniknąć pustych przestrzeni w przypadku krótszych tłumaczeń. Szerokość elementu zamiast na stałe wynosić zadaną wartość, będzie powiększana do 200px dynamicznie, w zależności od długości tekstu. Kolejnym krokiem ku automatycznemu skalowaniu interfejsu może być podanie wymiarów za pomocą względnych wartości procentowych (zamiast bezwzględnych w pikselach), a nawet całkowita rezygnacja z ręcznego ustalania wymiaru danej sekcji i zdanie się na rozkład generowany w locie przez silnik przeglądarki internetowej (w oparciu o zawartość strony).

Rozbudowane projekty o złożonym, trudnym do zmiany układzie elementów, mogą skorzystać ze stosunkowo nowego modułu *Flexbox Layout* wprowadzonego w CSS3. Przed jego użyciem należy sprawdzić kompatybilność z urządzeniami odbiorców docelowych, gdyż może on nie być obsługiwany przez starsze przeglądarki[12]. Flexbox pozwala na wygodne tworzenie kontenerów, których elementy dynamicznie dopasowują się względem siebie i swojej treści. Za pomocą pojedynczych właściwości można szybko uzyskać efekty wymagające wcześniej bardzo dobrej znajomości CSS. Można m.in. ustalić kierunek elementów (np. pionowy albo od prawej do lewej), zmienić ich kolejność bez ingerencji w kod HTML czy określić strategię, zgodnie z którą rozdysponowywana będzie wolna przestrzeń. Zakładając układ strony z kontenerem zawierającym elementy z tekstem w trzech różnych językach[13]:

```
<div class="text-container">
  <div class="text-chinese" lang="zh">
    人人生而自由,在尊严和权利上一律平等。
  </div>
  <div class="text-polish" lang="pl">
    Wszyscy ludzie rodzą się wolni i równi.
  </div>
  <div class="text-german" lang="de">
    Alle Menschen sind frei und gleich an Würde und Rechten geboren.
  </div>
</div>
```

12 Zob. https://caniuse.com/#search=flexbox.

13 W tym, jak i w dalszych przykładach wykorzystano fragmenty *Powszechnej deklaracji praw człowieka*, zob. http://www.ohchr.org/EN/UDHR/Pages/SearchByLang.aspx.

można użyć prostego, przykładowego stylu:

```
.text-container {
  display: flex;
}
.text-polish {
  flex-shrink: 0;
}
.text-german {
  flex-grow: 1;
}
```

Właściwość *display: flex* powoduje traktowanie kontenera zgodnie z regułami Flexbox, *flex-shrink: 0* oznacza, że element nie będzie się kurczyć ani zawijać treści, a *flex-grow: 1* powoduje proporcjonalne rozszerzanie elementu, jeżeli pozostałe nie będą wymagać dodatkowej przestrzeni. W ten sposób można budować kolejne fragmenty interfejsu odpornego na problemy zmiany długości zlokalizowanego tekstu. Bardzo dobry, a przede wszystkim interaktywny i regularnie aktualizowany przewodnik po module Flexbox można znaleźć w portalu *CSS-Tricks* [Coy18].

Awaryjna technika to ucinanie tekstu i kończenie go wielokropkiem w sytuacjach, gdy wyświetlenie pełnej treści nie jest bezwzględnie wymagane, a wygospodarowanie dodatkowego miejsca za pomocą wcześniej opisanych sposobów niemożliwe. Należy pamiętać, że użycie tej metody będzie niewłaściwe dla etykiet, przycisków i innych kontrolek odpowiedzialnych za sterowanie systemem, ponieważ może powodować niezrozumienie ich przeznaczenia. Sprawdzi się za to w przypadku odnośników, których kliknięcie skutkuje przejściem do widoku szczegółowego, np. nagłówek przenoszący do pełnej treści artykułu. Kontynuując wcześniejszy przykład:

```
.text-chinese {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Właściwość *white-space: nowrap* uniemożliwia złamanie linii, *overflow: hidden* ukrywa nadmiarowe znaki, a *text-overflow: ellipsis* powoduje wyświetlanie wielokropka w miejscu ucięcia tekstu. Pierwsza właściwość jest konieczna, ponieważ sposób ten działa tylko dla pojedynczej linii. CSS do dziś nie specyfikuje uniwersalnej metody eleganckiego ucinania tekstu w dowolnym wierszu akapitu [Ras16].

3.2.4. Kierunek pisma

Na tym etapie strona webowa posiada określony język, kodowanie znaków, fonty, układ przystosowany do różnej długości tekstu oraz przetłumaczone napisy. Ostatnią właściwością tekstu jaką należy uwzględnić jest jego kierunek. Przyjmuje się następujące skróty [Ish16b]:

1. kierunek od lewej do prawej – LTR lub LR (ang. *left-to-right*), większość pism świata,
2. kierunek od prawej do lewej – RTL lub RL (ang. *right-to-left*), głównie pismo arabskie i hebrajskie,
3. mieszanie LTR i RTL w tym samym tekście – BiDi (ang. *bidirectional*), np. liczby albo łacińskie nazwy własne pojawiające się w dokumencie syryjskim.

Kierunek tekstu powinien być odzwierciedlony także w układzie elementów interfejsu i na grafikach. Ciekawy przykład na poparcie tej tezy podaje [Tex05]. Jeżeli w reklamie środka piorącego będącej tryptykiem z obrazów: brudna koszula → użycie proszku → czysta koszula (LTR), nie zostanie odwrócona kolejność grafik, to w kulturze arabskiej (RTL) przeznaczenie produktu będzie niezrozumiałe lub wręcz zinterpretowane jako środek brudzący. Czy w takim razie w przypadku pisma RTL należy wykonać pełne lustrzane odbicie całej strony? Okazuje się, że wiele elementów interfejsu zostało na przestrzeni lat zglobalizowane do tego stopnia, że nawet w kulturze bliskowschodniej w niektórych przypadkach naturalny jest układ LTR. Dotyczy to m.in. adresów URL, wykresów, zapisu nutowego czy przycisków do obsługi odtwarzaczy wideo (sprzęt audiowizualny, sprowadzany przez wiele lat z Ameryki i Europy, nie był lokalizowany na rynek arabski) [App15]. Bardziej drastyczne skutki globalizacji można dostrzec w kulturze dalekowschodniej – pisma CJKV, choć tradycyjnie zapisywane kolumnami RTL, w Internecie przyjęły zapis zachodni, wierszami LTR. Posługują się nim nie tylko najpopularniejsze (według Amazon Alexa) strony komercyjne, ale również oficjalne strony rządowe[14]. Przez wiele lat było to spowodowane ograniczeniami technologii webowej powstającej w kulturze LTR, ale i dziś łatwiej wykonać responsywny interfejs skalujący się w stosunku do szerokości strony niż do jej wysokości[15].

Domyślnym kierunkiem tekstu w dokumentach HTML jest LTR. Prawidłowym i jedynym zalecanym przez organizację W3C sposobem jego zmiany jest dodanie atrybutu *dir* do tagu *html* [Ish16c]:

```
<html dir="rtl">
  ...
</html>
```

Nieprawidłowe jest stosowanie w tym celu ręcznie wstawianych, specjalnych znaków kontrolnych lub formatowanie za pomocą CSS. Informacja o kierunku tekstu jest ściśle związana z dokumentem i bezpośrednio wpływa na jego czytelność, nie może być zatem zależna od stylu wizualnego, który można wyłączyć. Poza tym użycie innych technik zaburza działanie algorytmu UBA (ang. *Unicode Bidirectional Algorithm*), który automatycznie wykrywa kierunek tekstu i umożliwia poprawne łączenie pism LTR i RTL w jednym dokumencie. Jest to kolejny przypadek, w którym opłaca się stosować zestaw znaków Unicode, ponieważ odciąża programistę od konieczności samodzielnego rozwiązywania problemów związanych z BiDi. Więcej na temat sposobu działania algorytmu UBA można przeczytać w [Ish16d].

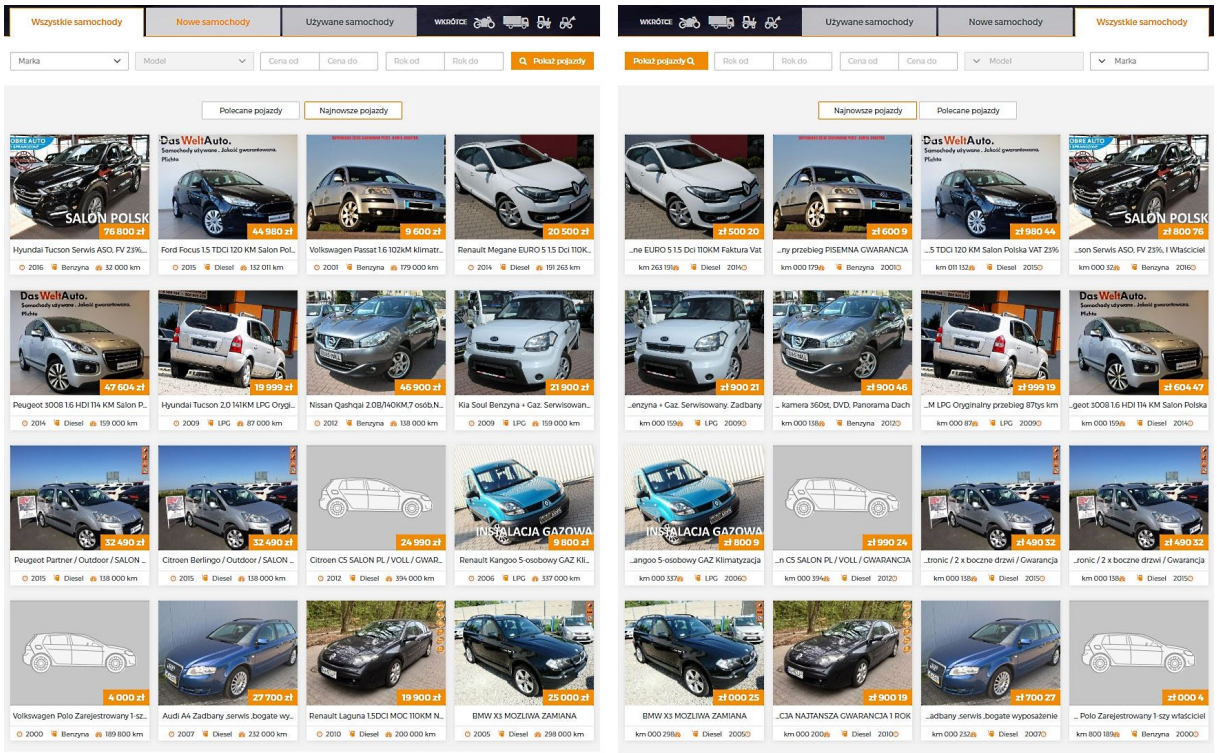
Opisane powyżej ustawienie określa bazowy kierunek całego dokumentu. Jeżeli jednak zachodzi potrzeba jego lokalnej zmiany lub wyświetlenia tekstu o nieznanym kierunku (np. pobranego z zewnętrznego źródła), należy taki fragment oddzielić za pomocą tagu *bdi* [Rob13]:

```
<html dir="rtl">
  ...
  يولد جميع الناس <bdi>Alle Menschen sind frei</bdi> يعامل بعضهم
</html>
```

14 Strona rządu chińskiego zob. http://www.gov.cn/, japońskiego zob. http://www.e-gov.go.jp/, koreańskiego zob. https://www.gov.kr/portal/main.

15 Według *Can I use…?*, moduły CSS ułatwiające to zadanie, takie jak *Flexbox* czy *Viewport Units*, zostały zaadaptowane na szeroką skalę dopiero po 2013 roku. *Grid Layout* obsługiwany jest tylko w absolutnie najnowszych wersjach przeglądarek (głównie na komputery stacjonarne).

Jest to wskazówka, która pomoże algorytmowi UBA jednoznacznie i zgodnie z intencją autora zinterpretować liczby, znaki interpunkcyjne oraz symbole takie jak < i > (domyślnie odwracane). Jeżeli kierunek jest znany, można go od razu określić atrybutem *dir* ustawionym na najbliższym znaczniku HTML [Ish16e].



Rys. 3.3. Motomi: dokument wyświetlany w kierunku LTR (po lewej) i RTL (po prawej).

Efekt zmiany kierunku tekstu wykonanej za pomocą opisywanych technik w systemie Motomi przedstawiono na Rys. 3.3. Odwrócony został układ większości elementów interfejsu: przycisków, zakładek, pól wyszukiwarki, a nawet kolejność w jakiej prezentowane są samochody (tu ponownie sprawdził się Flexbox omówiony w punkcie 3.2.3, który w takich sytuacjach potrafi reagować na zmianę kierunku automatycznie). Dlaczego zatem żółte pole z ceną nie zostało prawidłowo odbite? Wynika to ze specyfikacji HTML, zgodnie z którą przeglądarka odwraca kolejność logiczną, a nie wizualną elementów [Ish16f]. Jeżeli pozycja bloku została ustalona względem prawej krawędzi, to ustawienie to będzie obowiązywać również po zmianie kierunku tekstu. Pierwszym i naturalnym rozwiązaniem tego problemu jest unikanie sztywnego pozycjonowania elementów, kłopotliwego również w innych sytuacjach. Alternatywnym, bardziej realistycznym podejściem, zwłaszcza w dobie skomplikowanych wizualnie projektów stron webowych, jest utworzenie kopii pliku ze stylami dla RTL i zmiana wszystkich właściwości CSS zależnych od kierunku na przeciwne (*float: left* na *float: right*, itd.). W sieci dostępne są narzędzia, takie jak *RTLCSS*[16], które potrafią wygenerować odbitą wersję arkusza stylów na podstawie wgranego pliku, nie trzeba zatem robić tego ręcznie. Takie rozwiązanie powoduje jednak dużą redundancję kodu, ponieważ w odwróconym pliku powtórzone zostaną wszystkie niezwiązane z kierunkiem właściwości dotyczące kolorów, cieni, obramowań, itd. Usprawnienie tej metody zaproponowane przez [Lay15], polega na jednoczesnym wykorzystaniu dwóch plików CSS,

16 Zob. http://rtlcss.com/.

z których pierwszy zawiera pełny, podstawowy styl, a drugi nadpisuje go jedynie w miejscach, w których byłby odbity:

```
<link rel="stylesheet" type="text/css" href="style.css">
<link rel="stylesheet" type="text/css" href="style-rtl.css">
```

Plik style.css:

```
div {
  color: navy;
  border: 2px solid;
  float: left;
}
```

Plik style-rtl.css:

```
div {
  float: right;
}
```

Takie podejście upraszcza utrzymanie kodu i spójnego wyglądu, ponieważ wersja RTL współdzieli z LTR pełen komplet reguł zebranych we wspólnym pliku, skupia się za to na precyzyjnym nadpisaniu kierunku poszczególnych elementów, zamiast budować styl od podstaw.

Jak wykazano wcześniej, w Internecie upowszechnił się poziomy sposób przedstawiania pism CJKV. Jeżeli mimo tego zachodzi potrzeba ułożenia tekstu w kolumnach zamiast wierszami, należy użyć właściwości CSS *writing-mode* ustawionej na *vertical-lr* lub *vertical-rl* (zależnie od kierunku, w którym mają być dokładane kolumny) [Mey18]:

```
.text-chinese {
  writing-mode: vertical-rl;
  max-height: 50%;
}
```

Nie powinno się stosować bliźniaczych wartości *sideways-lr* ani *sideways-rl*, ponieważ zostały przewidziane do tworzenia podpisów rysunków i tabel, a nie pisma kolumnowego, i powodują obrócenie znaków o 90 stopni. Należy też pamiętać o ograniczeniu wysokości kontenera – inaczej znaki wypełnią pojedynczą, wysoką kolumnę.

3.3. Sortowanie

Sortowanie i wyszukiwanie to dwa popularne zadania stawiane przed wieloma systemami webowymi. Mogłoby się wydawać, że do porównania dwóch glifów wystarczy sprawdzenie ich pozycji w zestawie znaków, jednak z powodu wstecznej kompatybilności z ASCII, większość kodowań przedstawionych w podrozdziale 2.4 przechowuje wszystkie litery diakrytyczne za podstawowym alfabetem łacińskim, więc taki sposób zadziałałby tylko dla języka angielskiego. Kolejna komplikacja to sposób traktowania dwuznaków (np. *ch*, *dz*, *sh*) oraz ligatur (np. *æ*, *ʤ*, *ß*) – jako jeden znak czy osobno? Co jeżeli ten sam glif w dwóch językach jest traktowany w inny sposób i różni się pozycją (np. *ö* w języku szwedzkim jest osobną literą na końcu alfabetu, podczas gdy w języku niemieckim to nadal *o*, tylko z akcentem)? Jak traktować znaki, których reprezentacja powstała w wyniku złożenia kilku innych (np. wietnamskie *ộ*) albo jest zależna od pozycji w zdaniu (np. arabskie ﺺ i ﺻ)? Czy należy rozróżniać duże litery i uwzględniać znaki interpunkcyjne? Wreszcie jaką kolejność mają logogramy? Podobnych pytań można zadać więcej [Gil02, The Basics of Language-sensitive String Comparison], jednak już na tym etapie widać, że samodzielna

implementacja sortowania listy towarów wielojęzycznego sklepu internetowego staje się zadaniem nietrywialnym.

Odpowiedzią na te problemy jest (ponownie) zestaw znaków Unicode, który w ramach standardu UCA (ang. *Unicode Collation Algorithm*) opisuje algorytm porównywania znaków (ang. *collation*) oraz ich domyślną kolejność [Gil02, Language-sensitive Comparison on Unicode Text]. Model działa w oparciu o kilka poziomów wag przypisywanych każdemu znakowi oraz normalizację, której potrzeba wynika ze sposobu wewnętrznej organizacji Unicode. Po pierwsze, niektóre znaki można utworzyć z połączenia kilku innych, po drugie, większość z nich ułożona jest w porządku logicznym, a nie wizualnym. Oznacza to, że ten sam glif może być przypisany do kilku pozycji, np. Ω reprezentuje zarówno grecką literę *omega* (U+03A9), jak i symbol fizyczny *ohm* (U+2126) [Gil02, Multiple Representations]. Dostarczenie właściwej implementacji algorytmu spoczywa na twórcach bibliotek programistycznych i silników bazodanowych. W przypadku MySQL do zapytania wystarczy dodać klauzulę *COLLATE*:

```
SELECT * FROM products ORDER BY name COLLATE utf8_unicode_ci
```

3.4. Format danych

Różnice w sposobie przedstawiania danych występują nawet wśród krajów z tego samego kręgu kulturowego. Rozbieżne mogą być nie tylko wielkości podawane w innych jednostkach (np. w systemie metrycznym lub imperialnym), ale także format ich prezentacji, nawet jeżeli dotyczą de facto tych samych danych (np. numeru telefonu). Przykłady takich różnic wśród reprezentantów kultury europejskiej, amerykańskiej, azjatyckiej i arabskiej zebrano w Tab. 3.2. Podczas implementacji należałoby uwzględnić także odmienny format adresu, numeru telefonu, papieru czy nazwiska [Mar03, s. 443-445, 447].

Tab. 3.2. Format danych w zależności od kraju.

Kraj	Waluta	Czas	Data	Wielkości fizyczne	Dane liczbowe
Polska	123,45 zł	20:00	31.05.2018	1,23 kg	12 345,67
USA	$123.45	8:00 P.M.	5/31/18	1 lb 23 oz	12,345.67
Japonia	12,345 円	8 時 0 分	2018 年 05 月 31 日	1.23 kg	一万二千三百四十五[17]
Katar	12.345 QAR	20:00	1439/9/16[18]	1.23 kg	12,345.67

Źródło: opracowanie własne na podstawie [Mar03, s. 443-445, 447].

Powszechnie uznanym i zrozumiałym na całym świecie kalendarzem do użytku cywilnego i administracyjnego jest kalendarz gregoriański. Pomimo równoległego funkcjonowania wielu innych kalendarzy narodowych (hebrajski, perski, chiński, itd.), służą one głównie do wyznaczania świąt w celach religijnych i mają charakter lokalny [Lon05]. Analogiczna sytuacja ma miejsce w przypadku międzynarodowego wzorca czasu, za który przyjmuje się uniwersalny czas koordynowany UTC. Pomimo globalnych metod wyznaczania daty i godziny, poszczególne kraje różnią się formatem, w jakim je prezentują. Norma określająca sposoby ich zapisu została zawarta w standardzie ISO 8601 [Pow15]. Twórcy

17 Tradycyjny zapis w *Kanji* jest coraz częściej wypierany przez cyfry arabskie.

18 Kalendarz muzułmański pełni rolę pomocniczą w stosunku do gregoriańskiego.

wielojęzykowych systemów webowych nie muszą implementować jej samodzielnie i mogą skorzystać z zestawu funkcji zawartych w przestrzeni nazw *Intl* języka JavaScript, które umożliwiają prawidłowe formatowanie danych na podstawie podanych ustawień regionalnych (*locale*, patrz punkt 3.1.1). Pozwala to na przechowywanie danych w uniwersalnej postaci i deklaratywne przełączanie sposobu ich prezentacji w interfejsie w zależności od wybranego języka i kraju. Do formatowania daty i czasu służy obiekt *Intl.DateTimeFormat*:

```
let date = new Date(Date.UTC(2018, 4, 31, 12, 0, 0))

new Intl.DateTimeFormat('pl').format(date)

// 31.05.2018

new Intl.DateTimeFormat('zh', {
  year: 'numeric',
  month: 'long',
  day: 'numeric'
}).format(date)

// 2018年5月31日

new Intl.DateTimeFormat('de', {
  weekday: 'long',
  year: 'numeric',
  month: 'long',
  day: 'numeric',
  hour: 'numeric'
}).format(date)

// Donnerstag, 31. Mai 2018, 14 Uhr
```

Do formatowania danych liczbowych i walutowych służy obiekt *Intl.NumberFormat*:

```
let number = 123456.789

new Intl.NumberFormat('pl').format(number)

// 123 456,789

new Intl.NumberFormat('en-US', {
  style: 'currency',
  currency: 'USD'
}).format(number)

// $123,456.79
```

Pełna lista parametrów i przykłady zastosowania obiektu *Intl* znajdują się w dokumentacji prowadzonej na stronach *Mozilla MDN Web Docs* [Mdn18b].

3.5. Grafika i multimedia

Współczesne systemy webowe pełne są treści multimedialnych, takich jak grafiki, ikony, filmy i podcasty. Zasoby te należy przygotować w sposób uniwersalny dla wszystkich regionów albo dostosować do każdego z osobna. Ponieważ druga opcja jest o wiele bardziej czasochłonna, już na etapie przygotowywania materiałów powinno unikać się korzystania

z elementów ściśle wiążących z konkretną kulturą. Najprostszy przykład to rezygnacja z tekstu na grafikach, ale ostrożnie należy podchodzić także do przedstawiania zwierząt, ludzi, części ciała, kolorów, gestów, symboli religijnych i narodowych oraz gier słownych [Mar03, s. 446]. Multimedia trudne do przygotowania w sposób uniwersalny mogą być lokalizowane za pomocą informacji wielomodalnej: do filmu i podcastu można dołączyć ścieżkę lektora lub przetłumaczone napisy, zdjęcia adnotować atrybutem *alt*, a skróty rozwijać w *title*:

```
<img src="arrow-right.png" alt="Przejdź do strony 3." />

<abbr title="International Organization for Standardization">ISO</abbr>
```

Ewentualne odstępstwo od zasady unikania tekstu w multimediach może być umotywowane sytuacją, w której wybór fontów dla danego pisma jest niewielki, a użycie grafiki to jedyny sposób na uzyskanie atrakcyjnego kroju.

Symbole i ikony używane w interfejsie mają usprawniać pracę z systemem webowym, dlatego należy zadbać, aby ich przeznaczenie nie stanowiło mętnej zagadki, tylko było tak samo zrozumiałe wśród użytkowników z wszystkich obsługiwanych kręgów kulturowych. Jak przedstawiono na Rys. 3.4 różnić może się nawet sposób prezentacji powszechnie znanych pojęć takich jak *dom*. Dobrą praktyką jest zatem łączenie ikon z etykietami opisującymi ich znaczenie oraz korzystanie z symboli będących pochodnymi oznaczeń z międzynarodowych systemów bezpieczeństwa i ruchu drogowego, które w podobnej formie używane są na całym świecie [Rau11, s. 688-689]. Kierować można się także regułami zdefiniowanymi przez organizację ISO, która w serii kilku standardów opisuje zasady projektowania uniwersalnych, międzynarodowych symboli. Ich listę można znaleźć w [Iso13b]. Oprócz tego ISO udostępnia (odpłatnie) kilka tysięcy znormalizowanych znaków z różnych dziedzin, które można przeglądać za pomocą wyszukiwarki *Online Browsing Platform*[19].

Rys. 3.4. Symbol domu w kulturze europejskiej, japońskiej i arabskiej. Źródło: opracowanie własne na podstawie https://www.flaticon.com/.

Niejednoznacznie postrzegane są także gesty i kolory. Wydawałoby się uniwersalny znak kciuka w górę jest interpretowany w Australii jako niezbyt grzeczna forma kontrargumentacji podczas dyskusji. Z tego powodu pewien znany portal społecznościowy musiał w tym rejonie zrezygnować ze swojego flagowego symbolu na rzecz tekstowego odpowiednika [Yip13]. Podobny wpływ na odbiór grafiki mają kolory – dla Japończyka preferującego pastelowe barwy, intensywna czerwień będzie zbyt agresywnym i nieprofesjonalnym kolorem dla systemu bankowego. Świetnie sprawdzi się natomiast wśród Hindusów, którzy barwę tę utożsamiają z dobrobytem, sukcesem i bogactwem [Rau11, s. 680-681]. Nieznajomość kultury i przesadna pewność siebie projektanta może prowadzić nie tylko

19 Zob. https://www.iso.org/obp/ui/, zakładka *Graphical symbols*.

do wykonania niezrozumiałego i nieatrakcyjnego interfejsu, ale nawet obrazić odbiorców systemu. Dlatego przed wykonaniem pierwszej grafiki warto zapoznać się z literaturą opisującą różnice w komunikacji z poszczególnymi narodami, taką jak [Kuh16].

3.6. Optymalizacja dla wyszukiwarek internetowych

Modelowa lokalizacja systemu webowego ma niewielką wartość, jeżeli słaba widoczność w wynikach wyszukiwania nie pozwala zainteresowanym internautom odnaleźć jej adresu. Metodyka dążąca do uzyskania jak najwyższej widoczności nazywa się optymalizacją dla wyszukiwarek internetowych (ang. *Search Engine Optimization*, skr. SEO) [She16]. Jedna z jej technik jest związana z wielojęzykowością i polega na ustawieniu metadanych z odnośnikami do zlokalizowanej wersji każdej podstrony:

```
<head>
  ...
  <link rel="alternate" hreflang="pl" href="http://motomi.pl/oferty" />
  <link rel="alternate" hreflang="cz" href="http://motomi.cz/nabidky" />
  <link rel="alternate" hreflang="en" href="http://motomi.com/offers" />
  <link rel="alternate" hreflang="sk" href="http://motomi.com/sk/ponuky"
  />
</head>
```

Atrybut *hreflang* określa język zasobu w formacie zgodnym z regułami przedstawionymi w punkcie 3.1.1, natomiast *href* jego lokalizację. W przypadku strony o nieokreślonym języku, na przykład takiej, która sama przekierowuje do odpowiedniej wersji na podstawie adresu IP, atrybut *hreflang* może przyjąć wartość specjalną *x-default*.

Częstym błędem jest podawanie adresu strony głównej dla wszystkich podstron. Adres może być zbudowany w inny sposób, ale linki powinny prowadzić do analogicznych treści we wszystkich językach. Należy również pamiętać, aby w metadanych zawrzeć odwołanie strony do samej siebie (zabezpiecza to przed podpinaniem fałszywych, obcych witryn). Więcej informacji na temat *hreflang* można znaleźć w oficjalnym poradniku Google w serwisie YouTube [Ohy13].

3.7. Nazwa domenowa

Przejście do greckiej wersji językowej systemu po wpisaniu w przeglądarce adresu *https://αυτοκίνητα.ελ/* nadaje stronie ostatecznego szlifu i oryginalnego charakteru. Międzynarodowe nazwy domen (ang. *International Domain Names*, skr. IDN), których zapis nie jest ograniczony do wycinka zestawu znaków ASCII, to kolejne ciekawe narzędzie lokalizacji powstałe dzięki wprowadzeniu standardu Unicode. Choć w Polsce nie są zbyt popularne, to na świecie stanowią już 3% wszystkich zarejestrowanych domen, czyli kilka milionów stron. Szczególnym zainteresowaniem cieszą się w Chinach, Wietnamie, Rosji i Niemczech [Eur18].

Stosowanie międzynarodowych domen nie wymaga specjalnych zabiegów, ich rejestracja jest możliwa u dowolnego dostawcy na standardowych warunkach. Od strony technicznej polega to na zakodowaniu przez przeglądarkę adresu Unicode do postaci tzw. *Punycode*, który składa się wyłącznie ze znaków ASCII i w takiej formie jest rozgłaszany przez serwery DNS (ang. *Domain Name System*). Sam *Punycode* jest niewygodny dla człowieka, przykładowo przytoczony wcześniej grecki adres to *xn--kxadbxntl3bfd.xn--qxam*[20],

20 Dekoder Punycode jest dostępny pod adresem https://www.punycoder.com/.

który ciężko byłoby zapamiętać. Zadaniem przeglądarki jest maskowanie jego prawdziwej postaci przed użytkownikiem [Jon05].

Dwie największe wady IDN to nadal słabe wsparcie ze strony przeglądarek mobilnych oraz możliwość łatwego podszywania się pod inne strony za pomocą homogramów, czyli adresów zapisanych przy użyciu podobnych wizualnie znaków z różnych alfabetów. Mała litera a w cyrylicy i łacinie wygląda tak samo, tym samym umożliwia sfałszowanie adresu banku i wykonanie ataku *phishingowego*. Zalety IDN dostrzegą przede wszystkim posługujący się alfabetami o korzeniach innych niż łacińskie. Podsumowując, międzynarodową domenę należy traktować jako ciekawe uzupełnienie podstawowego adresu, które w przyszłości może zyskać dużą popularność, jednak obecnie powoduje istotne problemy związane z bezpieczeństwem.

3.8. Ograniczone fundusze

W wielu przypadkach, z przyczyn finansowych lub pochodnych, nie będzie możliwa lokalizacja we wszystkich językach ojczystych, jakimi posługują się użytkownicy systemu. Należy wówczas wybrać wersje priorytetowe, które trafią do największego grona odbiorców lub rozważyć przygotowanie uniwersalnego wariantu w uproszczonym podzbiorze języka angielskiego (lub pełniącego analogiczną rolę *języka kontrolowanego* [Leh98]). Zakłada on stosowanie ograniczonego słownictwa i nieskomplikowanego stylu, aby być efektywnym w nauce i zrozumiałym nawet dla osób o jego podstawowej znajomości. Reguły tzw. *Basic English* zostały przedstawione w książce Charlesa Kaya Ogdena [Ogd40], ale przygotowując taką wersję można się również posługiwać dostępnym w sieci konwerterem *Simplish*, który podpowie w jaki sposób zmienić tekst na przystępniejszy[21]. Nie można zapominać, że pomimo użycia języka angielskiego jest to specjalna wersja skierowana do odbiorcy globalnego, niekoniecznie anglosaskiego, zatem wszystkie użyte terminy i sformułowania muszą być jednoznaczne, techniczne lub naukowe, bez odniesień kulturowych, skrótów i akronimów. Przykładowo, opisanie w formularzu pól imienia i nazwiska jako *first name* i *last name* może być nieintuicyjne dla osoby wywodzącej się z kultury azjatyckiej, w której pierwsze jest zawsze nazwisko rodowe, a nie nadane podczas narodzin. W takim przypadku lepiej użyć jednoznacznych określeń *given name* i *family name*. Więcej przykładów nieprawidłowego użycia terminologii można znaleźć w wytycznych podanych przez firmę IBM w [Ibm16b].

21 Zob. https://www.simplish.org/.

4. Strategie internacjonalizacji

W tym rozdziale przedstawione zostaną wybrane techniki internacjonalizacji interfejsów wielojęzykowych systemów webowych. Opisane metody zostaną ze sobą porównane i ocenione.

4.1. Język a struktura adresu

Niezależnie od wybranego podejścia do internacjonalizacji, każda strona potrzebuje adresu internetowego. Do poszczególnych wersji językowych kierować można na kilka charakterystycznych sposobów opisanych poniżej (za [Mue10, Mor16]). Struktura adresu URL podyktowana jest strategią internacjonalizacji, rozmiarem strony, możliwościami technicznymi konfiguracji serwera, grupą docelową, a także przyjętą strategią biznesową i wymogami prawnymi.

4.1.1. Domena krajowa

Przykładowe strony określające język za pomocą krajowej domeny najwyższego poziomu (ang. *Country Code Top-Level Domain*, skr. ccTLD):

1. https://motomi.pl/ (region Polska),
2. https://honda.fr/ (region Francja),
3. https://www.rakuten.co.jp/ (region Japonia).

Adres w takiej postaci oznacza, że strona jest ukierunkowana geopolitycznie na odbiorców z konkretnego regionu. Pośrednio implikuje to również język, jednak nie można zakładać, że system będzie tak samo odpowiedni dla użytkowników w każdym kraju, który się nim posługuje. Przykładowo, zarówno w Australii, jak i w Wielkiej Brytanii używa się języka angielskiego, jednak odmienne pory roku oznaczają, że sklep sportowy w domenie *.au* będzie promował narty, w czasie gdy jego brytyjski odpowiednik *.co.uk* postawi na rowery. Nie jest to zatem rozwiązanie uniwersalne, tylko przeznaczone wyłącznie dla odbiorców w określonym kraju.

Zaletami użycia domeny krajowej są: izolacja poszczególnych wersji językowych, precyzyjna geolokalizacja dla wyszukiwarek internetowych, wysoka wiarygodność adresu dla użytkownika i – w niektórych przypadkach – zgodność z wymogami prawnymi. Do wad należy zaliczyć głównie koszty i rozbudowaną infrastrukturę. Wymagane jest przygotowanie treści dla kilku niezależnych systemów webowych, rejestracja osobnych domen i utrzymanie oddzielnych serwerów.

4.1.2. Poddomena

Przykładowe strony określające język za pomocą poddomeny w ramach domeny funkcjonalnej (ang. *Generic Top-Level Domain*, skr. gTLD):

1. https://pl.wikipedia.com/ (język polski),
2. https://pt.louisvuitton.com/ (język portugalski),
3. https://es.slideshare.net/ (język hiszpański).

Adres w takiej postaci definiuje język systemu, rzadziej kraj. Ta niejednoznaczność jest główną wadą rozwiązania – w pewnych sytuacjach użytkownik ani wyszukiwarka internetowa nie będą pewni, czy link wskazuje uniwersalną wersję systemu w podanym języku, czy też wersję przeznaczoną celowo na rynek konkretnego kraju. Przykładowo, kod języka niemieckiego (*de*) jest taki sam jak kod Niemiec (*DE*). Zaletą tego podejścia jest natomiast izolacja poszczególnych wersji językowych bez konieczności utrzymania wielu domen. Pozostałe cechy będą podobne jak w przypadku użycia domeny krajowej.

4.1.3. Podkatalog

Przykładowe strony określające język za pomocą podkatalogu w ramach domeny funkcjonalnej gTLD:

1. https://shop.lego.com/pl-PL/ (region Polska, język polski),
2. https://www.siemens.com/ch/de/ (region Szwajcaria, język niemiecki),
3. https://www.nhl.com/fi/ (język fiński).

Adres w takiej postaci pozwala określić zarówno język, jak i region dla którego przeznaczony jest system. Tym samym jest to rozwiązanie, w porównaniu do poprzednich, uniwersalne i jednoznaczne. Kolejną zaletą jest prostota infrastruktury – poszczególne wersje językowe działają w ramach jednej aplikacji, na tej samej maszynie. Jednocześnie stanowić może to wadę, ponieważ lokalizacje są od siebie zależne, a ich rozdział będzie utrudniony. Niepożądane może być również obciążenie pojedynczej maszyny ruchem ze wszystkich obsługiwanych języków, jednak w dobie automatycznie skalowanych rozwiązań chmurowych argument ten traci na znaczeniu.

4.1.4. Parametr

Przykładowe strony określające język za pomocą dodatkowych parametrów podanych w adresie URL:

1. https://twitter.com/?lang=de (język niemiecki),
2. http://www.farming-simulator.com/?lang=fr&country=ca (region Kanada, język francuski),
3. https://www.edmodo.com/?language=el (język grecki).

Adres w takiej postaci, choć nadal spotykany, jest odradzany z uwagi na tymczasowość i brak powiązania parametru z faktyczną strukturą strony. Można go podać nawet dla nieistniejącego zasobu lub skasować bez wpływu na poprawność semantyczną linku, co w konsekwencji rodzi problemy podczas pozycjonowania systemu w wyszukiwarkach internetowych. Ponadto użycie parametrów wydłuża niepotrzebnie adres i zmniejsza jego czytelność (np. */?lang=fr&country=ca* zamiast zgodnego ze standardem ISO formatu */fr-CA/*).

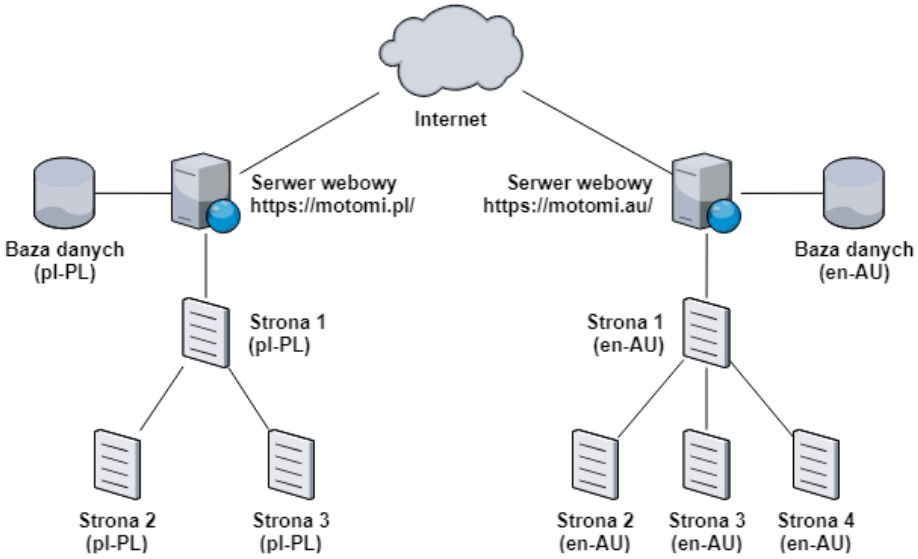
4.1.5. Ciasteczko

Ostatnim sposobem, niemającym wpływu na adres, jest zapisanie ustawień regionalnych w ciasteczku (ang. *cookie*) lub w magazynie lokalnym (ang. *local storage*) po stronie przeglądarki użytkownika. Uzależnienie dostępności zasobów w danym języku od stanu wewnętrznego klienta jest oczywistym utrudnieniem dla robotów internetowych i podczas udostępniania linków. Takie rozwiązanie sprawdzi się głównie w przypadku

serwisów opartych o strumieniowanie wideo, w których właściwą treść stanowi materiał multimedialny, a ustawienie języka wpływa jedynie na elementy interfejsu.

4.2. Osobna instancja dla każdego języka

Pierwszą z wybranych strategii internacjonalizacji jest utworzenie osobnych instancji systemu dla każdego języka. Podejście to zakłada współistnienie niezależnych wersji strony, z których każda dedykowana jest innej lokalizacji [Hil02, s. 5]. Rozwiązanie to w naturalny sposób łączy się z adresami internetowymi, które określają region za pomocą krajowej domeny najwyższego poziomu lub poddomeny w ramach domeny funkcjonalnej (patrz punkty 4.1.1 i 4.1.2). System tworzony jest z myślą o konkretnym języku i nie współdzieli żadnych zasobów z pozostałymi wersjami, będąc od nich całkowicie niezależnym [San05, s. 135]. Wymaga to poniesienia dodatkowych kosztów związanych z przygotowaniem merytorycznych treści (a nie tylko przetłumaczeniem istniejących), opracowaniem indywidualnych projektów graficznych oraz utrzymaniem niezależnej infrastruktury sieciowej, z drugiej jednak strony jest to jedyne podejście, które pozwala na utworzenie witryny idealnie dopasowanej do potrzeb i preferencji lokalnego odbiorcy bez stosowania kompromisów i międzynarodowych uogólnień (aspekty kulturowe omówiono szerzej w podrozdziale 3.1). Takie rozwiązanie sprawdzi się, gdy poszczególne lokalizacje mocno od siebie odbiegają (np. bankowość elektroniczna w Polsce i w Indiach) lub są przygotowywane i zarządzane przez niezależne zespoły (np. strona marki samochodowej, za którą w imieniu producenta jest odpowiedzialny oficjalny dystrybutor, inny w każdym kraju).

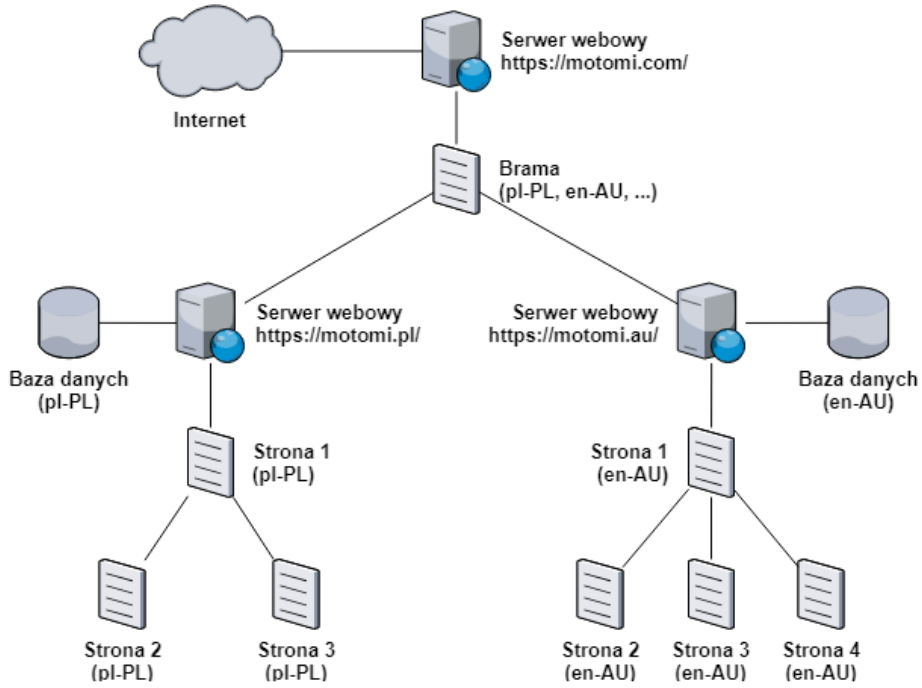


Rys. 4.1. Schemat systemu z osobnymi instancjami dla każdego języka.
Źródło: opracowanie własne na podstawie [Hil02, s. 3].

Od strony technicznej system taki nie musi zawierać żadnych zaawansowanych mechanizmów internacjonalizacji, ponieważ wszystkie wersje językowe są od siebie odizolowane (patrz Rys. 4.1), co pozwala traktować i pozycjonować poszczególne witryny indywidualnie. Elementy podlegające lokalizacji są ustawione na sztywno lub w jakiś sposób scalone z warstwą prezentacji [Fil05, s. 189]. Teksty nie są zatem wydzielone do kluczy tłumaczeń, tylko zapisane bezpośrednio w kodzie HTML, wartości liczbowe przechowuje się

wprost w docelowym formacie, bez konieczności konwersji do uniwersalnej postaci, a grafiki zawierają dużą liczbę symboli i odniesień regionalnych, gdzie indziej niezrozumiałych.

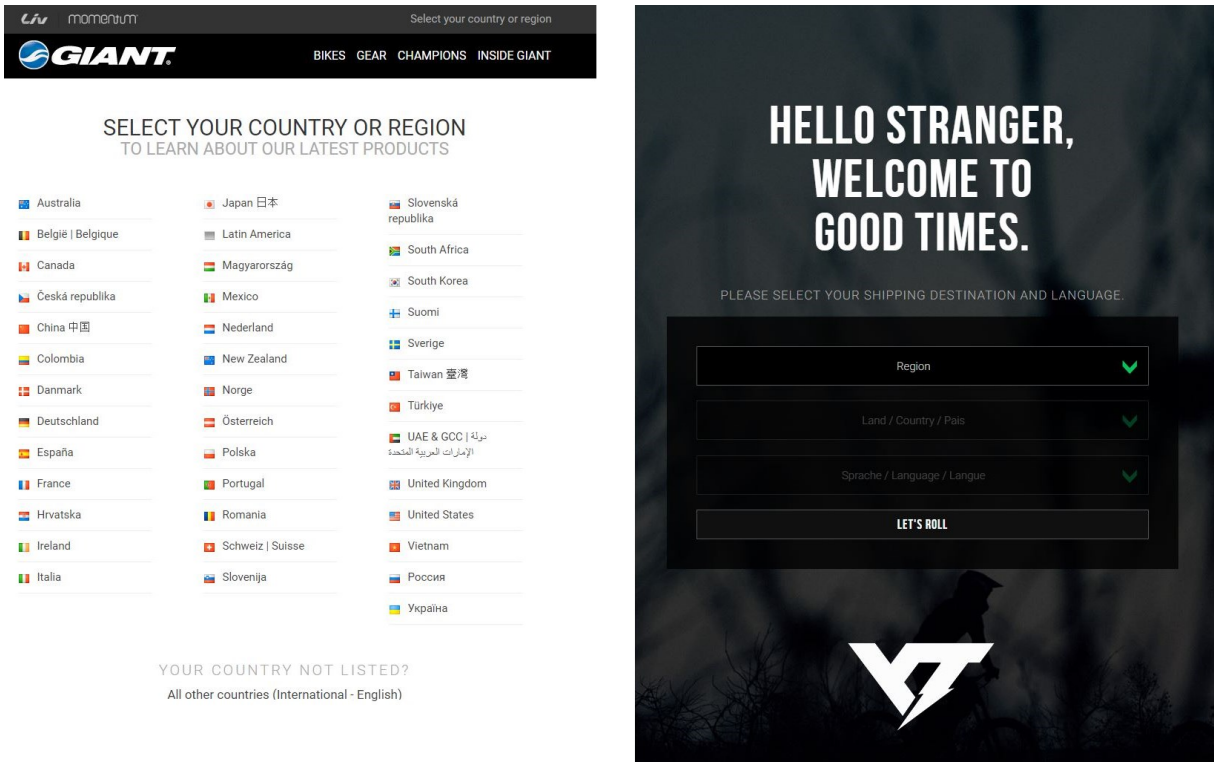
Stosowaną niekiedy modyfikacją opisywanego rozwiązania jest dodanie kosmopolitycznej bramy (ang. *global gateway*), która rozdziela ruch pomiędzy poszczególne instancje. Przekierowanie odbywa się automatycznie podczas negocjacji treści z serwerem (na podstawie adresu IP, ustawień przeglądarki i nagłówka *HTTP_ACCEPT_LANGUAGE*) albo ręcznie w oparciu o wybór użytkownika [Yun03, Building the Global Gateway]. Brama może być udostępniona na dedykowanej maszynie lub w ramach głównej wersji językowej systemu (np. angielskiej w domenie *.com*). Schemat takiego wariantu przedstawiono na Rys. 4.2.



Rys. 4.2. Schemat systemu z osobnymi instancjami dla każdego języka i bramą. Źródło: opracowanie własne na podstawie [Hil02, s. 3].

Podczas projektowania bramy trzeba pamiętać, że korzystać będą z niej osoby różnojęzyczne, konieczne jest zatem utrzymanie prostej stylistyki, która pozwoli użytkownikom szybko odnaleźć właściwy język lub region. Wszystkie opcje powinny być przetłumaczone i widoczne jednocześnie. Błędem jest pozostawienie anglojęzycznych nazw krajów czy ukrycie dostępnych języków pod postacią rozwijanych menu, ponieważ dla internauty, który nie zna języka angielskiego przeznaczenie takiej strony może wtedy nie być jasne [Yun03, The Art of the Global Gateway]. W takiej sytuacji dobrym pomysłem jest wzbogacenie tekstu o uniwersalną symbolikę – wykorzystać można na przykład zarys interaktywnej mapy świata, która pozwoli odwiedzającym wybrać właściwy dla nich region. Dwa przykłady implementacji bramy przez międzynarodowych producentów rowerów są widoczne na Rys. 4.3. Giant (po lewej) wykonał swoją stronę niemal wzorowo trzymając się opisywanych zaleceń. Odnośniki do wszystkich lokalizacji są widoczne jednocześnie, dzięki czemu łatwo zrozumieć przeznaczenie bramy, nazwy regionów zostały przetłumaczone, a w przypadku krajów wielojęzycznych użyto kilku oficjalnych języków urzędowych (np. Belgia, Szwajcaria). Dla pozostałych użytkowników przygotowano uniwersalną wersję

w języku angielskim. Modelowa realizacja zawierałaby również nagłówek strony zapisany w kilku najpopularniejszych językach świata, nie jest to jednak uchybienie uniemożliwiające zrozumienie idei bramy. Podręcznikowym przykładem błędnego projektu jest natomiast strona YT Industries (po prawej), która choć wygląda efektownie, została przygotowana w całości po angielsku. Ponadto wybór języka poprzedza tu konieczność skorzystania z kilku list rozwijanych, które i tak za każdym razem zawężają decyzję do trzech tych samych opcji.



Rys. 4.3. Różne podejścia do projektu międzynarodowej bramy systemu.

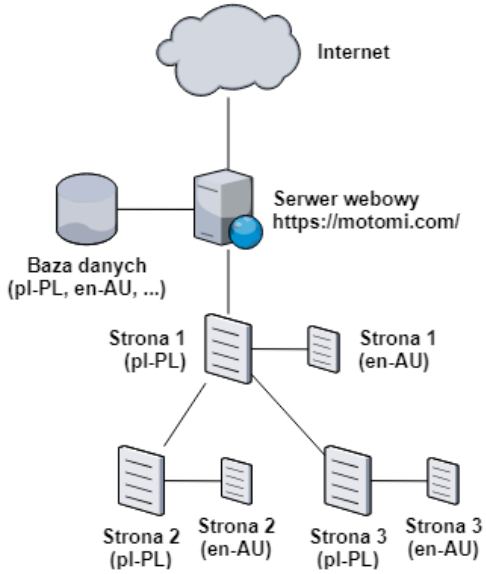
W momencie rozpoczęcia niniejszej analizy, internacjonalizacja przez lokalizację aplikacji *na sztywno* była wdrożona przez twórców badanego systemu *Motomi*.

4.3. Dynamiczna podmiana języka

Drugą z wybranych strategii internacjonalizacji jest utworzenie systemu, który pozwala na dynamiczne przełączanie języka. Podejście to zakłada istnienie pojedynczej witryny, która posiada mechanizmy umożliwiające podmianę napisów lub innej zawartości w locie [Hil02, s. 3-4]. Rozwiązanie to w naturalny sposób łączy się z adresami internetowymi, które określają lokalizację za pomocą podkatalogu w ramach domeny funkcjonalnej, parametru w adresie URL lub ciasteczka (patrz punkty 4.1.3, 4.1.4 i 4.1.5). System tworzony jest z myślą o przystosowaniu do potrzeb wielu języków, które wykorzystują te same zasoby. Pozwala to ograniczyć wydatki związane z przygotowaniem treści – można je utworzyć raz, centralnie, a następnie przetłumaczyć [San05, s. 135], będą one jednak pełne międzynarodowych uogólnień, bez regionalnego charakteru i odniesień. Konieczność współdzielenia materiałów, takich jak tekst i grafiki, wymusza zachowanie spójności między poszczególnymi wersjami językowymi. Pomaga w tym przygotowanie globalnego szablonu ze wspólnym układem elementów, nawigacją i schematem kolorów [Yun03, Building the Global Template]. Takie rozwiązanie sprawdzi się zatem, gdy poszczególne lokalizacje nie odbiegają od siebie w istotny sposób pod względem treści

i mogą przyjąć podobny układ stron (np. witryna wypożyczalni samochodów przeznaczona dla bliskich sobie kulturowo i prawnie mieszkańców Stanów Zjednoczonych i Kanady).

Od strony technicznej system taki jest bardziej skomplikowany niż rozwiązanie z osobnymi instancjami dla każdego języka, które mechanizmy internacjonalizacji wdrażały już na poziomie infrastruktury sieciowej. W tym przypadku rozdział poszczególnych wersji odbywa się dynamicznie dzięki technikom programistycznym (patrz Rys. 4.4). Dąży się zatem do jak najlepszej izolacji warstw danych i prezentacji: miejsca w których pojawia się tekst wskazują abstrakcyjne klucze tłumaczeń, wartości liczbowe przechowywane są w uniwersalnej postaci i formatowane (lub przeliczane) w locie przez interfejs, a grafiki i symbole przygotowuje się w sposób możliwie neutralny kulturowo. Ponieważ przy takim podejściu ruch ze wszystkich krajów jest kierowany do tego samego węzła, może spowodować to jego przeciążenie lub duże opóźnienia odpowiedzi dla użytkowników znajdujących się w odległość wielu skoków od serwera. Aby im przeciwdziałać należy rozważyć wykorzystanie możliwości oferowanych przez chmury obliczeniowe i sieci CDN (ang. *Content Delivery Network*), które pozwalają rozproszyć aplikację i przybliżyć fizyczną lokalizację hosta do odbiorcy.



Rys. 4.4. Schemat systemu z dynamiczną podmianą języka.
Źródło: opracowanie własne na podstawie [Hil02, s. 3].

Podstawową implementację opisywanej strategii internacjonalizacji można uzyskać przez umieszczenie zlokalizowanych wersji w osobnych podkatalogach i połączenie tożsamych dokumentów linkami [Fil05, s. 190]. Odnośniki i nazwy ścieżek mogą zostać przetłumaczone, jednak dla prawidłowego pozycjonowania przez wyszukiwarki internetowe należy zachować analogiczną hierarchię plików, jak w przykładzie poniżej (alternatywnie można zastosować atrybut *hreflang* omówiony w podrozdziale 3.6):

https://motomi.com/pl-PL/oferty-sprzedazy/samochody/

https://motomi.com/en-AU/trade-offers/cars/

Docelowa implementacja zakłada jednak przygotowanie uniwersalnego szablonu, który w oznaczone miejsca, na podstawie parametru, wczytuje z bazy danych treści przygotowane dla właściwej lokalizacji. Dzięki językowi JavaScript podmiana zawartości może następować

dynamicznie w przeglądarce internetowej po stronie klienta, bez konieczności przeładowywania strony. Wybór języka jest możliwy za pomocą selektora umieszczonego w widocznym miejscu na każdej podstronie. Podczas jego projektowania obowiązują zasady podobne jak w przypadku bramy opisanej w podrozdziale 4.2: wszystkie opcje powinny być łatwo dostępne, posortowane i wyświetlane we właściwym sobie języku [Ver15]. Jeżeli poszczególne warianty systemu nie zostały przygotowane specjalnie z myślą o danych krajach, nie należy stosować popularnych flag narodowych, ponieważ nie są one tożsame z konkretnym językiem (ten może być wykorzystywany w różnych częściach świata, jak hiszpański, lub być tylko jednym z kilku oficjalnych języków w danym kraju, jak francuski w Kanadzie). Niewielka grafika powoduje również problemy z czytelnością, rozróżnieniem podobnych flag (np. Włoch i Irlandii), dostępnością dla niepełnosprawnych [Vaz15] czy szybkim wyszukaniem języka w treści strony, które jest obecnie możliwe tylko dla danych tekstowych. Modelowy przykład selektora spełniającego przytoczone zasady stosuje w swoich artykułach Wikipedia (patrz Rys. 4.5).

Rys. 4.5. Selektor języka w serwisie Wikipedia.

Strategia dynamicznej podmiany języka została wybrana do wdrożenia w badanym systemie *Motomi* i zostanie szczegółowo opisana w rozdziale 5.

4.4. Tłumaczenie maszynowe

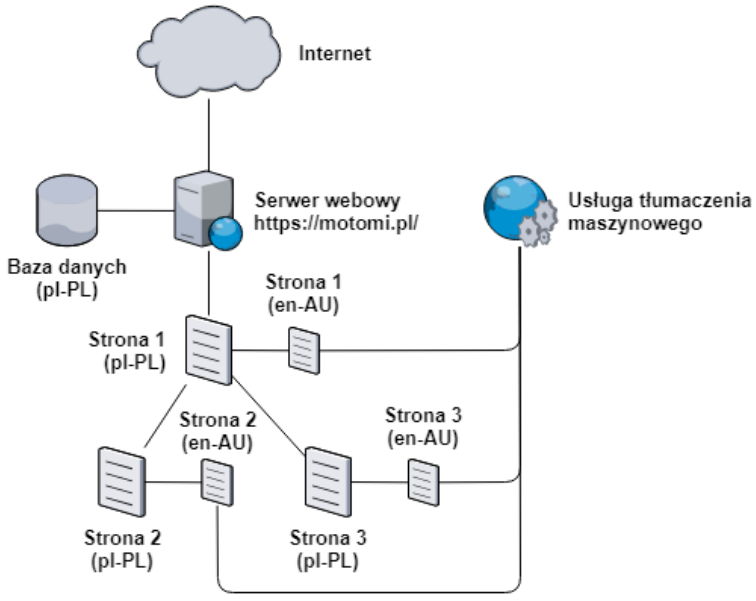
Ostatnią z wybranych strategii internacjonalizacji jest wykorzystanie mechanizmu tłumaczenia automatycznego dostarczanego przez firmę trzecią. Podejście to zakłada przygotowanie wyłącznie jednej, podstawowej wersji językowej systemu, która na żądanie jest tłumaczona maszynowo w całości lub w wybranych fragmentach. Rozwiązanie to łączy się z adresami internetowymi o dowolnym schemacie, gdyż translacja odbywa się na serwerach dostawcy zewnętrznego, w oparciu o wyrenderowaną wcześniej stronę. Jest to metoda szybka i tania, ponieważ nie wymaga poniesienia kosztów związanych z przygotowaniem dodatkowej infrastruktury ani treści, a podczas projektowania nie są konieczne żadne specjalne zabiegi przygotowawcze – wystarczy parę kliknięć na dowolnym etapie prac, aby otrzymać system komunikujący się z użytkownikiem w kilkudziesięciu językach[1]. Brzmi to obiecująco, jednak pomimo postępu jaki został dokonany w ostatnich

1 Zob. https://www.w3schools.com/howto/howto_google_translate.asp.

latach w dziedzinie przetwarzania języka naturalnego, nie należy spodziewać się wysokiej jakości tłumaczeń, zwłaszcza w przypadku krótkich fragmentów tekstu oderwanych od kontekstu lub przeplatanych specjalistyczną terminologią [Smi06]. Ponadto tego typu lokalizacji poddany może być wyłącznie tekst, a wszelkie elementy graficzne i symbole będą ignorowane. Jednostki miar i czasu pozostaną w oryginalnej formie, a projektant nie będzie mieć możliwości dopasowania poszczególnych wersji do specyficznych potrzeb kulturowych odbiorców. Mimo wspomnianych wad rozwiązanie to sprawdzi się w co najmniej kilku sytuacjach:

1. awaryjnie, gdy fundusze nie pozwalają na użycie innych metod, a dla użytkownika bardziej przystępne będzie niedoskonałe tłumaczenie maszynowe niż egzotyczne pismo (np. tajskie dla europejczyka),
2. do tworzenia prototypów, które pomogą szybko przetestować system pod kątem zgodności z danym językiem i pozwolą zaprezentować jego próbną wersję klientom,
3. jako tłumaczenie wstępne, dzięki któremu człowiek będzie mógł ograniczyć swoją pracę do recenzji i poprawek gotowych tekstów, zamiast zaczynać od zera,
4. do translacji treści niezależnych od twórców danego systemu, np. komentarzy wprowadzanych przez użytkowników.

Rozwiązania tego typu dostarczają m.in.: *SiteTran*[2], *Google Translate*[3] oraz *Microsoft Bing Translator*[4], które w zależności od polityki firmy mogą być płatne lub – przy niewielkim ruchu albo kosztem reklam – darmowe.

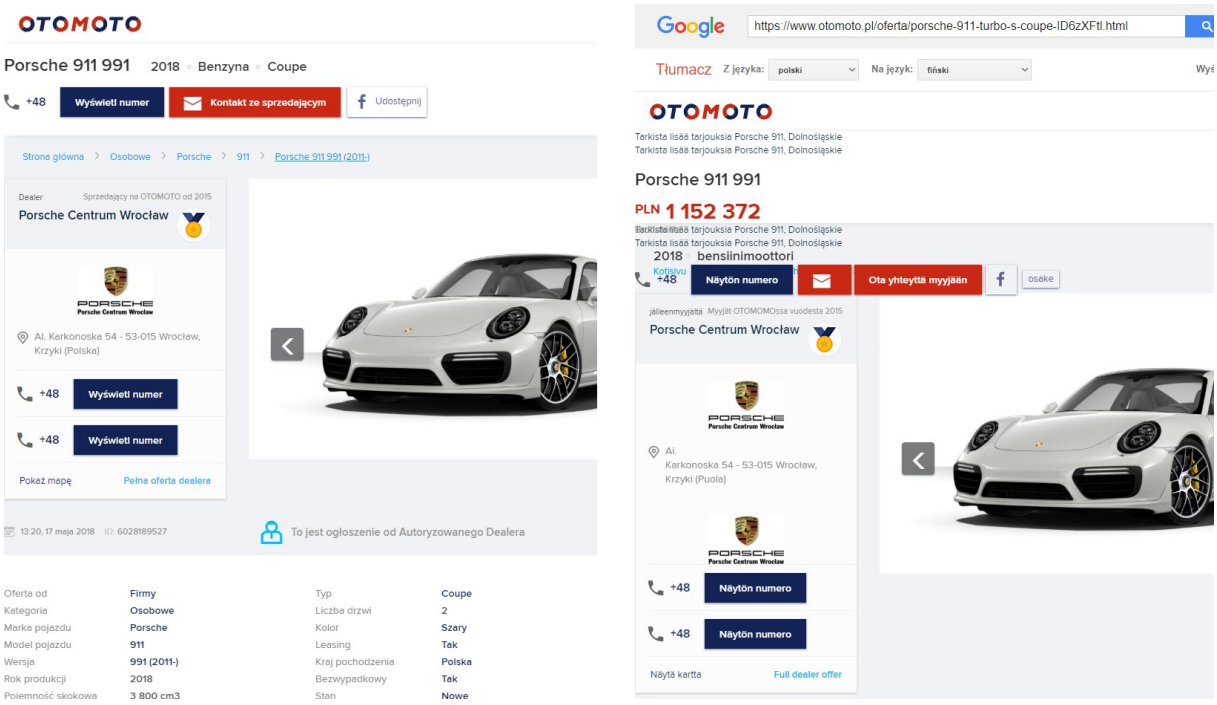


Rys. 4.6. Schemat systemu z zewnętrznym tłumaczeniem maszynowym. Źródło: opracowanie własne.

Od strony technicznej wdrożenie opisywanej strategii internacjonalizacji następuje przez umieszczenie w kodzie skryptu firmy trzeciej lub bezpośrednią integrację z jej interfejsem programistycznym. Dostawca tłumaczeń przejmuje zarządzanie renderowaną

2 Zob. http://www.sitetran.com/.
3 Zob. https://cloud.google.com/translate/.
4 Zob. https://www.microsoft.com/en-us/translator/web.aspx.

treścią przez osadzenie docelowej strony w pływającej ramce HTML (ang. *inline frame*), która pozwala użytkownikom na przełączanie się pomiędzy poszczególnymi lokalizacjami. Wariant oryginalny prezentowany jest bez zmian, natomiast w przypadku pozostałych języków następuje pobranie odpowiednio przetworzonej wersji danej podstrony z usługi zewnętrznej i wstawienie jej zawartości do ramki (patrz Rys. 4.6). Niestety, ramki bywają rozwiązaniem problematycznym, które powoduje błędy w wyświetlaniu, co można zaobserwować na Rys. 4.7. Elementy interfejsu systemu *Otomoto* zmieniły położenie i zaczęły na siebie nachodzić uniemożliwiając nawigację, pojawiły się także artefakty w postaci powtarzających się grafik, reklam i przycisków.



Rys. 4.7. Oryginalna (po lewej) i przetłumaczona maszynowo (po prawej) strona Otomoto.

W przypadku badanego systemu *Motomi*, wdrożenie opisywanego rozwiązania nie powiodło się, ponieważ wykorzystywane w nim biblioteki i narzędzia programistyczne nie są kompatybilne z technologią ramek (w efekcie aplikacja nie uruchamia się, a ekran pozostaje pusty).

4.5. Porównanie metod

Podczas porównywania przedstawionych w tym rozdziale technik internacjonalizacji przyjęto następujące kryteria:

1. łatwość pozycjonowania, czyli podatność rozwiązania na wprowadzenie metod optymalizacji dla wyszukiwarek internetowych i bycie znalezionym,
2. prostotę lokalizacji, inaczej nakład pracy konieczny do rozbudowania systemu o kolejne języki,
3. poziom nadzoru, rozumiany jako zdolność do zarządzania spójnością i treścią poszczególnych wersji językowych,
4. format adresu internetowego, kompatybilny z wybraną techniką internacjonalizacji,

5. jakość tłumaczenia, poprawność gramatyczną i zgodność tekstów z obowiązującymi regułami pisowni,
6. wrażliwość na aspekty kulturowe, to znaczy możliwość dopasowania zawartości i interfejsu do specyfiki i charakteru lokalnego odbiorcy,
7. wydajność systemu, to jest zdolność do obsłużenia międzynarodowego ruchu pochodzącego z różnych stron świata,
8. złożoność infrastruktury sieciowej, a zatem liczbę serwerów, domen, itd.,
9. narzut kodu źródłowego, koniecznego do przełączania języka i konwersji danych,
10. koszt wdrożenia, inaczej mówiąc wysokość nakładów finansowych jakie trzeba ponieść, aby wprowadzić wybrane rozwiązanie,
11. koszt utrzymania, a więc środki jakie należy regularnie przeznaczać na podtrzymanie funkcjonowania gotowego systemu,
12. czas przygotowania, poświęcony na wykonanie pierwszej działającej wersji internacjonalizowanej aplikacji,
13. izolację zasobów, innymi słowy stopień wykorzystania współdzielonych materiałów,
14. popularne problemy występujące podczas implementacji danej strategii.

Wyniki porównania zostały zebrane poniżej w Tab. 4.1.

Tab. 4.1. Porównanie wybranych strategii internacjonalizacji.

Kryterium	Osobna instancja dla każdego języka	Dynamiczna podmiana języka	Tłumaczenie maszynowe
Łatwość pozycjonowania	**Wysoka.** Ścisłe ukierunkowanie na konkretny region i dedykowany adres internetowy pozwalają na użycie narzędzi oraz technik SEO w standardowy sposób.	**Niska.** Treść generowana dynamicznie po stronie klienta może być problematyczna dla robota indeksującego. Konieczne przygotowanie prerenderowanej wersji systemu[5].	**Niska.** Treść generowana automatycznie powoduje przyznawanie kar w algorytmach wyszukiwarek internetowych[6].
Prostota lokalizacji	**Niska.** Wymaga przygotowania nowych treści od zera oraz skonfigurowania dodatkowej infrastruktury.	**Przeciętna.** Wymaga przetłumaczenia istniejących już treści oraz dodania ustawień regionalnych dla nowego języka.	**Wysoka.** Jeżeli dany język jest wspierany przez dostawcę tłumaczeń, wystarczy wybrać go z listy. Inaczej dodanie nowej lokalizacji będzie niemożliwe.

5 Zob. https://prerender.io/.
6 Zob. http://www.translateplus.com/blog/google-deal-translated-content/.

Kryterium	Osobna instancja dla każdego języka	Dynamiczna podmiana języka	Tłumaczenie maszynowe
Poziom nadzoru	**Przeciętny.** Poszczególne instancje mogą od siebie mocno odbiegać i być zarządzane przez różne zespoły.	**Wysoki.** Materiały przygotowywane są centralnie, a wszystkie lokalizacje funkcjonują w ramach jednej aplikacji.	**Niski.** Zlokalizowana treść zależy od dostawcy zewnętrznego, który może dowolnie modyfikować zawartość strony i dodawać do niej swoje elementy (np. banery reklamowe).
Format adresu internetowego	**Ograniczony:** • domena krajowa, • poddomena.	**Ograniczony:** • podkatalog, • parametr, • ciasteczko.	**Dowolny.**
Jakość tłumaczenia	**Wysoka.** Tłumaczenia przygotowywane przez człowieka.	**Wysoka.** Tłumaczenia przygotowywane przez człowieka.	**Niska.** Tłumaczenia przygotowywane maszynowo.
Wrażliwość na aspekty kulturowe	**Wysoka.** Każda instancja przygotowywana jest z myślą o odbiorcy lokalnym i konkretnym regionie.	**Przeciętna.** Większość treści ma charakter uniwersalny, międzynarodowy. Możliwa jest pewna elastyczność w ramach tłumaczonego tekstu.	**Niska.** Generowana treść jest dosłowna i nie uwzględnia aspektów kulturowych.
Wydajność systemu	**Wysoka.** Każda instancja obsługuje inny region świata.	**Niska.** Pojedyncza aplikacja obsługuje ruch z całego świata. Konieczne użycie mechanizmów CDN.	**Przeciętna.** Pojedyncza aplikacja obsługuje ruch z całego świata, jednak osadzenie w ramce dostawcy tłumaczeń pozwala na buforowanie zapytań.
Złożoność infrastruktury sieciowej	**Wysoka.** Każda instancja wymaga własnej domeny i serwera.	**Niska.** Pojedyncza domena i serwer.	**Niska.** Pojedyncza domena i serwer.
Narzut kodu źródłowego	**Niski.** Zmiana języka odbywa się za pomocą zwykłych linków	**Wysoki.** Wymagane jest samodzielne zaprogramowanie mechanizmów	**Niski.** Za przełączanie języka odpowiada skrypt firmy trzeciej.

Kryterium	Osobna instancja dla każdego języka	Dynamiczna podmiana języka	Tłumaczenie maszynowe
	i przekierowań.	pozwalających na współistnienie wielojęzycznej treści w ramach pojedynczej aplikacji.	
Koszt wdrożenia	**Wysoki.** Wymaga zaprojektowania kilku niezależnych aplikacji od podstaw.	**Przeciętny.** Przygotowanie globalnego szablonu pozwala ograniczyć koszty i współdzielić materiały.	**Niski.** Wymaga jedynie umieszczenia w systemie skryptu firmy trzeciej.
Koszt utrzymania	**Wysoki.** Konieczne utrzymanie zespołów redagujących nowe treści oraz rozbudowanej infrastruktury sieciowej.	**Przeciętny.** Wymaga opłacenia tłumaczy, gdy przygotowana zostanie aktualizacja materiałów.	**Niski.** Zależnie od wybranego planu około 10-20 dolarów na każdy milion znaków lub nawet za darmo[7].
Czas przygotowania	**Długi.** Wymaga przygotowania treści oddzielnie dla każdej instancji, co jest czasochłonne.	**Przeciętny.** Treść przygotowywana jest jednokrotnie, a następnie tłumaczona na wszystkie języki.	**Krótki.** Tłumaczenia są gotowe natychmiast.
Izolacja zasobów	**Wysoka.** Każda instancja posiada niezależny zestaw grafik, symboli, tekstów i plików.	**Niska.** Dąży się do wielokrotnego współdzielenia materiałów pomiędzy poszczególnymi wersjami językowymi.	**Niska.** Oryginalny tekst jest ścisłą bazą dla tłumaczeń na pozostałe języki.
Popularne problemy	Koszty rozwiązania i postępująca utrata spójności danych mogą przekroczyć możliwości finansowe i organizacyjne przeciętnej firmy.	Dostosowanie globalnego szablonu do potrzeb wszystkich obsługiwanych języków może okazać się zadaniem nietrywialnym.	Stosowana przez dostawców tłumaczeń technologia ramek może uniemożliwić implementację rozwiązania w systemie.

Źródło: opracowanie własne.

7 Zob. https://cloud.google.com/translate/pricing i https://azure.microsoft.com/en-us/pricing/details/cognitive-services/translator-text-api/. Dla porównania strona znormalizowanego maszynopisu ma 1800 znaków.

Jak widać po zestawieniu przygotowanym w powyższej tabeli, nie można jednoznacznie wskazać najlepszej lub najgorszej strategii internacjonalizacji. Osobna instancja dla każdego języka to podejście drogie i czasochłonne, ale pozwalające na przygotowanie systemu w pełni oddającego lokalny charakter kulturowy. Jego przeciwieństwem zdaje się być tłumaczenie maszynowe w formie usługi dostarczanej przez firmę trzecią. Jest tanie i daje natychmiastowe rezultaty, jednak jakość lokalizacji w tym przypadku pozostawia wiele do życzenia. Dynamiczna podmiana języka w ramach jednej aplikacji jest rozwiązaniem pośrednim, ale nowoczesnym technicznie. Sprawdzi się w przypadku systemów korporacyjnych, które wymagają spójności treści i przekazu na całym świecie, przy zachowaniu wysokiej jakości tłumaczenia.

5. Wdrożenie mechanizmów wielojęzyczności

W tym rozdziale przedstawiona zostanie praktyczna implementacja mechanizmów wielojęzyczności jaką zrealizowano w systemie webowym *Motomi*. Opisane rozwiązanie zostanie następnie zweryfikowane i ocenione.

5.1. Przegląd systemu Motomi

System webowy *Motomi* jest przygotowanym wyłącznie w języku polskim serwisem umożliwiającym publikowanie i przeglądanie ogłoszeń motoryzacyjnych w Internecie. Program został zaprojektowany w architekturze klient-serwer i działa w oparciu o model bogatej aplikacji internetowej (ang. *Rich Internet Application*, skr. RIA). Aplikacja serwerowa powstała na bazie szkieletu *Spring Boot 1.3.0*[1] funkcjonującego w środowisku wykonawczym Java 8[2] połączonym z bazą danych *MySQL 5.6*[3]. Do implementacji klienta wykorzystano platformę *AngularJS 1.4.8*[4] i język JavaScript w wersji *ECMAScript 6*[5]. Elementy interfejsu graficznego zbudowano używając preprocesora stylów *SCSS*[6] i zmodyfikowanych komponentów dostarczanych przez bibliotekę *Bootstrap 3*[7].

5.1.1. Bogata aplikacja internetowa

Badany system został wdrożony jako gruby klient (ang. *fat client*), spełnia również założenia koncepcji SPA (ang. *Single Page Application*), czyli pojedynczej strony webowej, która nie wymaga przeładowywania podczas przeglądania zawartości [Sco15] i zachowuje się jak zwykła aplikacja komputerowa [Dei08]. Interfejs renderowany jest w pełni po stronie klienta, zamiast w klasyczny sposób na serwerze, a komunikacja została ograniczona do przesyłu właściwych danych w formacie JSON (patrz punkt 2.5.2), zamiast kompletnych stron webowych. Wykonywanie asynchronicznych zapytań *XMLHttpRequest* (skr. XHR) umożliwia natychmiastowe odświeżanie konkretnych fragmentów drzewa DOM (ang. *Document Object Model*) w momencie nadejścia odpowiedzi od serwera lub po wykonaniu akcji przez użytkownika.

5.1.2. Platforma AngularJS

Platformą, która umożliwiła zbudowanie aplikacji zgodnej z opisywanymi koncepcjami RIA/SPA jest *AngularJS*. Jej pierwsza wersja powstała w 2009 roku jako projekt wewnętrzny firmy Google, jednak obecnie jest dystrybuowana za darmo na zasadach wolnego oprogramowania. Podczas realizacji swoich zadań posługuje się ona wzorcem projektowym model-widok-kontroler (ang. *Model-View-Controller*, skr. MVC) rozszerzonym o szereg komponentów ułatwiających manipulowanie dynamiczną zawartością interfejsu, takich jak szablony, dyrektywy, filtry i serwisy. Ponadto udostępnia mechanizmy: obsługi nawigacji, wstrzykiwania zależności i dwukierunkowego wiązania danych między modelem a widokiem (zmiana w modelu automatycznie aktualizuje widok, a zmiana w widoku – model). Więcej na temat podstaw platformy *AngularJS* można przeczytać w [Moo16].

1 Zob. https://spring.io/blog/2015/11/16/spring-boot-1-3-0-released.
2 Zob. https://www.java.com/pl/download/faq/java8.xml.
3 Zob. https://dev.mysql.com/doc/refman/5.6/en/mysql-nutshell.html.
4 Zob. https://angularjs.org/.
5 Zob. http://es6-features.org/.
6 Zob. https://sass-lang.com/guide.
7 Zob. https://getbootstrap.com/docs/3.3/.

5.1.3. Zastany poziom internacjonalizacji

Mechanizmy internacjonalizacji zastane w badanym systemie najbliższe były jednej z lokalnych instancji systemu z osobnymi kopiami dla każdego języka, ponieważ ukierunkowano je wyłącznie na odbiorcę polskojęzycznego. Na stałe zakodowano tekst:

```
<strong>Pokaż pojazdy</strong>
```

Linki:

```
<a href="oferty/">Wszystkie samochody</a>
```

Format daty:

```
<span>{{ entry.date | date: 'dd.MM.yyyy' }}</span>
```

Oraz walutę:

```
<span class="pln">zł</span>
```

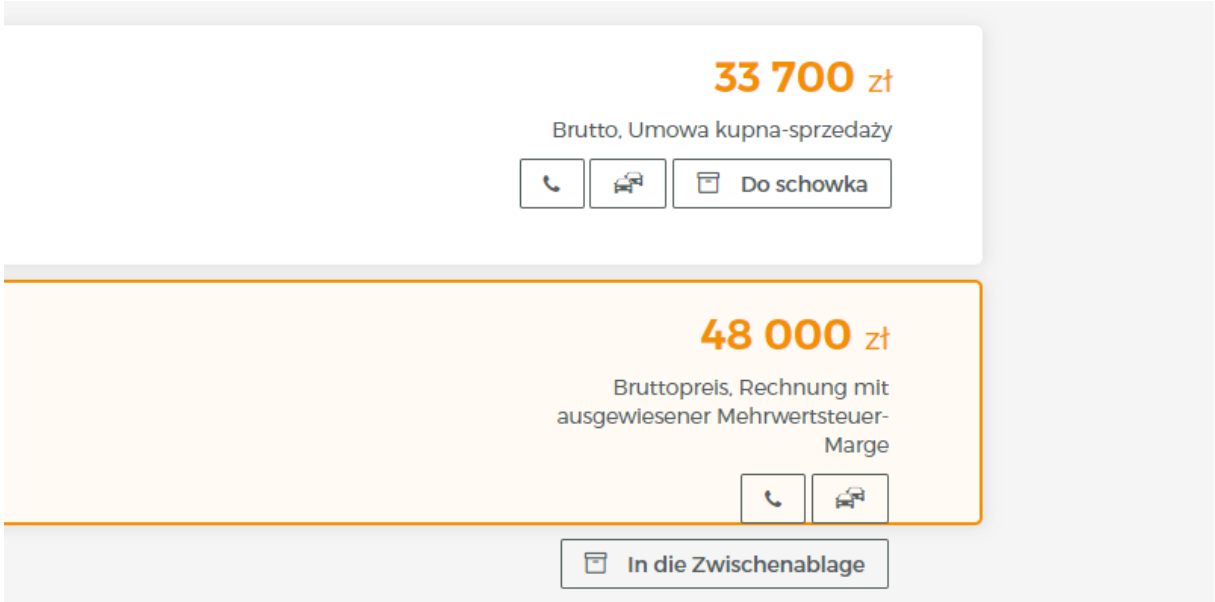
Stosowany w systemie font *Montserrat* obsługiwał wyłącznie pisma wywodzące się z alfabetu łacińskiego lub cyrylicy[8], a kodowanie znaków ISO 8859-2 (patrz punkt 2.4.3) przeznaczone było dla Europy Środkowo-Wschodniej (żaden konkretny język nie był określony wprost):

```
<meta http-equiv="content-type" content="text/html; charset=iso-8859-2">
```

Na szczęście baza danych od początku zapisywała tekst zgodnie z kodowaniem UTF-8 i zestawem znaków Unicode, nie była zatem konieczna migracja danych, a jedynie zmiany dotyczące sposobu wyświetlania w warstwie prezentacji:

```
mysql://localhost:3306/motomi?useUnicode=true&characterEncoding=UTF-8
```

Potencjalnie kłopotliwym elementem aplikacji były opisy ogłoszeń zamieszczanych przez użytkowników, od których trudno wymagać wprowadzenia tłumaczeń w kilku językach. Pomocne w tej sytuacji okazało się wykorzystanie technik tłumaczenia maszynowego.



Rys. 5.1. Motomi: tekst i przyciski wychodzą poza kontener ogłoszenia.

8 Zob. https://www.fontsquirrel.com/fonts/montserrat.

Widoki zaprojektowano zgodnie z regułami RWD, co zapewniało prawidłowe skalowanie do różnej wielkości urządzeń, także mobilnych, jednak pojedyncze elementy strony nie były przystosowane do dłuższych tekstów i w pewnych sytuacjach powodowały rozpychanie układu (patrz Rys. 5.1). Jedynymi zauważonymi materiałami multimedialnymi posiadającymi odniesienia kulturowe były reklamy – w większości w formie grafik, ale pojawiały się także filmy. Zawierały one polskie napisy, a ich humorystyczna treść została skomponowana wokół charakterystycznych dla Polski stereotypów na temat handlu samochodami. Zidentyfikowano również powiązany system *Magazyn Motomi*[9], zbudowany na platformie *WordPress*[10], który dla zachowania spójności powinien zostać poddany wdrożeniu analogicznych mechanizmów wielojęzyczności.

5.2. Internacjonalizacja systemu Motomi

W odpowiedzi na zidentyfikowane w poprzednim podrozdziale problemy, wdrożono mechanizmy zmieniające badany system na wielojęzyczny. Jako strategię internacjonalizacji przyjęto dynamiczną podmianę języka w ramach jednej aplikacji. Głównymi czynnikami, które wpłynęły na taką decyzją były:

1. dostępność wyłącznie jednej domeny,
2. działająca platforma *AngularJS*, która dzięki mechanizmom SPA pozwalała na dynamiczną zmianę zawartości strony widocznej w przeglądarce,
3. infrastruktura sieciowa dostarczana przez *Amazon Web Services* (skr. AWS), która przez automatyczne skalowanie maszyn w chmurze rozwiązywała problem ogólnoświatowego ruchu kierowanego do jednego węzła.

Najważniejsze kroki ku internacjonalizacji zostały opisane w kolejnych punktach.

5.2.1. Konfiguracja regionów

Jako reprezentatywne dla badań wybrano cztery języki: polski, angielski, grecki i arabski. Konfigurację poszczególnych regionów opisano strukturą *LOCALES*:

```
export const LOCALES = {
  'pl_PL': {
    languageName: 'polski',
    languageDirection: 'ltr',
    languageCode: 'pl',
    countryCode: 'PL'
  },
  'el_GR': {
    languageName: 'ελληνικά',
    languageDirection: 'ltr',
    languageCode: 'el',
    countryCode: 'GR'
  },
  ...
};
```

Kluczami kolejnych obiektów są kody ustawień regionalnych (patrz punkt 3.1.1), a ich wartościami: nazwa języka prezentowana użytkownikowi w interfejsie (*languageName*), kierunek pisma (*languageDirection*, patrz punkt 3.2.4), kod języka (*languageCode*, patrz

9 Zob. https://magazynmotomi.pl/.
10 Zob. https://wordpress.org/about/.

punkt 2.3.1) i kod kraju (*countryCode*, patrz punkt 2.3.2). Można zauważyć, że przyjęta postać obiektu konfiguracyjnego pozwala na rozróżnienie rynków i regionów posługujących się tym samym językiem (np. Stany Zjednoczone to klucz *en_US*, natomiast Wielka Brytania to *en_GB*), co jest konieczne między innymi dla prawidłowego kierowania wyświetlanych materiałów reklamowych.

Bieżące ustawienie regionu jest wykorzystywane przez aplikację w wielu widokach, kontrolerach, filtrach i serwisach do automatycznego generowania treści. Aby uniknąć konieczności każdorazowego wstrzykiwania odpowiednich zależności i dla wygody programisty, aktywną konfigurację umieszczono w globalnej przestrzeni nazw *$rootScope.locale* (a w przypadku widoków: *{{ $root.locale }}*).

5.2.2. Biblioteki do internacjonalizacji

Platforma *AngularJS* nie posiada wbudowanego rozwiązania pozwalającego na kompletną implementację mechanizmów internacjonalizacji. Udostępnia natomiast gotowe zestawy reguł dotyczących prawidłowego formatowania liczb, daty, czasu i walut dla różnych krajów (patrz podrozdział 3.4) w postaci modułu *angular-locale*[11]. Wybraną konfigurację wystarczy zaimportować podczas ładowania aplikacji:

```
import 'angular/bower-angular-i18n/angular-locale_el-gr';
```

Budowa modułu zakłada użycie jednego języka przez cały okres działania systemu, co w przypadku przyjętej strategii internacjonalizacji stanowiło nieakceptowalne ograniczenie. Do jego obejścia wykorzystano zewnętrzną bibliotekę *lgalfaso/angular-dynamic-locale*[12], która pozwala na aktywne przełączanie między importowanymi konfiguracjami. Podczas inicjalizacji serwisu należało jedynie podać ścieżkę do katalogu z plikami reguł:

```
tmhDynamicLocaleProvider
  .localeLocationPattern('i18n/angular-locale-{{locale}}.js');
```

Wczytanie nowego zestawu reguł następuje w odpowiedzi na zdarzenie *$translateChangeSuccess* emitowane podczas każdej zmiany języka:

```
$rootScope.$on('$translateChangeSuccess', (event, data) =>
  tmhDynamicLocale.set(data.locale));
```

Kolejną istotną biblioteką zewnętrzną, której użyto jest *angular-translate*[13]. Zintegrowano w niej szereg komponentów, które służą do podstawowego zarządzania tłumaczeniami: wczytywania napisów z pliku, dynamicznej podmiany tekstu czy przechowywania ustawień regionalnych między kolejnymi wizytami. Moduł skonfigurowano za pomocą następujących ustawień:

```
const availableLocales = Object.keys(LOCALES);
const fallbackLocale = 'en_US';

$translateProvider
  .useStaticFilesLoader({prefix: 'i18n/', suffix: '.json'})
  .registerAvailableLanguageKeys(availableLocales)
  .fallbackLanguage(fallbackLocale)
  .determinePreferredLanguage()
  .useLocalStorage();
```

11 Zob. https://docs.angularjs.org/guide/i18n.
12 Zob. https://lgalfaso.github.io/angular-dynamic-locale/.
13 Zob. https://angular-translate.github.io/docs/#/guide.

Gdzie:

1. *useStaticFilesLoader(...)* jako źródło tłumaczeń określa pliki JSON z katalogu *i18n/*,
2. *registerAvailableLanguageKeys(...)* definiuje zakres dostępnych ustawień regionalnych (w postaci tablicy kluczy wyodrębnionych z opisywanej w poprzednim punkcie struktury *LOCALES*),
3. *fallbackLanguage(...)* oznacza awaryjne źródło napisów, gdy dla wybranego języka nie będzie dostępny komplet tłumaczeń (przyjęto angielski jako uniwersalny dla europejskiej części świata, w której działa system),
4. *determinePreferredLanguage()* powoduje, że podczas pierwszej wizyty na stronie, spośród listy podanych wcześniej ustawień regionalnych, wybrany zostanie wariant najbliższy preferencjom zdefiniowanym w przeglądarce użytkownika (za pomocą API HTML5 *NavigatorLanguage.language*[14]),
5. *useLocalStorage()* zapisuje wybrane ustawienie w magazynie lokalnym przeglądarki, aby przy kolejnej wizycie załadować system w tym samym języku.

Sterowanie tłumaczeniami w tak skonfigurowanym systemie jest możliwe za pomocą udostępnianego przez bibliotekę serwisu *$translate* i funkcji *use(...)*, która pełni jednocześnie rolę pobierającą i ustawiającą – wywołana bez parametru zwraca bieżące ustawienie:

```
$translate.use('el_GR');
$translate.use(); // el_GR
```

Mechanizm przełączania języka zostanie szerzej omówiony w punkcie 5.2.5.

5.2.3. Dane liczbowe

Ogłoszenia motoryzacyjne zamieszczane w systemie *Motomi* cechują się licznymi parametrami numerycznymi: ceną w dwóch walutach (złoty i euro), przebiegiem, pojemnością i mocą silnika, rokiem produkcji oraz datą publikacji. Dysponując skonfigurowanymi regułami zapisu danych liczbowych w różnych językach, należało wskazać miejsca, w których ma zostać zaaplikowane stosowne formatowanie. Posłużono się w tym celu filtrami wbudowanymi w platformę *AngularJS*, bezpośrednio w widokach:

```
<span>{{ offer.mileageValue | number }} km</span>
<span>{{ offer.prices.EUR | currency:'€' }}</span>
<span>{{ offer.creationDate | date:'longDate' }}</span>
```

Number[15] to filtr ogólnego przeznaczenia, który zapewnia prawidłowe grupowanie cyfr i użycie odpowiedniego separatora dziesiętnego. *Currency*[16] jest filtrem walutowym, rozbudowanym o możliwość określenia symbolu waluty (jego pozycja po prawej lub po lewej stronie kwoty jest ustalana automatycznie). *Date*[17] odpowiada natomiast za formatowanie czasu. Filtr wymaga podania sposobu prezentacji danych w standardzie ISO 8601 (patrz podrozdział 3.4), jednak zamiast tego posłużono się aliasami *shortDate*, *mediumDate* i *longDate*, które powodują automatyczne wstrzykiwanie formatu odpowiedniego dla wybranego języka.

14 Zob. https://developer.mozilla.org/en-US/docs/Web/API/NavigatorLanguage/language.
15 Zob. https://docs.angularjs.org/api/ng/filter/number.
16 Zob. https://docs.angularjs.org/api/ng/filter/currency.
17 Zob. https://docs.angularjs.org/api/ng/filter/date.

Dodatkowo cena widoczna przy ogłoszeniu różni się w zależności od regionu użytkownika. Dla odbiorców z Polski eksponowana jest kwota w złotówkach, a w pozostałych przypadkach wyróżnione jest euro. Efekt uzyskano nadając odpowiednią klasę CSS na podstawie porównania bieżących ustawień regionalnych i kodu kraju:

```
<div ng-class="{highlight: $root.locale.countryCode === 'PL'}">
  {{ vm.offer.prices.PLN | currency:'zł':0 }}
</div>

<div ng-class="{highlight: $root.locale.countryCode !== 'PL'}">
  {{ offer.prices.EUR | currency:'€':0 }}
</div>
```

Dyrektywa *ngClass*[18] przydzieli elementowi powodującą wyróżnienie klasę *highlight*, tylko jeśli spełniony zostanie warunek podany po dwukropku.

5.2.4. Tekst i metadane

Podstawowe metadane odpowiadające za prawidłową interpretację tekstu przez przeglądarkę, wyszukiwarki internetowe i programy do syntezy mowy (tj. język dokumentu, kierunek tekstu oraz kodowanie znaków), zostały podane bezpośrednio w głównym szablonie *index.html*, zgodnie z zaleceniami opisanymi w podrozdziale 3.2:

```
<html lang="{{ $root.locale.languageCode }}"
      dir="{{ $root.locale.languageDirection }}">
  <head>
    <meta charset="utf-8">
    ...
  <head>
  ...
</html>
```

Pozostałe metadane generowane są osobno dla każdego widoku i wstrzykiwane do sekcji *head* strony za pomocą modułu *vinaygopinath/ngMeta*[19]. Do wykrycia zmiany języka ponownie wykorzystano nasłuch na zdarzenie *$translateChangeSuccess*:

```
$rootScope.$on('$translateChangeSuccess', () => {
  const keys = ['homeView.meta.title', 'homeView.meta.description'];
  $translate(keys).then(translations => {
    ngMeta.setTitle(translations[keys[0]]);
    ngMeta.setTag('description', translations[keys[1]]);
    ...
  });
  ...
})
```

Serwis *$translate* pozwala pobrać tłumaczenia metadanych według kluczy (*keys*) odpowiadających strukturą widokowi, w którym znajduje się użytkownik (tutaj: widok główny *homeView*), a serwis *ngMeta* ustawić ich wartości w odpowiednich tagach za pomocą funkcji *setTitle* (tytuł strony) i *setTag* (wszystkie inne metadane).

Aby w pełni pokryć zestaw wymaganych znaków, konieczna była zmiana używanego fontu. Wybrano uniwersalną rodzinę *Noto Sans Display*[20], która zawiera glify dla 582

18 Zob. https://docs.angularjs.org/api/ng/directive/ngClass.
19 Zob. https://vinaygopinath.github.io/ngMeta/#/.
20 Zob. https://www.google.com/get/noto/#sans-lgc-display.

najpopularniejszych języków świata. W ten sposób zapobieżono również przyszłym problemom mogącym powstać podczas rozbudowy systemu o obsługę kolejnych regionów. Font zaimportowano zgodnie z techniką przedstawioną w punkcie 3.2.2, wykorzystując regułę CSS *@font-face* i optymalizację *unicode-range* dla zakresu znaków *U+0000-017F, U+0370-03FF, U+0600-06FF* (pierwsza grupa to symbole uniwersalne i alfabety oparte o łaciński, druga to alfabet grecki, a trzecia arabski).

5.2.5. Przełączanie języka systemu

Domyślnym mechanizmem odpowiadającym za nawigację w systemach zbudowanych na bazie platformy *AngularJS* jest *ngRoute*[21]. Sprawdza się on w przypadku prostych aplikacji, jednak w rozbudowanym, internacjonalizowanym systemie z dynamiczną zmianą języka brakuje mu wygodnych sposobów rozszerzania funkcjonalności, przekazywania parametrów i powiadamiania innych komponentów o zmianie stanu. Ponadto nie posiada żadnej warstwy abstrakcji dla odnośników i opiera się na ich bezpośredniej postaci, a tym samym uniemożliwia lokalizację adresu strony. Spowodowało to konieczność wymiany routera na niestandardowy *angular-ui/ui-router*[22], który nie posiada wspomnianych wad.

Główną zmianą było przejście na wewnętrzną nawigację za pomocą abstrakcyjnych stanów, na podstawie których generowane są widoczne dla użytkownika linki. Pozwoliło to na zdefiniowanie poprzedzającego każdy adres parametru, który określa aktywny język systemu:

```
$stateProvider.state('motomi', {
  abstract: true,
  url: '/{languageCode:(?:pl|en|el|ar)}',
  ...
});
```

Konfiguracja stanów odbywa się za pomocą serwisu *$stateProvider* i funkcji *state(...)*, która przyjmuje dwa argumenty: wewnętrzną nazwę danego widoku oraz obiekt z ustawieniami. W powyższym fragmencie kodu zdefiniowano główny stan aplikacji o nazwie *motomi*, względem którego zbudowano hierarchię pozostałych widoków. Właściwość *abstract: true* oznacza, że choć sam z siebie nie stanowi miejsca w które może przejść użytkownik, to fragment podany w atrybucie *url* będzie doklejany do adresu wszystkich jego pochodnych. W tym przypadku jest to wspomniany parametr określający język systemu (zawężony do zbioru: *pl*, *en*, *el*, *er*). Dostęp do jego wartości jest możliwy bezpośrednio przez zmienną *languageCode* udostępnianą przez serwis *$stateParams* lub przez obiekty *fromParams* i *toParams* przekazywane wraz ze zdarzeniem *$stateChangeStart*, emitowanym podczas przejścia do innego widoku.

Doprowadziwszy do sytuacji, w której adres URL zawiera kod języka w postaci opisanej w punkcie 4.1.3, należało zadbać o odpowiednią reakcję systemu na jego zmiany. Po pierwsze, podczas wczytywania strony sprawdzane jest czy podczas poprzedniej wizyty, w magazynie lokalnym dostępnym przez funkcję *$translate.storage()*, został zapisany kod preferowanych ustawień regionalnych (*storedLocale*). Jeżeli tak się nie stało, bo są to na przykład pierwsze odwiedziny systemu na danym komputerze lub użytkownik wyłączył możliwość przechowywania ustawień stron webowych na swoim urządzeniu, to zostanie on dobrany w oparciu o konfigurację przeglądarki internetowej (służy do tego funkcja *$translate.preferredLanguage()*). Na podstawie wyznaczonego w ten sposób kodu ustawień

21 Zob. https://scotch.io/tutorials/single-page-apps-with-angularjs-routing-and-templating.
22 Zob. https://ui-router.github.io/ng1/.

regionalnych pobierany jest kod języka *languageCode* znajdujący się w strukturze *LOCALES*:

```
const determinedLanguageCode = () => {
  const storedLocale = $translate
    .storage()
    .get($translate.storageKey());

  return storedLocale ?
    LOCALES[storedLocale].languageCode :
    LOCALES[$translate.preferredLanguage()].languageCode;
};
```

Tak określony język jest ustawiany w adresie za pomocą funkcji mapującej *$urlRouterProvider.when(...)*, której pierwszym argumentem jest bieżący adres, a drugim ten, na który następuje przekierowanie:

```
$urlRouterProvider.when('/', '/' + determinedLanguageCode());
```

W efekcie, po wpisaniu w przeglądarce adresu *https://motomi.pl/* automatycznie dodany zostanie do niego człon określający język, na przykład Grek zobaczy *https://motomi.pl/el/*. Dostępny w adresie parametr pozwala na bieżącą aktualizację tłumaczeń podczas każdej zmiany stanu (zdarzenie *$stateChangeStart*):

```
$rootScope.$on('$stateChangeStart', (event, toState, toParams) => {
  $translate.use(toParams.languageCode);
});
```

Podsumowując, dzięki kombinacji technik udostępnianych przez moduły *angular-translate* i *ui-router* uzyskano rozwiązanie, którego algorytm wyboru języka działa z następującymi priorytetami: najpierw parametr w adresie, następnie ustawienie zapisane w magazynie lokalnym podczas poprzedniej wizyty, a na końcu preferencje przeglądarki internetowej zainstalowanej na komputerze użytkownika.

Dodatkowo przygotowano selektor języka dostępny jako element graficznego interfejsu aplikacji (patrz Rys. 5.2). Komponent wykonano zgodnie z dobrymi praktykami podanymi w podrozdziale 4.3 - jest on umieszczony w widocznym miejscu ekranu (górna część strony, nad logiem) i oznaczony międzynarodowym symbolem kuli ziemskiej, a poszczególne opcje prezentowane są w postaci tekstowej, w językach narodowych, zamiast przy użyciu flag.

Rys. 5.2. Motomi: selektor języka.

Od strony technicznej jest to prosty szablon, który na podstawie struktury konfiguracyjnej *LOCALES* automatycznie generuje przyciski w pętli *ngRepeat*[23]. Zmiana języka po kliknięciu jednego z nich odbywa się za pomocą opisywanego już wcześniej serwisu *$translate.use(...)* wstrzykiwanego jako zależność kontrolera selektora. Wykonanie tej akcji wyzwala emisję zdarzenia *$translateChangeSuccess*, które przekazuje nowe ustawienie w obiekcie *data.language*. Dzięki temu możliwe jest odświeżenie parametru w adresie, tak aby odpowiadał on nowo wybranemu językowi:

23 Zob. https://docs.angularjs.org/api/ng/directive/ngRepeat.

```
$rootScope.$on('$translateChangeSuccess', (event, data) => {
  $stateParams.languageCode = data.language;
  $state.go($state.current, $stateParams, { reload: false });
});
```

W powyższym kodzie *$state.current* zawiera wewnętrzną nazwę bieżącego widoku, a *$stateParams* wszystkie parametry jakie były dotąd przekazywane w adresie URL. Funkcja *$state.go(...)* pozwala przejść do podanego stanu – w tym przypadku chodzi o ten sam, w którym znajduje się już odwiedzający, jednak z innymi parametrami (nowym językiem). Dzięki opcji *reload: false* nie nastąpi przy tym przeładowanie strony, a jedynie aktualizacja adresu, zatem podczas przełączania języka nie zostaną utracone żadne dane wprowadzone dotychczas przez użytkownika do formularzy, itp.

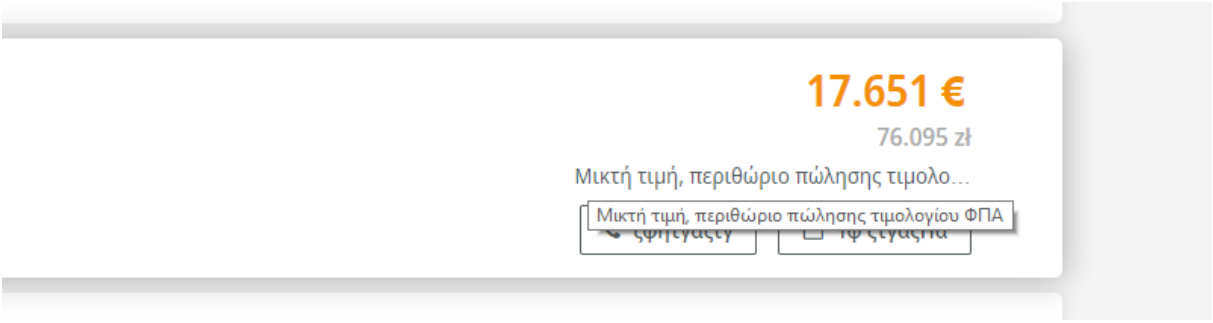
5.2.6. Tłumaczenie treści

Do przechowywania tłumaczeń wybrano naturalny dla języka JavaScript format JSON (patrz punkt 2.5.2). Każdy region posiada osobny plik nazwany zgodnie z kodem ustawień regionalnych (np. *ar_QA.json*). Wewnętrzna struktura jest we wszystkich przypadkach taka sama – przyjęto grupowanie kluczy ze względu na elementy wspólne (nagłówek, stopka, itp.) i widoki (strona główna, wyniki wyszukiwania, detal ogłoszenia, itd.). Powoduje to powtarzanie niektórych tłumaczeń, jednak podczas przeprowadzonych badań okazało się to właściwym podejściem, ponieważ w niektórych przypadkach ten sam tekst jest różnie formułowany ze względu na budowę widoku (np. *wszystkie samochody* i *wszystkie pojazdy*).

Problem zbyt długiego tekstu w innych językach został rozwiązany za pomocą wstawki *text-overflow* dostępnej w wykorzystywanej przez system bibliotece *Bootstrap*:

```
.tax-field {
  @include text-overflow;
  ...
}
```

Dołączenie jej do problematycznej klasy CSS za pomocą dyrektywy *@include* skutkuje nadaniem atrybutów, które powodują ucinanie zbyt długiego tekstu za pomocą wielokropka. Aby nie ograniczać w ten sposób kompletność informacji, użyto atrybutu HTML *title*, dzięki któremu, po najechaniu kursorem na skrócony fragment tekstu, wyświetlany jest dymek z pełną treścią (patrz Rys. 5.3). W niektórych przypadkach zmieniono również sposób wyświetlania elementów z domyślnego blokowego na *Flexbox Layout* (zgodnie z opisem z punktu 3.2.3).



Rys. 5.3. Motomi: przycinanie tekstu i dymek z pełną informacją.

Mając na uwadze różnice w składni zdań występujące między poszczególnymi językami, umożliwiono zmianę pozycji zmiennych pojawiających się w tłumaczonych

frazach. Uprościło to lokalizację i podniosło jej jakość, ponieważ dzięki parametryzacji tłumaczeń nie trzeba było już rozbijać tekstu na krótkie, pozbawione kontekstu fragmenty ręcznie sklejane ze zmiennymi. Miejsce wstrzyknięcia parametru określa odpowiednio nazwany symbol zastępczy w podwójnych nawisach klamrowych:

```
{ "allCars": "Wszystkie pojazdy: {{ carsCount }}"; }
```

Parametry przekazywane są jako argumenty filtra *translate*[24], bezpośrednio w widoku:

```
<span>
  {{ 'allCars' | translate:{ carsCount: numberOfAllCars } }}
</span>
```

Dodatkowo posłużono się dyrektywą *ngBindHtml*[25], która pozwala tłumaczowi wyróżnić wybrane fragmenty tekstu za pomocą prostych tagów HTML:

```
{ "saleCars": "Wyprzedaż <strong>2018</strong>"; }

<span ng-bind-html="'saleCars' | translate"></span>
```

O ile elementy interfejsu i dane słownikowe wybierane ze zdefiniowanych wcześniej list mogły zostać przetłumaczone z góry przez człowieka, o tyle właściwy opis pojazdu, wprowadzany przez użytkownika ręcznie, spowodował konieczność wprowadzenia mechanizmu tłumaczenia maszynowego. Stosując techniki inżynierii wstecznej zbadano interfejs z jakim komunikuje się strona *Google Translate*[26] i w oparciu o jej API przygotowano serwis umożliwiający pobieranie translacji automatycznych. Ograniczeniem tej metody jest potencjalny limit zapytań, który po przekroczeniu pewnej częstotliwości odpytywania może spowodować zablokowanie dostępu do usługi, w takiej formie jest to jednak rozwiązanie darmowe:

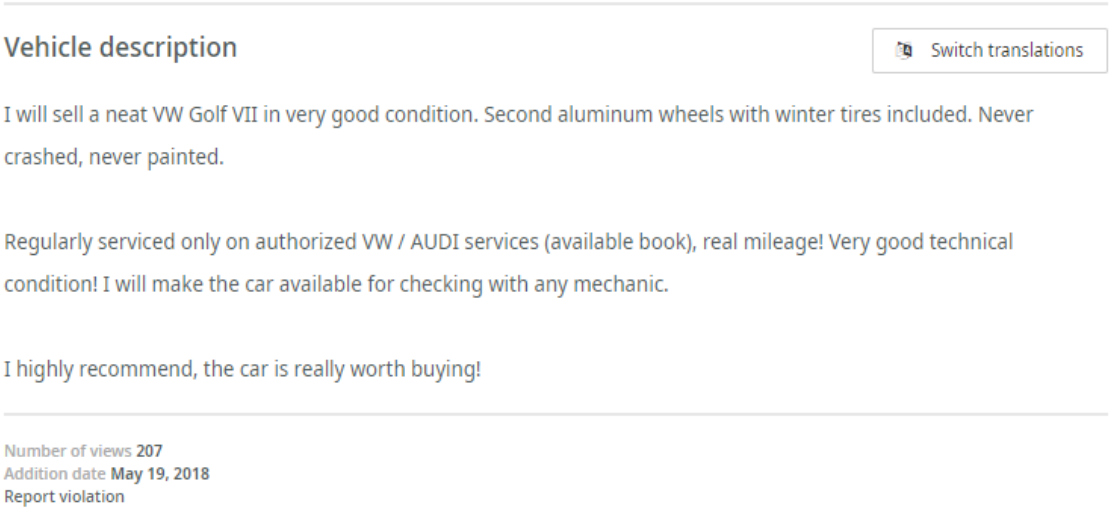
```
const apiUrl = 'https://translate.googleapis.com/translate_a/single';

getTranslation(text, fromLang, toLang) {
  let config = {
    cache: true,
    params: {
      client: 'gtx',
      sl: fromLang,
      tl: toLang,
      dt: 't',
      q: text
    }
  };

  return $http.get(apiUrl, config).then(response =>
    response.data[0].reduce((acc, sentence) => acc + sentence[0], '')
  );
}
```

Serwis *$http*[27] jest domyślnym dla platformy *AngularJS* sposobem wykonywania zapytań asynchronicznych. Przyjmuje on dwa argumenty: adres oraz obiekt konfiguracyjny. Istotne jest tutaj użycie atrybutu *cache: true*, dzięki któremu rezultat raz wykonanego zapytania

24 Zob. https://angular-translate.github.io/docs/#/api/pascalprecht.translate.filter:translate.
25 Zob. https://docs.angularjs.org/api/ng/directive/ngBindHtml.
26 Zob. https://translate.google.com/.
27 Zob. https://docs.angularjs.org/api/ng/service/$http.

zostanie zapisany w pamięci podręcznej i nie będzie ponowiony do kolejnego uruchomienia aplikacji. W ten sposób zmniejszono ryzyko przekroczenia limitów ustalonych przez Google. Parametry *sl* i *tl* oznaczają odpowiednio kod języka źródłowego i docelowego (zgodny ze standardem ISO 639), *q* to właściwy tekst do przetłumaczenia, a *client* i *dt* przyjmują stałe wartości specyficzne dla tego API. Zwracana odpowiedź zawiera liczne metadane Google i jest rozbita na zdania umieszczone w osobnych, zagnieżdżonych tablicach. Wymagało to spłaszczenia struktury za pomocą funkcji *reduce(...)*.

Biorąc pod uwagę niedoskonałą wciąż jakość tłumaczenia maszynowego, umożliwiono użytkownikowi powrót do wersji oryginalnej za pomocą przycisku obok tekstu. Jest to dobra praktyka, która ułatwia zrozumienie tekstu, jeżeli w przetłumaczonej postaci została utracona struktura opisu lub wymieszane zostały dane liczbowe. Efekt działania opisanego mechanizmu jest widoczny poniżej na Rys. 5.4.



Rys. 5.4. Motomi: tłumaczenie maszynowe opisu samochodu.

5.2.7. Multimedia i reklamy

Jedynymi materiałami multimedialnymi występującymi w systemie, które nie są przesyłane przez użytkowników, a zatem można poddać je lokalizacji, są reklamy. Istniejący algorytm losuje adresy zdjęć z tablicy *BANNERS*:

```
const BANNERS = [{imageUrl: '...', link: '...'}, ...];
```

Wystarczyło zatem zmodyfikować strukturę powyższego obiektu o kody krajów i jako parametr algorytmu podstawiać tablicę adekwatną do bieżących ustawień regionalnych:

```
const BANNERS = {
  'PL': [{imageUrl: '...', link: '...'}, ...],
  'US': [{imageUrl: '...', link: '...'}, ...],
  ...
};
```

Pozostałe reklamy występowały najczęściej w formie zintegrowanych z interfejsem odnośników do zewnętrznych systemów finansowania zakupów. Ich lokalizacja jest zatem zależna wyłącznie od decyzji biznesowej podmiotów zewnętrznych. Można również przyjąć, że standardowa oferta ratalna nie byłaby możliwa dla obcokrajowców ze względów prawnych, dlatego elementy tego typu zdecydowano się wyświetlać tylko w wersji polskiej.

Wykorzystano do tego dyrektywę *ngShow*[28], która sprawdza kod kraju dla bieżących ustawień regionalnych i wyświetla element, tylko jeśli jest to kod Polski:

```
<div class="ad" ng-show="$root.locale.countryCode === 'PL'">...</div>
```

Rozwiązanie to można traktować jako uniwersalne, przydatne we wszystkich sytuacjach wymagających doraźnej modyfikacji zawartości interfejsu w zależności od kraju lub języka. Warto również zwrócić uwagę na bliźniacze dyrektywy *ngHide*[29] i *ngIf*[30].

5.2.8. Tekst dwukierunkowy

Jednym z ciekawszych wyzwań internacjonalizacji okazało się dostosowanie interfejsu aplikacji do potrzeb języka pisanego od prawej do lewej, w tym przypadku arabskiego. Jak pokazano wcześniej na Rys. 3.3 sama zmiana kierunku tekstu w dokumencie jest niewystarczająca, a wręcz powoduje pewne problemy wizualne. Oprócz niej konieczne było również odbicie układu graficznego elementów. Strategie rozwiązania tego problemu wymienione w punkcie 3.2.4 zakładały użycie różnych technik nadpisywania kluczowych atrybutów CSS lub utworzenie oddzielnego arkusza stylów. Ostatecznie zaimplementowano rozwiązanie, które całkowicie odciąża od myślenia na ten temat i pozwala skupić się wyłącznie na kierunku LTR, na podstawie którego automatycznie generowane jest pełne odbicie lustrzane strony dla kierunku RTL.

Do realizacji zadania posłużono się narzędziem *Gulp*[31], które pozwala automatyzować czynności deweloperskie w aplikacjach JavaScript. Opiera się ono na potokowym przetwarzaniu wskazanych plików przez szereg wtyczek, które swoje wyjście przekazują jako wejście kolejnej. Grupując kilka z nich utworzono przepływ, który zamienia odpowiednie atrybuty CSS (np. *margin-left: 10px* na *margin-right: 10px*) w źródłowym arkuszu stylów *styles/motomi-ltr.scss* i zapisuje efekt w bliźniaczym pliku *styles/motomi-rtl.css*:

```
gulp.task('styles:rtl', () =>
  gulp.src('styles/motomi-ltr.scss')
    .pipe(sassVariables({ $direction: 'rtl' }))
    .pipe(sass())
    .pipe(rtlcss())
    .pipe(rename({ basename: 'motomi-rtl' }))
    .pipe(gulp.dest('styles/')));
```

Powyższy kod wpleciony w proces budowania aplikacji pozwolił nasłuchiwać na zmiany w podstawowym arkuszu stylów LTR i automatycznie generować na jego podstawie styl RTL. Uruchomienie zadania *styles:rtl* zdefiniowanego za pomocą *gulp.task(...)* powoduje przetworzenie zawartości pliku *motomi-ltr.scss* pobranego funkcją *gulp.src(...)* przez następujące wtyczki *gulp.pipe(...)*:

1. *sassVariables(...)*, która przekazuje do arkusza parametr *$direction*, dzięki któremu styl RTL może być nie tylko odbity, ale i w dowolny sposób zmodyfikowany względem swojego pierwowzoru,
2. *sass()*, która konwertuje wygodny dla programisty format SCSS na obsługiwany przez przeglądarki CSS,

28 Zob. https://docs.angularjs.org/api/ng/directive/ngShow.
29 Zob. https://docs.angularjs.org/api/ng/directive/ngHide.
30 Zob. https://docs.angularjs.org/api/ng/directive/ngIf.
31 Zob. https://github.com/gulpjs/gulp/blob/v3.9.1/docs/getting-started.md.

3. *rtlcss()*, która jest implementacją narzędzia RTLCSS opisanego w punkcie 3.2.4 i wykonuje właściwe odbicie układu,
4. *rename(...)*, która nadaje odpowiednią nazwę plikowi wynikowemu,
5. *gulp.dest(...)*, która określa katalog docelowy dla wygenerowanego wyjścia.

Wspomniany parametr *$direction* został wykorzystany do zmiany głównego motywu kolorystycznego na złoty, który kojarzy się w krajach arabskich z bogactwem i prestiżem. Wystarczyło użyć instrukcji warunkowej *@if*, aby nadpisać odpowiednią zmienną w kodzie arkusza SCSS:

```
$main-color: orange;

@if $direction == 'rtl' {
  $main-color: gold;
}
```

Zmiana stylu z *motomi-ltr.css* na *motomi-rtl.css* odbywa się dynamicznie, na podstawie bieżących ustawień regionalnych, w pliku *index.html*:

```
<link rel="stylesheet"
      href="styles/motomi-{{ $root.locale.languageDirection }}.css">
```

Nie wszystkie elementy strony mogły zostać odwrócone za pomocą zmiany kierunku tekstu w dokumencie lub alternatywnego arkusza stylów. Kłopotliwe okazały się grafiki strzałek *„<”* oraz *„>”* wykorzystywanych podczas nawigacji w systemie. Pochodzą one z biblioteki *Font Awesome*[32] i są ustawiane za pomocą klas CSS: *fa-angle-left* dla *„<”* i *fa-angle-right* dla *„>”*:

```
<span class="fa fa-angle-right"></span>
```

Receptą na problem było przygotowanie dyrektywy *class-by-direction*, która ustawia podane w atrybutach klasy w zależności od wybranego w systemie języka (*ltr-class* dla pisma LTR i *rtl-class* dla pisma RTL):

```
<span class="fa" class-by-direction
      ltr-class="fa-angle-right" rtl-class="fa-angle-left"></span>
```

Implementacja polegała na obserwowaniu zmian ustawień regionalnych w powszechnie wykorzystywanej w poprzednich przykładach zmiennej *$rootScope.locale* i przydzielaniu klasy na podstawie zawartej w obiekcie konfiguracyjnym wartości *languageDirection*:

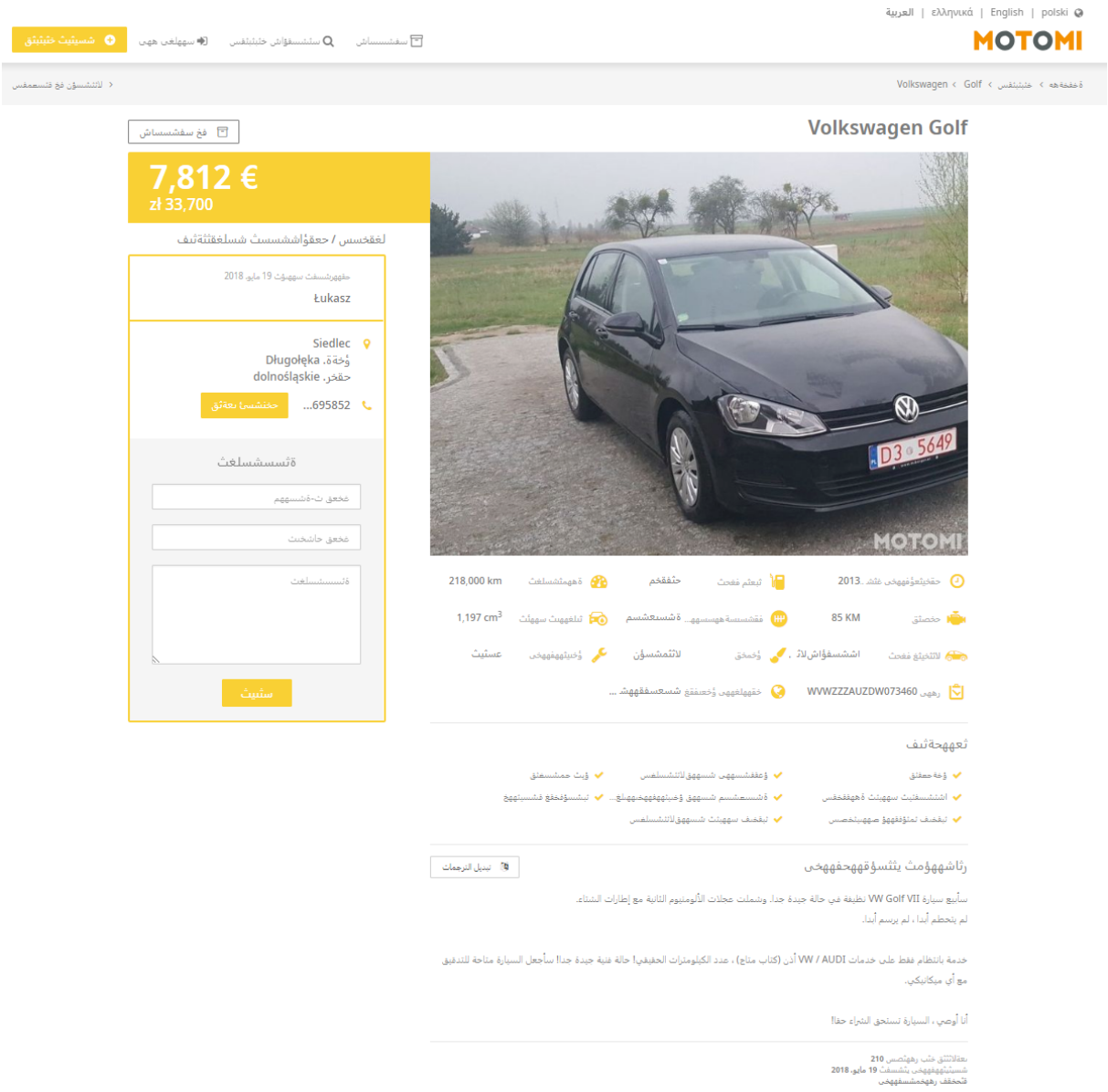
```
$rootScope.$watch('locale', newLocale => {
  if (newLocale.languageDirection === 'ltr') {
    attr.$addClass(attr.ltrClass);
    attr.$removeClass(attr.rtlClass);
  } else {
    attr.$addClass(attr.rtlClass);
    attr.$removeClass(attr.ltrClass);
  }
});
```

Dodanie lub usunięcie klasy odbywa się za pomocą funkcji *attr.$addClass(...)* oraz *attr.$removeClass(...)*, które są dostępne w każdej dyrektywie platformy *AngularJS*.

Ostatnim koniecznym do uwzględnienia zagadnieniem było liczne użycie opisywanego w punkcie 3.2.4 znacznika HTML *bdi* dla danych numerycznych

32 Zob. https://fontawesome.com/icons?d=gallery.

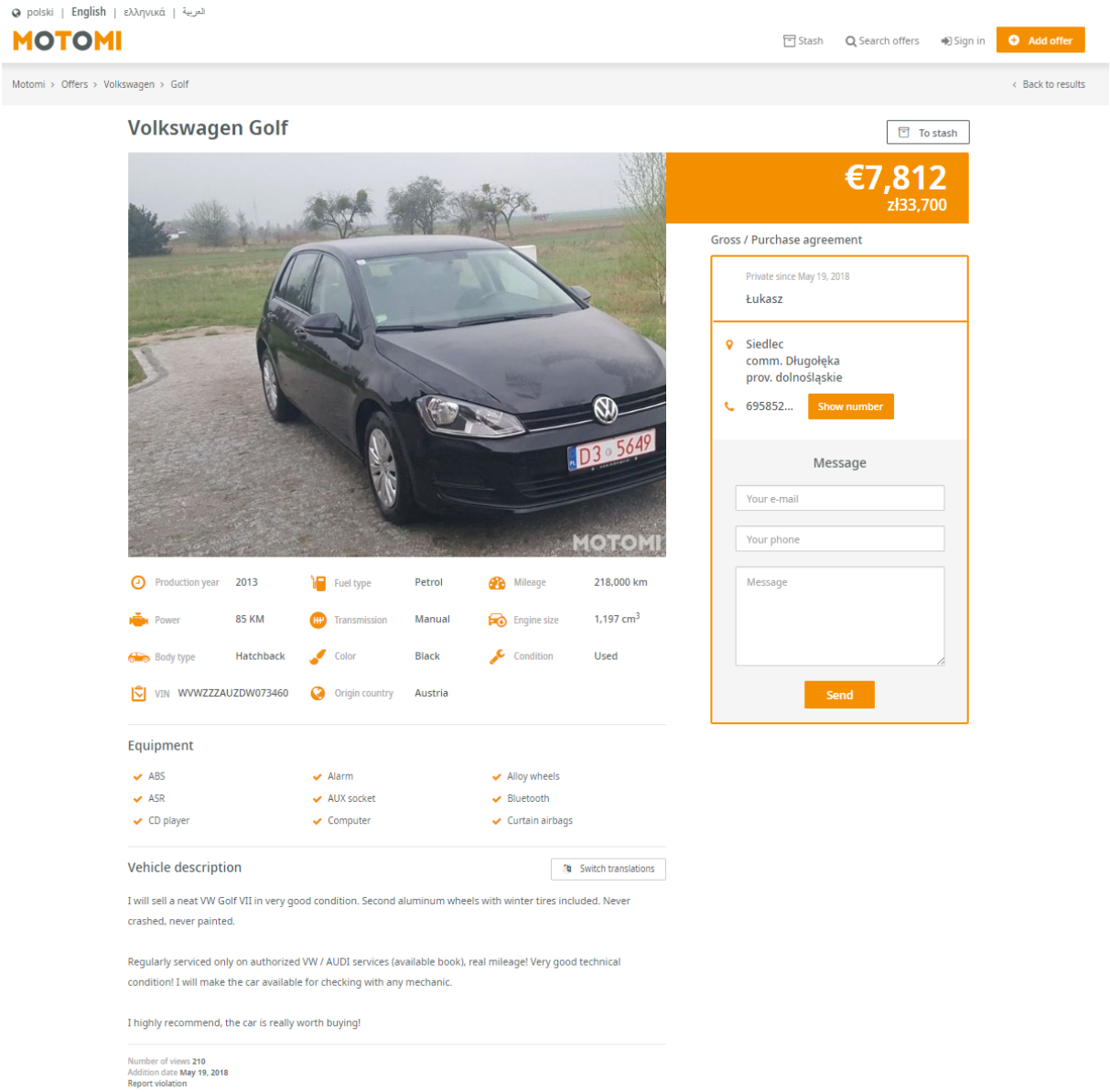
i wczytywanych z bazy danych. Dotyczyło to głównie cen i parametrów technicznych ofert, które zapisywane klasycznie w kierunku LTR musiały być izolowane od reszty tekstu, aby zachować swój sens (inaczej cena 20 300zł zmieniała się w zł300 20).



Rys. 5.5. Motomi: widok ogłoszenia w języku arabskim (pismo RTL).

Ostateczny efekt wdrożenia języka arabskiego (pismo RTL) jest widoczny powyżej na Rys. 5.5. Dla porównania widok tego samego ogłoszenia, ale w języku angielskim (pismo LTR), przedstawiono poniżej na Rys. 5.6. Obie wersje różnią się przede wszystkim kierunkiem tekstu (również strzałek na szarej belce nawigacyjnej u góry), położeniem elementów interfejsu graficznego (od najmniejszych, jak ikony obok poszczególnych parametrów pojazdu, po największe, jak formularz kontaktowy z ceną) oraz kolorystyką (złotą dla języka arabskiego i pomarańczową dla języka angielskiego). Pełne wdrożenie mechanizmów RTL nie powiodło się z powodu galerii zdjęć. Jej twórca błędnie przyjął pewne założenia na temat kierunkowości tekstu bezpośrednio w kodzie JavaScript swojego komponentu, zamiast ograniczyć się w nim do zaprogramowania logiki, a kwestie

strukturalne i wizualne rozwiązać za pomocą kodu HTML i CSS. Taka implementacja jest bardzo specyficzna i pozostała poza zasięgiem zmian wprowadzanych przez opisywane, uniwersalne techniki.



Rys. 5.6. Motomi: widok ogłoszenia w języku angielskim (pismo LTR).

5.3. Ocena rozwiązania i wnioski

Zaprojektowane i wykonane rozwiązanie jest kompromisem pomiędzy dostosowaniem do najważniejszych wymagań niepolskojęzycznych odbiorców, a nakładem pracy i kosztami jakie trzeba było ponieść, aby je wdrożyć. Przedstawiona implementacja spełnia najważniejsze zalecenia związane z prawidłową lokalizacją systemów webowych, które opisano na początku pracy w punkcie 2.1.1:

1. prawidłowe wyświetlanie zawartości, uwzględniające odpowiednie kodowanie znaków, dobór fontów i rozmiar kontenerów tekstowych,
2. przetłumaczony interfejs oraz treści publikowane przez użytkowników,

3. formatowanie danych liczbowych, dat, czasu i walut w sposób właściwy dla konkretnego języka,
4. możliwość sterowania kolorystyką i – do pewnego stopnia – układem kluczowych elementów, takich jak cena,
5. grafiki i reklamy zróżnicowane w zależności nie tylko od języka, ale i kraju odbiorcy,
6. uwzględnienie specyfiki pism RTL i potraktowanie ich na równi z pismami LTR.

Zrealizowane zostały także główne założenia internacjonalizacji, pomimo że powinno się ją przeprowadzać w początkowej fazie projektu, a nie w funkcjonującym parę lat systemie. Osiągnięto to przez staranny dobór najbardziej uniwersalnych technik, które w sposób kompleksowy uodporniły badaną aplikację na problemy jakie potencjalnie mogłyby się pojawić podczas dodawania kolejnych języków. Użyte kodowanie znaków UTF-8, font *Noto Sans*, kontenery *Flexbox*, tłumaczenie maszynowe czy stosowanie się do międzynarodowych standardów ISO zapewnią prawidłową współpracę z większością języków, bez konieczności podejmowania w przyszłości poważniejszych działań wymagających ręcznego dostosowywania. Również od strony typowo technicznej spełnione zostały najlepsze praktyki programistyczne – wprowadzone mechanizmy potrafią w pełni automatycznie reagować na nowe ustawienia regionalne, a do rozszerzenia działającej konfiguracji wystarczy prosta modyfikacja jednego pliku.

Warto także odnotować, że powstałe rozwiązanie jest na tyle niezależne, że nie wymagało jakiejkolwiek ingerencji w infrastrukturę sieciową, aplikację serwerową czy schemat bazy danych. Nie wpłynęło zatem na zacieśnienie niepożądanego związania komponentów systemu webowego ani nie obciążyło budżetu dodatkowymi kosztami utrzymania. Jest to nowoczesne i eleganckie podejście, którego cechy są istotne nie tylko z punktu widzenia inwestora, ale i zespołu, który nie musi być w pełni zaangażowany w wytwarzanie mechanizmów wielojęzyczności i może skupić się na innych zadaniach.

Z perspektywy odwiedzającego stronę internauty największym minusem opisywanej metody jest niska wrażliwość na aspekty kulturowe. Wynika to po części ze specyfiki samego systemu, który zakłada sprzedaż samochodów na rynku polskim, wyłącznie przez polskich przedsiębiorców, ale także z głównego założenia wybranej strategii internacjonalizacji, które polega na przygotowaniu możliwie uniwersalnych i neutralnych kulturowo treści. Zachowano pod tym względem pewną elastyczność, która pozwoliła podmieniać lub ukrywać niektóre elementy interfejsu, jednak techniki te nie umożliwiają całkowitego przeorganizowania układu graficznego dla egzotycznych rynków takich jak Japonia. Próba wykorzystania ich do tego celu doprowadziłaby do powstania trudnego w zrozumieniu i utrzymaniu kodu pełnego rozgałęzień i instrukcji warunkowych. W takiej sytuacji lepiej sprawdziłyby się osobne instancje systemu dla wybranych języków (choć niesie to ze sobą inne problemy i dodatkowe koszty).

Choć zaimplementowane rozwiązanie jest w pełni działającym, możliwym do wdrożenia podejściem, to nadal istnieje pole do jego dalszego rozwoju i podnoszenia jakości. Kolejnymi krokami mogą być:

1. pełna lokalizacja linków, zgodna z opisem w podrozdziale 4.3 – usprawniłoby to pozycjonowanie w wynikach wyszukiwarek internetowych, ale także ułatwiło użytkownikom nawigację w systemie (obecnie, niezależnie od wybranych ustawień regionalnych, odnośniki pozostają w języku angielskim),

2. podjęcie współpracy z zawodowymi tłumaczami lub osobami, dla których wybrane języki są ojczystymi – umożliwiłoby to szczegółową weryfikację poprawności lingwistycznej rozwiązania, która zwłaszcza w przypadku pism niewywodzących się z alfabetu łacińskiego jest utrudniona; pozwoliłoby to również przygotować profesjonalne tłumaczenia w miejsce obecnych, które powstały jako rezultat tłumaczenia maszynowego lub tzw. *pseudolokalizacji* (więcej na jej temat można przeczytać w [Sta04]),
3. rozszerzenie opisanych metod na całą aplikację oraz systemy pochodne; na potrzeby niniejszej pracy skupiono się głównie na technicznym wymiarze działania mechanizmów wielojęzyczności, do którego analizy nie była konieczna lokalizacja wszystkich ekranów, przygotowanie kompletnych tłumaczeń czy prawdziwych wersji alternatywnych grafik reklamowych,
4. wymiana komponentu galerii zdjęć na kompatybilny z pismem RTL lub podatny na taką modyfikację bez konieczności głębokiej ingerencji w kod źródłowy samej biblioteki,
5. stopniowe wprowadzanie obsługi kolejnych języków – ciekawe pod tym względem mogą być zwłaszcza pisma ideograficzne, charakterystyczne dla kultury dalekowschodniej.

5.3.1. Porównanie z systemem Auto.de

Ostateczna ocena wdrożenia nie byłaby pełna bez porównania z rozwiązaniami stosowanymi w podobnych systemach webowych. Bezpośredni konkurent, serwis *Otomoto*, przygotowany został jedynie w języku polskim, a więc pod każdym względem jest mniej przystępny dla obcokrajowców. O wiele ciekawsza w analizie jest wykorzystywana przez importerów samochodów używanych niemiecka autogiełda *Auto.de*, której interfejs udostępniono w 12 językach. Prawdopodobnie przyjęto w niej strategię internacjonalizacji wykorzystującą osobne instancje systemu, na co wskazuje adres z regionem określanym za pomocą poddomeny (patrz punkt 4.1.2) oraz zupełnie inny, zintegrowany z lokalnym magazynem motoryzacyjnym, wygląd interfejsu dla odbiorców z Niemiec. Pozostałe wersje językowe współdzielą prostszy szablon, działając przy tym jako niezależne aplikacje. Ubocznym skutkiem takiej architektury objawiającym się podczas zmiany języka jest konieczność przeładowania okna przeglądarki i utrata wprowadzonych przez użytkownika danych oraz przekierowanie na stronę główną nowej wersji językowej, co zmusza do ponownego, samodzielnego wyszukania wyświetlanego wcześniej ogłoszenia. To nieeleganckie pod względem użyteczności rozwiązanie spisuje się o wiele gorzej niż zastosowane w *Motomi*, które umożliwia swobodne przełączanie języka bez utraty danych, nawet podczas wypełniania formularzy. Podejście to zaskakuje tym bardziej, że struktura odnośników pozostaje w niezmienionej formie niezależnie od wybranych ustawień regionalnych, a po ręcznej podmianie kodu kraju w adresie strony nie następuje niepożądane przeniesienie na ekran główny. Same linki w przypadku obu systemów nie zostały zlokalizowane, jednak w takiej sytuacji powinny być prezentowane w uniwersalnym dla europejskiego odbiorcy (a do takiego kierowane są porównywane portale) języku angielskim, a nie jak w przypadku *Auto.de*, niemieckim.

Kontynuując analizę porównawczą serwisów wysnuto wniosek, że twórcy *Auto.de* zdecydowali się na wdrożenie jedynie podstawowej formy wielojęzyczności na bazie wariantu niemieckiego. Pomimo profesjonalnego przetłumaczenia głównych elementów

interfejsu, wiele tekstów pozostało niezmienione, dane liczbowe, ceny i daty wyświetlane są zawsze w formacie niemieckim, obrazki zawierają niemieckie napisy, a reklamy kierują do niemieckich banków i warsztatów. Nie uwzględniono również różnic w długości tłumaczeń, nawet w przypadku tak reprezentatywnego komponentu jakim jest selektor języka. Prowadzi to do sytuacji, w której wybór niektórych opcji uniemożliwia dalsze zmiany z powodu błędnego przemieszczenia listy rozwijanej. Trudności napotkają także internauci z krajów, w których preferuje się przeglądanie sieci za pomocą smartfonów. Witryna mobilna po serii kilku zapętlonych przekierowań ostatecznie zwraca błąd, a wersja dla komputerów stacjonarnych zakodowana została całkowicie na sztywno i wbrew regułom RWD (patrz punkt 3.1.3). Implementacja wielojęzycznego interfejsu w systemie *Motomi* ponownie wypada w tym porównaniu lepiej, ponieważ wolna jest od przytoczonych błędów, a wymienione powyżej aspekty lokalizacji podlegają w niej dostosowaniu w zależności od wybranego języka. Jedynym podobnie wykonanym mechanizmem w przypadku obu aplikacji jest tłumaczenie maszynowe treści ogłoszenia z przyciskiem pozwalającym przełączać się między wersją oryginalną i *Google Translate*.

5.3.2. Porównanie z systemem Amazon.com

Więcej podobieństw można dostrzec podczas porównania z systemami z pokrewnej branży e-handlu. Sklep *Amazon* wprowadził interesujące połączenie strategii internacjonalizacji za pomocą osobnych instancji systemu i dynamicznej podmiany języka w ramach jednej aplikacji. Poszczególne wersje korzystają z domen krajowych najwyższego poziomu i są skierowane wyłącznie do odbiorców z określonego w ten sposób terytorium. Są to witryny autonomiczne, z osobnymi bazami klientów i towarów, różnymi reklamami i promocjami, a łączy je głównie podobny, neutralny kulturowo interfejs (wyjątkiem są nieco bardziej kolorowe motywy azjatyckie, ale ogólny układ pozostaje ten sam). Jednocześnie można wyróżnić instancje, które pełnią rolę regionalnego, wielojęzycznego węzła współdzielonego przez kilka krajów, na przykład *Amazon.de* (język niemiecki, polski, czeski, niderlandzki i turecki) czy *Amazon.com* (język angielski i hiszpański). Użytkownik może co prawda przeglądać i porównywać produkty za pomocą zlokalizowanego interfejsu, a także widzi ceny przeliczone na właściwą sobie walutę, jednak ewentualna decyzja o zakupie skutkuje wysyłką towaru z Niemiec, przez niemieckiego sprzedawcę, któremu trzeba ostatecznie zapłacić w euro, a nie na przykład w koronach czeskich. Idea stojąca za tego typu podejściem jest tożsama z rozwiązaniem wdrożonym w systemie *Motomi*, w którym niezależnie od wybranego języka oferowane są samochody polskich sprzedawców, rozliczających się w złotówkach.

5.3.3. Porównanie z systemem eBay.com

Aukcje internetowe *eBay.com* również wykorzystują w pełni niezależne instancje systemu, na co wskazują różne adresy IP poszczególnych witryn. Niektóre z nich dzielą bazę ogłoszeń na lokalną i międzynarodową, dostępną jedynie w wybranych regionach. Witryny cechują się jednolitym wyglądem, ale generalnie różnią kategoriami i asortymentem oferowanych towarów. Wersja brytyjska pozwala na zakup samochodu, a filipińska jedynie jego zabawkowej makiety. Wyjątkowo potraktowano natomiast rynek południowokoreański i turecki, dla których funkcjonują osobne systemy pod zmienioną marką, odpowiednio: *gmarket.co.kr* i *gittigidiyor.com*. Różnice w wyglądzie ich interfejsów są fundamentalne, nie wynikają jednak z chęci uwzględnienia specyficznych aspektów kulturowych, tylko

z ekspansji na nowe rynki i przejęcia większości udziałów u dotychczasowych liderów aukcji internetowych w danym kraju [Eba11].

Poważnym błędem powtarzanym we wszystkich instancjach aplikacji jest selektor języka ukryty w stopce i niewidoczny poniżej pewnej rozdzielczości ekranu. Ponadto dostępne opcje wyświetla on w bieżącym języku strony, przez co nie spełnia swojego zadania. Aukcje *eBay* są również dobrym przykładem problemów z utrzymaniem spójności pomiędzy poszczególnymi kopiami systemu, jakie powstają w tej strategii internacjonalizacji. Zamiast serwisować jedną aplikację, konieczny jest nadzór nad 26 wersjami, które różnią się głównie detalami, a te można było zapewnić prostszymi technikami zaprezentowanymi na przykładzie *Motomi*. W efekcie niektóre warianty posiadają selektor ustawień regionalnych, inne nie, część prowadzi ostatecznie do wersji angielskiej, a niektóre nie działają poprawnie (np. wersja wietnamska, wyświetlająca błąd na głównej stronie).

5.3.4. Porównanie z systemem BBC.com

Najbardziej schludne i kompleksowe rozwiązanie pochodzi jednak z zupełnie innej branży – dziennikarskiej. W 2012 roku autorzy serwisu informacyjnego *BBC.com* zostali postawieni przed zadaniem przygotowania strony dostępnej w 30 językach świata, posługującej się pełnym przekrojem egzotycznych pism i alfabetów (m.in. hindi, syngaleskim, suahili czy perskim). Wyznaczony w tym celu zespół ekspertów dopracowywał pomysł przez lata, bazując na strategii dynamicznej podmiany języka w ramach jednej aplikacji. Wspólny kod źródłowy jest dostosowywany do potrzeb konkretnego regionu za pomocą konfiguracji znajdującej się w zewnętrznym pliku. Na jego podstawie narzędzia automatyzujące generują odpowiednio zmodyfikowane szablony HTML i style CSS. Skalowalność do potrzeb różnych krajów została osiągnięta przez podzielenie wyświetlanej zawartości na moduły (np. ostatnie wiadomości, sekcja artykułów, dział sportowy), które można konfigurować we wspomnianym wcześniej pliku (np. zmienić kolejność wyświetlania, liczbę przechowywanych elementów lub po prostu ukryć). Tłumaczenia przechowywane są w formacie JSON, fonty wczytywane dynamicznie na podstawie wybranego języka, dane liczbowe formatowane zgodnie z ustawieniami regionalnymi, a SCSS wykorzystywany do odbicia układu dla pism RTL. Duży nacisk położono też na zachowanie kompatybilności z urządzeniami mobilnymi i odpowiednią adaptację widoków dla mniejszych ekranów. W tym momencie można zauważyć, że rozwiązanie to pod względem podejścia do większości problemów pokrywa się de facto z implementacją wykonaną w badanym systemie *Motomi*. Więcej na temat szczegółów technicznych i historii wdrożenia wielojęzyczności w serwisie *BBC* można przeczytać w serii artykułów napisanych przez jego architekta na blogu *Responsive News* [Mas16].

5.3.5. Wnioski z porównania

Żaden z analizowanych systemów nie korzysta ze strategii pełnego tłumaczenia maszynowego, często pozostawiając w języku oryginalnym nawet opisy zamieszczane przez użytkowników. O ile w przypadku strony jako całości wydaje się być to zrozumiałe – duże i rozpoznawane marki nie mogą sobie pozwolić na tego typu niedoskonałe rozwiązania ze względów wizerunkowych, o tyle zastanawiające jest to w przypadku opisów produktu (zdecydowali się na to jedynie twórcy *Auto.de*). Przy tej skali systemów wynikać może to z nieznanych poza daną organizacją decyzji biznesowych lub marketingowych. Pomimo popularności strategii internacjonalizacji z osobnymi instancjami dla każdego języka, w żadnym systemie nie uświadczono globalnej bramy pozwalającej odwiedzającemu wybrać

region podczas pierwszej wizyty. Nie następuje także automatyczne przekierowanie na podstawie geolokalizacji czy ustawień przeglądarki, wydaje się zatem, że twórcy takich systemów polegają głównie na instynktach internautów, którzy odruchowo wpisują adresy zakończone domeną właściwą dla swojego kraju bądź klikają wyżej pozycjonowane wyniki w wyszukiwarkach, które najpierw prezentują strony w języku użytkownika. W ani jednej aplikacji nie wykonano też uporządkowania i lokalizacji odnośników, które niezależnie od regionu pozostają w języku angielskim lub w formie ciągu parametrów niezrozumiałych dla osoby nietechnicznej. Powszechne było posługiwanie się jednym, uniwersalnym projektem interfejsu graficznego, wspólnym dla wszystkich wariantów systemu. Wyjątek stanowiły regiony z azjatyckiego kręgu CJKV, które ze względu na swoją egzotyczną kulturę, potrzeby i potencjał biznesowy, otrzymywały przynajmniej częściowo zmodyfikowaną lub całkiem nową aplikację, jak koreański *eBay*. Problem pisma RTL został podjęty jedynie przez *BBC*, zatem na rynku międzynarodowych ogłoszeń internetowych istnieje luka czekająca na wypełnienie.

Podsumowując, dzięki wybranym technologiom i pomimo wspomnianych kompromisów, w systemie webowym *Motomi* udało się wykonać kompletną implementację wielojęzycznego interfejsu użytkownika, zgodną z celem badawczym niniejszej pracy. Rozwiązanie dorównuje technologicznie metodom używanym przez liderów z wieloletnim doświadczeniem w komunikacji międzynarodowej jak *BBC*, a bezpośrednią konkurencję z branży handlu samochodami pod wieloma względami na tym polu przewyższa.

6. Podsumowanie

Celem pracy był przegląd i porównanie technologii tworzenia interfejsów wielojęzykowych systemów webowych. Przedmiotem badań była natomiast analiza istniejącej aplikacji pod kątem przystosowania do komunikacji w wielu językach, wdrożenie wybranych mechanizmów wielojęzyczności i weryfikacja wykonanego za ich pomocą rozwiązania.

W rozdziale 2 wyjaśnione zostały różnice między kluczowymi terminami lokalizacji (czyli dostosowania interfejsu aplikacji do określonych wymagań językowych i kulturowych), internacjonalizacji (czyli sposobu przygotowania aplikacji do łatwego wdrożenia lokalizacji) i wielojęzyczności (czyli połączenia obu wcześniejszych technik, które umożliwia jednoczesne korzystanie z aplikacji przez różnojęzycznych użytkowników). Zapoznano też czytelnika z najważniejszymi wytycznymi organizacji W3C, ISO i Unicode, zajmujących się standaryzacją związanych z wielojęzycznością aspektów sieci Web oraz popularnymi sposobami kodowania znaków i przechowywania przetłumaczonych napisów.

W rozdziale 3 zawarto opis najczęstszych problemów występujących podczas lokalizacji oraz technik, za pomocą których można je rozwiązać. Uwzględniono tu aspekty kulturowe, projekt interfejsu i związane z nim ograniczenia techniczne, sposób prezentacji tekstu, odpowiedni format danych i elementy multimedialne. Podkreślona została rola tłumacza oraz optymalizacji dla wyszukiwarek internetowych. Wspomniano również o wykorzystaniu nowoczesnych technik takich jak *Flexbox Layout* czy międzynarodowe nazwy domen, problemie pisma RTL i sposobach uniwersalnej lokalizacji przy ograniczonym budżecie.

W rozdziale 4 podjęto temat różnych technik internacjonalizacji systemów webowych. Przedstawiono trzy strategie oraz związane z nimi sposoby strukturyzowania adresu strony: lokalizację w osobnych instancjach dla każdego języka, dynamiczne przełączanie ustawień regionalnych w ramach jednej aplikacji i tłumaczenie maszynowe za pomocą wtyczki firmy trzeciej. Metody zostały ze sobą zestawione i porównane pod kątem 14 kryteriów. Osobne instancje okazały się rozwiązaniem drogim i pracochłonnym, jednak najlepiej dopasowanym do potrzeb konkretnego rynku, tłumaczenia maszynowe stanowiły dla nich tanią i szybką w implementacji przeciwwagę, cechując się jednak niską ogólną jakością, a aplikacje z dynamiczną podmianą języka okazały się podejściem kompromisowym między wrażliwością na lokalny charakter kulturowy, a koniecznymi do poniesienia kosztami.

W rozdziale 5 opisano praktyczne wdrożenie technologii wielojęzyczności w rzeczywistym systemie webowym. Na potrzeby pracy podjęto współpracę z dysponentem polskiego serwisu z ogłoszeniami motoryzacyjnymi – *Motomi.pl*, którego funkcjonalność została rozszerzona o obsługę języków: angielskiego, greckiego i arabskiego. Implementację wykonano zgodnie z dobrymi praktykami lokalizacji opisanymi w rozdziale 3 i według założeń strategii internacjonalizacji za pomocą dynamicznej podmiany języka z rozdziału 4. Rozwiązanie zostało porównane z mechanizmami stosowanymi w największych systemach podobnego typu: *Otomoto*, *Auto.de*, *Amazon* i *eBay*, a także z liderem nowoczesnych rozwiązań wielojęzykowych, dostępnym w 30 językach serwisem informacyjnym *BBC*. Metody wdrożone podczas badań nad internacjonalizacją aplikacji *Motomi* dorównały technikom stosowanym przemysłowo w analizowanych systemach, a na polu dbałości o detale i pozytywne wrażenia użytkownika wypadły lepiej.

Wybrane podejście okazało się skuteczne i pozwoliło na zrealizowanie wszystkich celów niniejszej pracy. Zaproponowane mechanizmy umożliwiają kompleksowe spojrzenie na problem wielojęzyczności i są trafną odpowiedzią na zmiany zachodzące w ostatnich latach w lingwistycznym obrazie globalnej sieci wraz z popularyzacją Internetu. Dzięki szerokiemu zastosowaniu standardów i silnej izolacji warstwy prezentacji, opisane rozwiązanie jest uniwersalne i może zostać w łatwy sposób powielone także w innych aplikacjach webowych typu SPA.

Dalsze prace nad zagadnieniem wielojęzykowych interfejsów webowych powinny uwzględnić badanie użyteczności systemu przeprowadzone pod kątem potrzeb przedstawicieli konkretnych kultur. Należałoby również nawiązać współpracę z profesjonalnymi tłumaczami i ekspertami w dziedzinie kulturoznawstwa, aby móc opracować wyższej jakości treści, a także poznać sposoby ich pracy i zaprojektować wygodne narzędzia do rozwiązywania problemów na jakie trafiają podczas lokalizacji. Kierunkiem rozwoju technicznego byłaby próba wdrożenia analogicznych mechanizmów wielojęzyczności w aplikacjach bazujących na platformach webowych konkurencyjnych dla *AngularJS*, takich jak *React* i *Vue*, oraz na zyskujących popularność urządzeniach mobilnych.

Bibliografia

[App15] *Creating Right-to-Left User Interfaces*. W: *Internationalization and Localization Guide*, Apple Developer Guides and Sample Code, 2015, https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPInternational/SupportingRight-To-LeftLanguages/SupportingRight-To-LeftLanguages.html (dostęp 31.05.2018).

[Bnu14] *The New Table of General Standard Chinese Characters Issued*, School of Chinese Language and Literature, Beijing Normal University, Beijing, 2014, http://english.bnu.edu.cn/universitynews/52835.htm (dostęp 31.05.2018).

[Bur17] Burchardt A.: *Bitext*. W: *The Language of Localization*, red. Brown-Hoekstra K., XML Press, 2017. ISBN 978-1-937434-58-8.

[Coy18] Coyier C.: A *Complete Guide to Flexbox*, CSS-Tricks, 2018, https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (dostęp 31.05.2018).

[Cry97] Crystal D.: *English as a Global Language*, Cambridge University Press, New York, 1997, s. 1, 29, 86, 120-122. ISBN 978-0-511-07862-0.

[Cza17] *Kierunek: 120 tysięcy studentów obcokrajowców w 2020 roku!*, Czas internacjonalizacji, nr 3/2017.

[Dav08] Davis M.: *Moving to Unicode 5.1*. W: *Google Official Blog*, Google, 2008, https://googleblog.blogspot.com/2008/05/moving-to-unicode-51.html (dostęp 31.05.2018).

[Dea18] Dean J.: *Web Fonts*. W: *Web Programming with HTML5, CSS, and JavaScript*, Jones & Bartlett Learning, Burlington, 2018. ISBN 978-1-284-09180-9.

[Dea17] De Argaez E.: *Top 10 Languages Used in the Web*. W: *Internet World Stats*, Miniwatts Marketing Group, Bogota, 2017, http://www.internetworldstats.com/stats7.htm (dostęp 31.05.2018).

[Dei08] Deitel H. M., Deitel P. J.: *Rich Internet Applications (RIAs)*. W: *Deitel Developer Series AJAX, Rich Internet Applications, and Web Development for Programmers*, Prentice Hall Press, Boston, 2008. ISBN 978-0-13-714230-9.

[Eba11] *eBay Agrees to Acquire Shares in Turkey's GittiGidiyor*, eBay Stories, San Jose, 2011, https://www.ebayinc.com/stories/news/ebay-agrees-acquire-shares-turkeys-gittigidiyor/ (dostęp 31.05.2018).

[Ecm94] *ECMA-35 Character Code Structure and Extension Techniques*, European Association for Standardizing Information and Communication Systems, Geneva, 1994.

[Ecm17] *ECMA-404 The JSON Data Interchange Syntax*, European Association for Standardizing Information and Communication Systems, Geneva, 2017.

[Ede10] Edemariam A.: *Who Still Wants to Learn Languages?*, The Guardian, 25.08.2010.

[Eur18] *World Report on Internationalised Domain Names 2017*, EURid Insights, 2018, https://idnworldreport.eu/year-2017/ (dostęp 31.05.2018).

[Fil17] Filip D.: *XLIFF*. W: *The Language of Localization*, red. Brown-Hoekstra K., XML Press, 2017. ISBN 978-1-937434-58-8.

[Fil18] Filip D., Comerford T., Saadatfar S., Sasaki F., Savourel Y.: *XLIFF Version 2.1*, Organization for the Advancement of Structured Information Standards, 2018, http://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1-os.html (dostęp 31.05.2018).

[Fil05] Filipczyk B., Filipczyk G.: *Aspekty językowe w tworzeniu serwisów internetowych*, Akademia Ekonomiczna w Katowicach, Katowice, 2005.

[Fin17] Finelli R.: *Google Web Fonts*. W: *Mastering CSS*, Packt Publishing, Birmingham, 2017. ISBN 978-1-78728-158-5.

[Gai02] Gaither C.: *New Riddle for Xbox: Will It Play in Japan?*, The New York Times, 18.02.2002.

[Gil13] Gilbert D.: *Why Japanese Web Design Is So… Different*, Randomwire Blog, 2013, https://randomwire.com/why-japanese-web-design-is-so-different/ (dostęp 31.05.2018).

[Gil02] Gillam R.: *Unicode Demystified*, Addison-Wesley Professional, Boston, 2002. ISBN 978-0-201-70052-7.

[Gle16] Glezos D.: *Common File Formats for Localization*, Transifex Blog, 2016, https://www.transifex.com/blog/2016/common-localization-file-formats/ (dostęp 31.05.2018).

[Gou09] Gourley D., Totty B.: *Character Sets and HTTP*, *Multilingual Character Encoding Primer*. W: *HTTP: The Definitive Guide*, O'Reilly Media, Sebastopol, 2009. ISBN 978-1-56592-509-0.

[Gus16] *Rocznik statystyczny Rzeczypospolitej Polskiej*, Główny Urząd Statystyczny, Warszawa, 2016, s. 341, 361. ISSN 1506-0632.

[Hab11] Bel Habib I.: *Multilingual Skills Provide Export Benefits and Better Access to New Emerging Markets*, International Web Journal, nr 10/2011, s. 1-6, 17. ISSN 2104-3272.

[Har07] Haralambous Y.: *Fonts & Encodings*, O'Reilly Media, Sebastopol, 2007. ISBN 978-0-596-10242-5.

[Hil02] Hillier M.: *Multilingual Website Usability: Cultural Context*. W: *4th International Conference on Electronic Commerce*, City University of Hong Kong, Hong Kong, 2002.

[Hof10] Hofstede G., Hofstede G. J., Minkov M.: *Dimensions of National Cultures*. W: *Cultures and Organizations: Software of the Mind*, McGraw-Hill, 2010. ISBN 978-0-07-177015-6.

[Ibm06] *Locales*. W: *National Language Support Guide and Reference*, International Business Machines, 2006, s. 7-9.

[Ibm16a] *Guideline A3: UI Expansion*. W: *IBM Globalization Design Guidelines*, International Business Machines, 2016, https://www-01.ibm.com/software/globalization/guidelines/a3.html (dostęp 31.05.2018).

[Ibm16b] *Guideline B2: Terminology*. W: *IBM Globalization Design Guidelines*, International Business Machines, 2016, https://www-01.ibm.com/software/globalization/guidelines/b2.html (dostęp 31.05.2018).

[Ing17] Ingraham C.: *The Languages That Let You Say More with Less*, The Washington Post, 28.09.2017.

[Ish07] Ishida R.: *Text Size in Translation*, World Wide Web Consortium, 2007, https://www.w3.org/International/articles/article-text-size (dostęp 31.05.2018).

[Ish14a] Ishida R.: *Declaring Character Encodings in HTML*, World Wide Web Consortium, 2014, https://www.w3.org/International/questions/qa-html-encoding-declarations (dostęp 31.05.2018).

[Ish14b] Ishida R.: *Declaring Language in HTML*, World Wide Web Consortium, 2014, https://www.w3.org/International/questions/qa-html-language-declarations (dostęp 31.05.2018).

[Ish15] Ishida R., Miller S. K.: *Localization vs. Internationalization*, World Wide Web Consortium, 2015, https://www.w3.org/International/questions/qa-i18n.en (dostęp 31.05.2018).

[Ish16a] Ishida R.: *Why Use the Language Attribute?*, World Wide Web Consortium, 2016, https://www.w3.org/International/questions/qa-lang-why (dostęp 31.05.2018).

[Ish16b] Ishida R.: *Creating HTML Pages in Arabic, Hebrew and Other Right-to-Left Scripts*, World Wide Web Consortium, 2016, https://www.w3.org/International/tutorials/bidi-xhtml/ (dostęp 31.05.2018).

[Ish16c] Ishida R.: *CSS vs. Markup for Bidi Support*, World Wide Web Consortium, 2016, https://www.w3.org/International/questions/qa-bidi-css-markup (dostęp 31.05.2018).

[Ish16d] Ishida R.: *Unicode Bidirectional Algorithm Basics*, World Wide Web Consortium, 2016, https://www.w3.org/International/articles/inline-bidi-markup/uba-basics (dostęp 31.05.2018).

[Ish16e] Ishida R., Lanin A.: *Inline Markup and Bidirectional Text in HTML*, World Wide Web Consortium, 2016, https://www.w3.org/International/articles/inline-bidi-markup/ (dostęp 31.05.2018).

[Ish16f] Ishida R.: *Visual vs. Logical Ordering of Text*, World Wide Web Consortium, 2016, https://www.w3.org/International/questions/qa-visual-vs-logical (dostęp 31.05.2018).

[Iso10] *Language Codes – ISO 639*, International Organization for Standardization, Geneva, 2010, https://www.iso.org/iso-639-language-codes.html (dostęp 31.05.2018).

[Iso13a] *Country Codes – ISO 3166*, International Organization for Standardization, Geneva, 2013, https://www.iso.org/iso-3166-country-codes.html (dostęp 31.05.2018).

[Iso13b] *The International Language of ISO Graphical Symbols*, International Organization for Standardization, Geneva, 2013, s. 33. ISBN 978-92-67-10605-2.

[Iso17] *About ISO*, International Organization for Standardization, Geneva, 2017, https://www.iso.org/about-us.html (dostęp 31.05.2018).

[Iso91] *ISO/IEC 646:1991 7-bit Coded Character Set for Information Interchange*, International Organization for Standardization, Geneva, 1991, https://www.iso.org/standard/4777.html (dostęp 31.05.2018).

[Jon05] Jones R.: *Obfuscation: International Domain Names*, W: *Internet Forensics*, O'Reilly Media, Sebastopol, 2005. ISBN 978-0-596-10006-3.

[Jur15] Jurkiewicz Z.: *Kodowanie znaków narodowych*, Instytut Informatyki, Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, 25.05.2015.

[Kew12] Kew J., Leming T., Blokland E.: *Introduction*. W: *WOFF File Format 1.0*, World Wide Web Consortium, 2012, https://www.w3.org/TR/WOFF/ (dostęp 31.05.2018).

[Kuh16] Kuhnke E.: *Cross-Cultural Communication*. W: *Body Language*, John Wiley & Sons, Hoboken, 2016. ISBN 978-0-857-08704-1.

[Lay15] Layalk: *3 Different Ways to RTL your CSS*, RTL CSS Blog, 2015, https://rtl-this.com/tutorial/3-different-ways-rtl-your-css (dostęp 31.05.2018).

[Leh98] Lehtola A., Bounsaythip C., Tenni J.: *Controlled Language Technology in Multilingual User Interfaces*, VTT Information Technology, 1998, s. 1-2.

[Lon05] Longstaff A.: *Gregorian Calendar*. W: *Calendars From Around The World*, National Maritime Museum, London, 2005.

[Lun09] Lunde K.: *Font Formats, Glyph Sets, and Font Tools*. W: *CJKV Information Processing*, O'Reilly Media, Sebastopol, 2009. ISBN 978-0-596-51447-1.

[Mac80] MacKenzie C. E.: *Terms and Conditions*. W: *Coded Character Sets, History and Development*, Addison-Wesley, Reading, 1980. ISBN 978-0-201-14460-4.

[Mar01] Marcus A.: *Cross-Cultural User-Interface Design*, Aaron Marcus and Associates, New Orleans, 2001, s. 3-4.

[Mar03] Marcus A.: *Global and Intercultural User-Interface Design*. W: *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, red. Jacko J. A. i Sears A., Lawrence Erlbaum Associates, Mahwah, 2003. ISBN 0-8058-3838-4.

[Mas16] Maslen T.: *13 Tips for Making Responsive Web Design Multi-Lingual*, Responsive News Blog, 2016, http://responsivenews.co.uk/post/123104512468/13-tips-for-making-responsive-web-design (dostęp 31.05.2018).

[Mcg17] McGowan D.: *Why Japanese Web Design Is (Still) the Way It Is*, Moravia Blog, 2017, https://info.moravia.com/blog/why-japanese-web-design-is-still-the-way-it-is (dostęp 31.05.2018).

[Mdn18a] *text-emphasis*. W: *CSS Reference*, Mozilla MDN Web Docs, 2018, https://developer.mozilla.org/en-US/docs/Web/CSS/text-emphasis (dostęp 31.05.2018).

[Mdn18b] *Intl*. W: *JavaScript Reference*, Mozilla MDN Web Docs, 2018, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl (dostęp 31.05.2018).

[Mey18] Meyer E. A.: *writing-mode*. W: *CSS Pocket Reference, 5th Edition*, O'Reilly Media, Sebastopol, 2018. ISBN 978-1-492-03339-4.

[Moo16] Moore J., Klauzinski P.: *AngularJS and MVW*. W: *Mastering JavaScript Single Page Application Development*, Packt Publishing, Birmingham, 2016. ISBN 978-1-78588-164-0.

[Mor16] Morris K.: *The Guide to International Website Expansion: Hreflang, ccTLDs, & More!*, The Moz Blog, 2016, https://moz.com/blog/guide-to-international-seo (dostęp 31.05.2018).

[Mue10] Mueller J.: *Working with Multi-Regional Websites*. W: *Google Webmaster Central Blog*, Google, 2010, https://webmasters.googleblog.com/2010/03/working-with-multi-regional-websites.html (dostęp 31.05.2018).

[Nbp16] *Obywatele Ukrainy pracujący w Polsce – raport z badania*, Departament Statystyki NBP, Warszawa, 2016, s. 6-8.

[Nie93] Nielsen J.: *Usability Engineering*, Academic Press, San Diego, 1993, s. 123-129. ISBN 0-12-518406-9.

[Ogd40] Ogden C. K.: *Basic English: A General Introduction with Rules and Grammar*, Paul Trebner & Co., London, 1940.

[Ohy13] Ohye M.: *Expanding Your Site to More Languages*. W: *Google Webmasters*, Google, 2013, https://www.youtube.com/watch?v=8ce9jv91beQ (dostęp 31.05.2018).

[Ous12] Oustinoff M.: *English Won't Be the Internet's Lingua Franca*. W: *NET.Lang Towards The Multilingual Cyberspace*, red. Vannini L. i Crosnier H. L., C&F Editions, Caen, 2012. ISBN 978-2-915825-24-4.

[Pat17] Patni S.: *Introduction – XML, JSON*. W: *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*, Apress, New York, 2017. ISBN 978-1-4842-2664-3.

[Phi09] Phillips A., Davis M.: *BCP 47: Tags for Identifying Languages*, Network Working Group, 2009, https://tools.ietf.org/html/rfc4647 (dostęp 31.05.2018).

[Pow15] Powers S.: *Converting an ISO 8601 Formatted Date to a Date Object Acceptable Format*. W: *JavaScript Cookbook*, O'Reilly Media, Sebastopol, 2015. ISBN 978-1-491-90188-5.

[Rau11] Rau P. P., Plocher T., Choong Y.: *Cross-Cultural Web Design*. W: *Handbook of Human Factors in Web Design*, red. Vu K. L. i Proctor R. W., CRC Press, Boca Raton, 2011. ISBN 978-1-4398-2595-2.

[Rau17] Raulf C.: *Locale*. W: *The Language of Localization*, red. Brown-Hoekstra K., XML Press, 2017. ISBN 978-1-937434-58-8.

[Ras16] Rasmusson N.: *CSS Line-Clamp – The Good, the Bad and the Straight-up Broken*, Medium, 2016, https://medium.com/mofed/css-line-clamp-the-good-the-bad-and-the-straight-up-broken-865413f16e5 (dostęp 31.05.2018).

[Rob13] Robbins J.: *<bdi>*. W: *HTML5 Pocket Reference, 5th Edition*, O'Reilly Media, Sebastopol, 2013. ISBN 978-1-4493-6335-2.

[Ros17] Rosen A., Ihara I.: *Giving You More Characters to Express Yourself*. W: *Twitter Official Blog,* Twitter, 2017, https://blog.twitter.com/official/en_us/topics/product/2017/Giving-you-more-characters-to-express-yourself.html (dostęp 31.05.2018).

[San05] Sandrini P.: *Website Localization and Translation*. W: *Challenges of Multidimensional Translation*, red. Gerzymisch-Arbogast H. i Nauert S., Saarland University, Saarbrücken, 2005. ISBN 069-1306222.

[Sch15] Schweitzer S., Alexander L., Waisfisz B.: *Access to Asia: Your Multicultural Guide to Building Trust, Inspiring Respect, and Creating Long-Lasting Business Relationships*, John Wiley & Sons, Hoboken, 2015. ISBN 978-1-118-91901-9.

[Sco15] Scott E. A.: *What Is a Single-Page Application?*. W: *SPA Design and Architecture: Understanding Single-Page Web Applications*, Manning Publications, Shelter Island, 2015. ISBN 978-1-61729-243-9.

[She16] Shenoy A., Prabhu A.: *Introduction to SEO*. W: *Introducing SEO: Your Quick-Start Guide to Effective SEO Practices*, Apress, New York, 2016. ISBN 978-1-4842-1853-2.

[Smi06] Smith-Ferrier G.: *Machine Translation*. W: *.NET Internationalization: The Developer's Guide to Building Global Windows and Web Applications*, Addison-Wesley Professional, Upper Saddle River, 2006. ISBN 978-0-321-34138-9.

[Sni10] Snitker T. V., Jeffers J.: *User Research Throughout the World*. W: *Handbook of Global User Research*, red. Schumacher R. M., Morgan Kaufmann, San Francisco, 2010. ISBN 978-0-12-374852-2.

[Som10] Somssich R., Várnai J., Bérczi A.: *Study on Lawmaking in the EU Multilingual Environment*, Publications Office of the European Union, Luxembourg, 2010, s. 15-17. ISBN 978-92-79-17599-2.

[Sub12] Subocz D.: *Geert Hofstede – praktyczne zastosowanie wymiarów kultur narodowych*, Uniwersytet Marii Curie-Skłodowskiej, Lublin, 2012, s. 46-51.

[Sta04] Stanwick V., Fowler S.: *Appendix B: Quality Testing*. W: *Web Application Design Handbook*, Morgan Kaufmann, San Francisco, 2004. ISBN 978-1-55860-752-1.

[Tal16] Talesra A., Libby A., Gupta G.: *Introducing Responsive Web Design*. W: *Responsive Web Design with HTML5 and CSS3 Essentials*, Packt Publishing, Birmingham, 2016. ISBN 978-1-78355-307-5.

[Tex05] Texin T.: *User Interfaces for Right-to-Left Languages*, I18n Guy Home Page, 2005, http://www.i18nguy.com/MiddleEastUI.html (dostęp 31.05.2018).

[Tex10] Texin T.: *Origin of the Abbreviation i18n*, I18n Guy Home Page, 2010, http://www.i18nguy.com/origini18n.html (dostęp 31.05.2018).

[Tex11] Texin T.: *Comparing Characters in Windows-1252, ISO-8859-1, ISO-8859-15*, Quality Assurance for Software Internationalization, Localization and Globalization, 2011, http://www.i18nqa.com/debug/table-iso8859-1-vs-windows-1252.html (dostęp 31.05.2018).

[Tit05] Tittel E., Burmeister M.: *HTML4 for Dummies*, Wiley Publishing, Indianapolis, 2005, s. 24. ISBN 978-0-7645-8917-1.

[Uni17] *The Unicode Standard: Version 10.0.0 Core Specification*, Unicode Consortium, Mountain View, 2017. ISBN 978-1-936213-16-0.

[Usc15] *The 2015 Digital Future Report*, USC Annenberg School for Communication and Journalism, Los Angeles, 2015, s. 22-23, 43-53.

[Vaz15] Vázquez R. S.: *Exploring Current Accessibility Challenges in the Multilingual Web for Visually-Impaired Users*. W: *Proceedings of the 24th International Conference on World Wide Web*, ACM, Florence, 2015, s. 872. ISBN 978-1-4503-3473-0.

[Ver15] Vertommen K.: *Tips for Designing and Building a Multilingual Website*, Envato Tuts+, 2015, https://webdesign.tutsplus.com/articles/tips-for-designing-and-building-a-multilingual-website--cms-24708 (dostęp 31.05.2018).

[Web18] *Usage of UTF-8 for Websites*, W3Techs Web Technology Surveys, 2018, https://w3techs.com/technologies/details/en-utf8/all/all (dostęp 31.05.2018).

[Wit11] Witte C.: *Improved Usability through Internationalization*. W: *Design, User Experience, and Usability: Theory, Methods, Tools and Practice*, red. Marcus A., Springer,

Orlando, 2011, s. 114-118. ISBN 978-3-642-21674-9.

[Wor16] *Internationalization techniques: Developing specifications*, World Wide Web Consortium, 2016, https://www.w3.org/International/techniques/developing-specs (dostęp 31.05.2018).

[Yip13] Yip P.: *Why Did Facebook Change Its Thumb Icon?*. W: *OneSky Official Blog*, OneSky, 2013, http://www.oneskyapp.com/blog/facebooks-missing-thumb-may-just-design/ (dostęp 31.05.2018).

[Yul03] Yu L., Tang T. H.: *Culture and Design for Mobile Phones for China*. W: *Machines That Become Us: The Social Context of Personal Communication Technology*, red. Katz J. E., Transaction Publishers, New Brunswick, 2003, s. 196-198. ISBN 978-0-7658-0158-6.

[Yun03] Yunker J.: *Beyond Borders: Web Globalization Strategies*, New Riders, 2003. ISBN 978-0-7357-1208-9.

Indeks rysunków

Indeks tabel