



**WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA**
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA
Specjalność: Programowanie

Yuliia Hrynyk
Nr albumu studenta 58983

Projekt i implementacja systemu do zarządzania tłumaczeniami zawartości aplikacji internetowych

Promotor: dr inż. Mariusz Wrzesień

PRACA DYPLOMOWA INŻYNIERSKA

Rzeszów 2021

Ja niżej podpisany/a oświadczam, że składana przeze mnie praca dyplomowa pt. „Projekt i implementacja systemu do zarządzania tłumaczeniami zawartości aplikacji internetowych” została przygotowana samodzielnie. Oświadczam również, że praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni. Oświadczam ponadto, że niniejsza wersja pracy jest identyczna ze złożoną wersją elektroniczną.

.....
data czytelny podpis autora

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....
data czytelny podpis promotora

Spis treści

Wstęp	4
1 Wprowadzenie do problemu	6
1.1 Opis procesu dostarczania tłumaczeń.....	6
2 Wybór narzędzi i technologii do implementacji	9
2.1 SPA.....	9
2.2 REST API.....	10
2.3 JWT	11
2.4 HTML 5.....	11
2.5 SASS	12
2.6 TypeScript	13
2.7 Angular 9.....	14
2.8 RxJS.....	15
2.9 IndexedDB.....	15
2.10 Jest	15
2.11 Node.js.....	16
2.12 SQL Server	16
2.13 C#	17
2.14 .NET Core	18
2.15 Entity framework Core.....	18
3 Projektowanie systemu	19
3.1 Analiza wymagań	19
3.1 Specyfikacja wymagań.....	19
3.2 Diagram przypadków użycia.....	20
3.3 Diagramy sekwencji	22
3.4 Projekt bazy danych	24
3.5 Projekt interfejsu użytkownika.....	25
4 Implementacja.....	29
4.1 Implementacja aplikacji internetowej	29
4.2 Implementacja REST API.....	32
4.3 Implementacja serwisów do przechowywania tłumaczeń.....	33
4.4 Opis działania aplikacji	33
5 Testowanie	37
Zakończenie	39
Literatura.....	40
Streszczenie.....	42
Załączniki.....	43

Wstęp

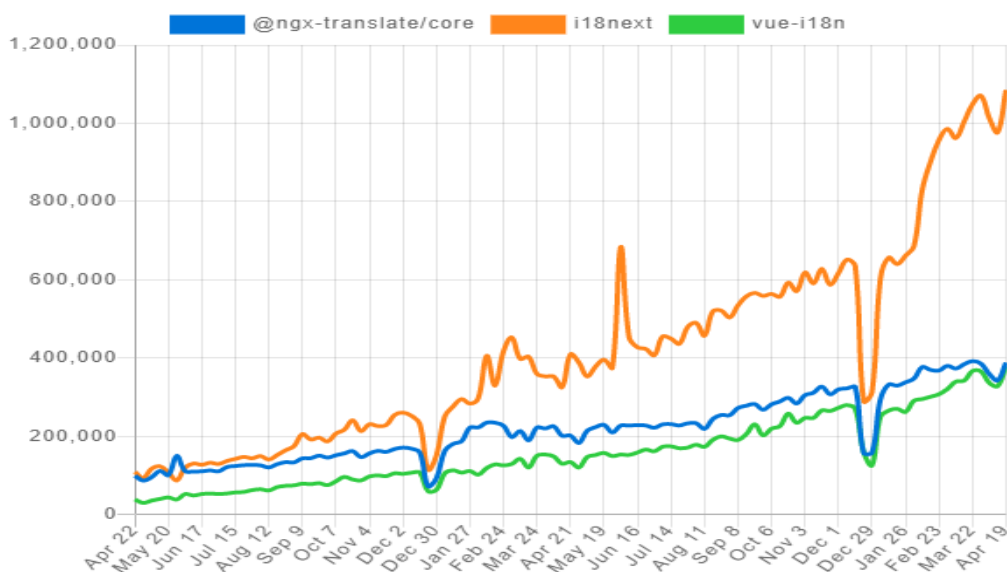
W dniu dzisiejszym konkurencja na rynku usług internetowych jest bardzo duża. Użytkownicy mają możliwość wyboru wśród dużej ilości serwisów, które rozwiązują ich problem. Ze względu na to niezbędne jest dbanie o jakość takich serwisów oraz dostarczanie najbardziej satysfakcjonującego produktu końcowego. Dlatego podczas projektowania oraz implementacji aplikacji, w centrum uwagi powinny być użytkownicy końcowi. Do najważniejszych kryteriów, które muszą być uwzględnione podczas wytwarzania serwisów internetowych, należą użyteczność oraz dostępność. W przypadku, gdy interfejs lub jego zawartość nie są zrozumiałe dla użytkownika, system traci na popularności, co często prowadzi do strat finansowych.

Ponieważ w chwili obecnej w dużym stopniu wzrasta sprzedaż towarów i usług za granicę danego kraju, system powinien być dostępny dla każdego użytkownika z całego świata. Usługodawcy dbający o dostępność ich produktów dla klientów z różnych krajów, zyskają na popularności.

W celu upraszczania korzystania z serwisów internetowych użytkownikom z różnych krajów, jest używana tak zwana globalizacja aplikacji, która zawiera procesy internacjonalizacji oraz lokalizacji. Internacjonalizacja aplikacji polega na izolacji kodu aplikacji od treści tłumaczeń, obsłudze różnych języków oraz dostarczeniu odpowiednich dla danego kraju formatów liczb oraz dat [WWW-1, 2020].

W praktyce proces internacjonalizacji polega na dostosowaniu zawartości interfejsu do konkretnego użytkownika. Większość aplikacji internetowych oraz mobilnych są dopasowywane do takiego procesu od początku wytwarzania. Programiści przygotowują zestawy danych, zawierające klucz, który identyfikuje konkretną część treści oraz tłumaczenie w odpowiednim języku. Pliki zawierające takie dane, są przekazywane do osób lub firm, zajmujących się tłumaczeniem [Yunker, 2017].

Dla uproszczenia procesu internacjonalizacji aplikacji są używane odpowiednio dobrane do konkretnego stosu technologicznego biblioteki. W ramach planowania oraz projektowania systemu, zostały przeanalizowane najbardziej popularne narzędzia z danego zakresu. Podczas implementacji aplikacji internetowych najczęściej są używane platforma Angular, Vue.js oraz React.js. Do bibliotek, wspomagających w procesie internacjonalizacji aplikacji w narzędziu Angular należą ngx-translate, i18next oraz moduł, i18n który dostarcza Angular. Dla biblioteki React najbardziej popularnym jest narzędzie react-i18next. W przypadku wyboru platformy Vue.js dostępny jest wbudowany moduł do internacjonalizacji. Na rysunku Rys. 1 przedstawiono tendencję wykorzystywania wyżej wymienione bibliotek w ciągu ostatnich 2 lat. Każde z wyżej wymienionych narzędzi oraz większość nowoczesnych narzędzi do wytwarzania aplikacji mobilnych oraz internetowych są oparte na obsłudze danych w postaci plików JSON.



Rys. 1 Statystyka wykorzystania bibliotek do tłumaczeń za ostatnie 2 lata

Źródło: <https://www.npmtrends.com/@ngx-translate/core-vs-i18next-vs-vue-i18n>, z dnia 24.04.2020

Po odpowiednim przygotowaniu danych, są one przekazywane zespołom zajmującym się tłumaczeniami. Proces ten często odbywa się w sposób bardzo niewygodny zarówno dla zespołów informatycznych, jak i dla tłumaczy. Nie każdy tłumacz jest w stanie się posługiwać plikami takich typu JSON. I praca z nimi jest bardzo czasochłonna i uciążliwa. Z tego powodu tłumaczenia najczęściej są dostarczane w postaci zwykłych plików tekstowych. W wyniku tego programiści spędzają czas na ręczne przenoszenie treści do odpowiedniego pliku. Jest to bardzo nieefektywny sposób, który wymaga automatyzacji.

Celem pracy dyplomowej jest zaprojektowanie oraz implementacja systemu do zarządzania tłumaczeniami. Produkt końcowy powinien rozwiązać wyżej opisany problem. Serwis powinien udostępniać użyteczny i prosty w obsłudze interfejs, który nie wymaga wiedzy z dziedziny informatyki. Narzędzie powinno umożliwić śledzenie zmian tłumaczeń oraz bezpośrednią integrację z własnymi serwisami informatycznymi. Serwis powinien umożliwić bezpośredni kontakt z tłumaczami.

Żeby osiągnąć założony cel, zostanie dokonana dokładna analiza problemu. Na podstawie tej analizy zostanie zaprojektowany system, zostaną wybrane odpowiednie dla danego problemu technologie i narzędzia oraz zostanie zaimplementowany i przetestowany dany system.

1 Wprowadzenie do problemu

W niniejszym rozdziale zostanie przedstawiony problem internacjonalizacji aplikacji oraz czynności związane z tym procesem.

Interfejs użytkownika to cały szereg tekstów, zdjęć, komunikatów oraz innych źródeł graficznych. Właśnie poprzez interfejs użytkownicy mogą korzystać ze wszystkich dostępnych funkcjonalności. Dlatego bardzo ważna jest zgodność treści interfejsu w każdym języku, w którym system jest dostarczany. Błędna lub nie zgodna z rzeczywistością treść aplikacji może doprowadzić do poważnych skutków i strat firmy.

Wytwarzanie oprogramowania z wielojęzycznym interfejsem użytkownika polega na podziale systemu na kod aplikacji oraz reprezentację treści w postaci plików ze zmiennymi i odpowiednimi wartościami. Podczas zmiany języka interfejsu przez użytkownika w odpowiednio oznaczonych miejscach zostaje podmieniana treść. W przypadku dobrze zaprojektowanej aplikacji dostarczenie nowego języka nie wymaga wytwarzania nowego oprogramowania, tylko niezbędne jest dodanie nowego zbioru danych dla danego języka [Hofmann-Delbor, Bartnicka, 2017].

System do zarządzania tłumaczeniami jest to aplikacja, która wprowadza automatyzację tego procesu, możliwość podglądu zmian, rozwiązuje problem związany z komunikacją między zespołami oraz problem związany z trudnościami posługiwania się plikami typu JSON. System ten jest to proste narzędzie zarówno dla osób, które nie pracują na co dzień z projektami informatycznymi jak i dla zespołów IT, które dokonują integrację własnych systemów z wyżej wspomnianym systemem do zarządzania tłumaczeniami.

Na rynku istnieją podobne produkty. Najbardziej popularnym rozwiązaniem z dostępnych na rynku jest system „Phrase”. System ten oferuje możliwość integracji z API, możliwość importowania plików z tłumaczeniami oraz dostarcza edytor tłumaczeń. Analizując dany system zostały wykryte braki, oraz ewentualne ulepszenia. System „Phrase” nie dostarcza możliwości bezpośredniego nawiązywania kontaktu z tłumaczami. Jest to ważna funkcjonalność, która wspomaga zredukować czas poszukiwania i w taki sposób czas wytwarzania aplikacji. Drugim ważnym aspektem jest użyteczność systemu. System „Phrase” oferuje dosyć skomplikowany interfejs użytkownika, co jest jedną z wad danego serwisu.

Rozwiązanie wyżej wymienionych problemów powinien zapewnić, że zostanie zredukowany czas zarządzania zespołem, czas implementacji, czas dostarczania tłumaczeń oraz czas testowania. Kierownik projektu uzyska pełną kontrolę nad obecnym stanem tłumaczeń, będzie miał dostęp do statystyki oraz historii dokonywanych zmian. Specjalista dziedziny zapewniania jakości będzie mógł w prosty sposób sprawdzić poprawność integracji serwisu z systemem udostępniającym tłumaczenia. Zadaniem programisty będzie tylko przygotowywanie jednego pliku. W przypadku potrzeby powrotu do poprzedniej wersji tłumaczeń, nie będzie potrzeby ponownego przeprowadzenia procesu przenoszenia tekstów, tylko wycofanie do wybranej wersji poprzez administratora projektu.

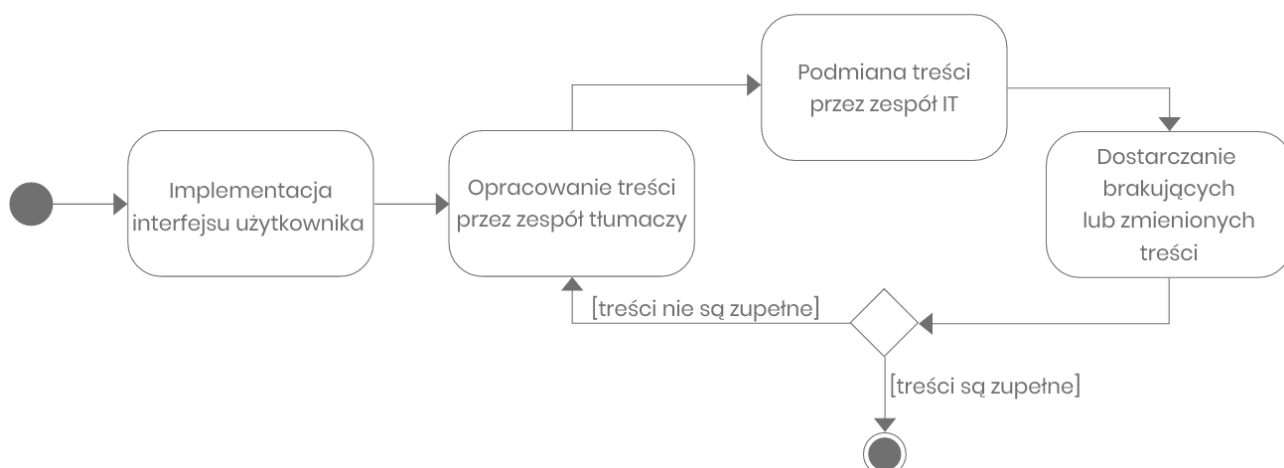
1.1 Opis procesu dostarczania tłumaczeń

Proces dostarczania oraz podmiiany tłumaczeń w przeciwieństwie do procesu wytwarzania oprogramowania jest procesem prostym. Ze względu na to proces taki nie jest uwzględniany podczas przygotowywania kosztorysu projektu. Jest to błędne założenia i często prowadzi do wyczerpania założonego czasu oraz powoduje trudności w przestrzeganiu harmonogramu. Programiści spotykają się z problemami w tym procesie. Dostarczone tłumaczenia rzadko są w postaci, która jest korzystną dla specjalisty dziedziny IT (Rys. 2).

Cykl dostarczania oraz integracji wersji językowej można podzielić na takie fazy jak:

1. Implementacja projektu graficznego interfejsu użytkownika w postaci działającego prototypu;
2. Przekazywanie projektu graficznego interfejsu użytkownika do zespołu tłumaczy;
3. Opracowanie treści przez zespół tłumaczy;
4. Ręczna podmiana każdej wartości w odpowiednim miejscu zbioru tłumaczeń;

5. Opracowanie brakujących treści oraz przejście do punktu 3;
6. Proces ten jest cykliczny, ponieważ interfejs użytkownika może być rozszerzany, nowe języki mogą być dodawane, ze względu na to są dodawane nowe wartości do plików z tłumaczeniami.

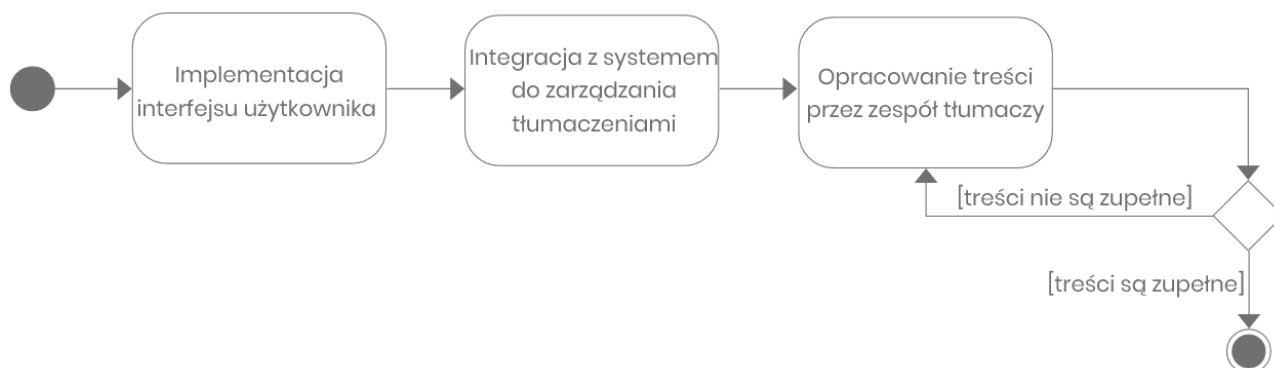


Rys. 2 Standardowy proces dostarczania tłumaczeń za pośrednictwem
Źródło: opracowanie własne

Zaproponowany system powinien zmniejszyć te fazy oraz zredukować czas przejścia pomiędzy fazami (Rys. 3). Proces dostarczania tłumaczeń przy użyciu danego systemu będzie wyglądał następująco:

1. Implementacja projektu graficznego interfejsu użytkownika w postaci działającego prototypu;
2. Rejestracja projektu w systemie do zarządzania tłumaczeniami;
3. Opracowanie treści przez zespół tłumaczy;

Dostarczanie tłumaczeń za pośrednictwem systemu do zarządzania tłumaczeniami upraszcza ten proces oraz skraca czas dostarczania materiałów.



Rys. 3 Proces dostarczania tłumaczeń za pośrednictwem danego systemu
Źródło: opracowanie własne

Podczas korzystania z takiego rozwiązania do przechowywania i przetwarzania tłumaczeń, jest wyeliminowany krok związany komunikacją pomiędzy zespołami, która jest czasochłonnym procesem, oraz krok związany z przenoszeniem danych do systemu. W taki sposób zaoszczędzony czas jest wykorzystywany na wytwarzanie oprogramowania.

Dostarczanie wielojęzycznych rozwiązań informatycznych ma dużo zalet. Na przykład takie podejście zwiększa liczbę potencjalnych odbiorców, może pomóc zwiększyć sprzedaż produktu czy usługi, a także zwiększa dostępność aplikacji czy strony internetowej. Są to jednak nie wszystkie korzyści, które oferują witryny wielojęzyczne.

Jedną z największych zaletą witryn wielojęzycznych jest to łatwiejsza optymalizacja SEO. Dostarczanie systemów w języku angielskim, oczywiście, umożliwia korzystanie z niego dla dużego procentu osób

z całego świata. Ze względu na to wiele stron oraz aplikacji internetowych czy mobilnych nie są dostarczane w wersji wielojęzycznej, jak widać, a ponad 60% serwisów WWW są dostarczane tylko w języku angielskim [WWW-6, 2020]. Istnieje jednak wiele korzyści w dostarczaniu oprogramowania nie tylko w języku angielskim, ale również w innych językach. Do takich zalet należy:

1. Rozszerzanie audytorium;
2. Zwiększenie sprzedaży produktów i usług. Użytkownicy są bardziej zainteresowani w korzystaniu z witryn, które udostępniają treści w ich języku ojczystym;
3. Zwiększenie konkurencyjności. Wprowadzając wielojęzyczny system, zyskamy na większym zainteresowaniu wśród dużego audytorium.

Używając tylko i wyłącznie język angielski jako język interfejsu użytkownika, tracona jest duża ilość potencjalnych klientów. Oczywiście język angielski jest najbardziej popularnym i najbardziej rozpowszechnionym językiem, ale dla osób, dla których język angielski nie jest językiem ojczystym, korzystanie z interfejsów niedopasowanych do jego potrzeb, jest mniej użyteczny i użytkownicy tracą zainteresowanie w korzystaniu z takich serwisów. Wybierając język angielski jako jedyny język do przekazywania informacji w takiego rodzaju serwisach, uwzględniano tylko 25% użytkowników serwisów internetowych [WWW-5, 2020].

Optymalizacja aplikacji webowej pod kątem pozycjonowania jest to proces, który zapewnia widoczność witryny na stronach wyszukiwania. Wielojęzyczność poprawia SEO pod kątem takich aspektów [WWW-4, 2020]:

1. Witryna wielojęzyczna w prosty sposób może być dopasowywana do konkretnego regionu;
2. Wielojęzyczność witryny optymalizuje jej ranking;
3. Im więcej osób przegląda witrynę, tym lepszy ranking.

Wnioskując z powyższej informacji, globalizacja aplikacji jest ważnym etapem w procesie wytwarzania aplikacji, który powoduje, że serwis staje się bardziej dostępny oraz użyteczny dla osób z całego świata. Częstym powodem pominięcia tego procesu jest czas oraz koszty danego procesu. Ze względu na to serwis rozwiązujący dany problem jest przyszłościowy.

2 Wybór narzędzi i technologii do implementacji

W tym rozdziale zostaną opisane wybrane narzędzia do implementacji projektowanej aplikacji, zostaną również przeanalizowane ich wady oraz zalety.

W wyniku analizy wymagań zarówno funkcjonalnych, jak i нефункциональных (opisane w kolejnym rozdziale) zostały wybrane technologie, które umożliwiają wytwarzanie realizowanej aplikacji. Nad danymi rozwiązaniami pracują takie duże firmy jak Microsoft oraz Google.

Projekt zostanie podzielony na 2 główne części: aplikację internetową w postaci SPA oraz aplikację po stronie serwera w postaci REST API. W procesie wytwarzania aplikacji zostaną wykorzystane środowiska do edycji kodu Visual Studio Code, dodatkowe narzędzia oraz wtyczki takie jak Prettier, TSLint oraz GitLens, które ułatwiają proces wytwarzania aplikacji webowych.

Front-end jest to wizualna część aplikacji, z której bezpośrednio korzysta użytkownik. Część ta została zaprojektowana przy użyciu technologii webowych takich jak Angular 9, HTML 5, SASS oraz języku TypeScript.

REST API jest to aplikacja po stronie serwera, za pośrednictwem której będą dokonywane wszystkie zmiany związane z systemem bazodanowym. Dla wytwarzania danej części aplikacji została wykorzystana technologia .NET Core 3.1.

W procesie tworzenia projektu zostaną przeprowadzane testy jednostkowe oraz testy systemowe. Do każdego cyklu wytwarzania danego oprogramowania niezbędne są odpowiednie narzędzia, które mają swoje zalety oraz rozwiązują konkretny problem.

2.1 SPA

SPA (*ang. Single Page Application – Aplikacje jednostronicowe*) lub aplikacja jednostronicowa to rodzaj aplikacji internetowych, która wymaga załadowania pojedynczej strony HTML oraz połączonych plików JavaScript. Dzięki dynamicznemu odświeżaniu przy pomocy JavaScript, wszystkie inne strony, które odwiedza użytkownik w ramach danej aplikacji nie muszą być przeładowywane. Dzięki danemu podejściu użytkownik przez cały czas korzystania z aplikacji ma możliwość interakcji oraz uzyskuje niezbędną informację o takich procesach jak załadowywanie nowej treści, wysyłaniu formularza, wczytywaniu plików itd.

Aplikacje typu SPA zapewniają dobre doświadczenie użytkownika. Aplikacje jednostronicowej desktopowej, która błyskawicznie reaguje na wszystkie działania użytkownika. Efekt ten można osiągnąć wykorzystując takie narzędzia jak Angular, React, Vue, Ember i inne.

Aplikacje jednostronicowe w dniu dzisiejszym są bardzo popularnym rozwiązaniem dla wielu problemów z dziedziny usług internetowych. SPA mają następujące zalety:

1. Prędkość ładowania strony. Aplikacje jednostronicowe wszystkie zasoby ładowane podczas pierwszego zapytania do serwera, a w trakcie interakcji poprzez interfejs użytkownika są niezbędne dane doładowywane oraz różnego rodzaju pliki graficzne.
2. Lepsze doświadczenie użytkownika. SPA są szybkie, co jest jednym z najważniejszych czynników, które sprawiają, że użytkownicy korzystają chętniej z takiego rodzaju rozwiązania.
3. Łatwość rozwoju. Dzięki takim narzędziom takim jak Electron, Ionic oraz React Native aplikacje jednostronicowe w bardzo prosty sposób mogą być użyte do implementacji międzyplatformowych aplikacji mobilnych oraz desktopowych.

Aplikacje jednostronicowe oprócz dużej ilości zalet mają również swoje wady. Najczęściej są spotykane następujące problemy związane z SPA:

1. Optymalizacja SEO. Aplikacje jednostronicowe często są gorzej indeksowane przez wyszukiwarki. Problem ten występuje ponieważ aplikacje są dynamicznie generowane po stronie klienta. Natomiast w dniu dzisiejszym wyszukiwarka Google jest w stanie przeprocesować strony typu SPA.

2. Dłuższe załadowywanie strony podczas pierwszego uruchomienia. Za względu na to, że większość funkcjonalności aplikacji jednostronicowych jest procesowana przez przeglądarkę użytkownika, są dostarczane duże pliki JavaScript. Nowe wersje frameworków takie jak Angular, React oraz Vue umożliwiają tak zwane ładowanie leniwe w celu podziału kodu JavaScript na mniejsze części, co rozwiązuje dany problem.
3. Niekompatybilność z różnymi wersjami przeglądarek. Większość narzędzi do implementacji SPA są zaimplementowane w najnowszych wersjach JavaScript, które nie są obsługiwane przez wielu przeglądarek. Natomiast w większości przypadków kod aplikacji SPA jest kompilowany do innych wersji JavaScript przy pomocy kompilatora Babel, który umożliwia wskazanie konkretnej wersji, w której aplikacja będzie dostarczana.

Analizując wyżej wymienione wady i zalety rozwiązań typu SPA, widać, że większość problemów, które są związane z takiego rodzaju aplikacjami mogą być wyeliminowane [WWW-2, 2020].

2.2 REST API

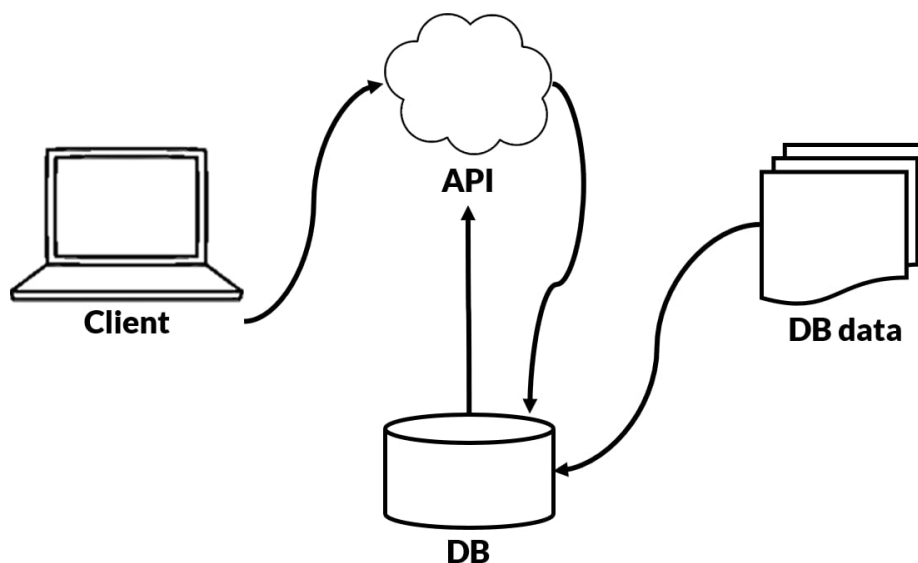
REST staje się powszechnym podejściem do dostarczania usług zewnętrznym systemom. Takie podejście jest bardzo popularne ze względu na prostotę, łatwość i przejrzystość obsługi oraz możliwość wielokrotnego używania.

Ze względu na sposób korzystania z REST API, nie ma zależności platformowej. Aplikacje, które korzystają z REST API mogą być zarówno aplikacjami webowymi, jak i mobilnymi oraz desktopowymi. Podstawą działania REST są to HTTP zapytania. Aplikacja po stronie klienta wysyła zapytania do API w celu uzyskania niezbędnych danych lub modyfikacji danych (Rys. 4) [WWW-3, 2020].

W zależności od rodzaju żądania, które powinno być dokonane, do serwera są wysyłane 4 główne metody:

- GET – metoda do pobierania danych;
- POST – metoda do dodawania nowych danych;
- UPDATE – metoda do aktualizacji danych;
- DELETE – metoda do usuwania danych.

Żeby zapewnić bezpieczeństwo korzystania z REST API, w niektórych przypadkach jest niezbędne wprowadzanie autoryzacji i autentyfikacji użytkowników. Autentykacja polega na weryfikacji tożsamości osoby próbującej uzyskać dostęp do interfejsu API. Natomiast na etapie autoryzacji sprawdzane są uprawnienia użytkownika do korzystania przez użytkownika z poszczególnych funkcjonalności [WWW-7, 2020].



Rys. 4 Schemat działania systemu przy użyciu REST API

Źródło: <https://blog.yellowant.com/rest-api-calls-made-easy-7e620e4d3e82>, z dnia 23.06.2020

Podczas projektowania REST API muszą być uwzględnione dobre praktyki implementacji takiego rodzaju serwisów. Najczęstsze błędy oraz poprawne przykłady są opisane w [Richardson, Ruby, 2011].

2.3 JWT

Podczas tworzenia REST API czasami niezbędna jest autentykacja i autoryzacja użytkownika. Ze względu na to aplikacja kliencka musi zadbać o przy każdym wysyłanym zapytaniu do API.

Najłatwiejszym i najbardziej niebezpiecznym sposobem uwierzytelnienia w REST jest to wysłanie identyfikatora użytkownika oraz hasła przy każdym żądaniu. Takie podejście grozi przechwyceniem danych użytkownika, co może prowadzić do poważnych skutków.

Bardziej bezpiecznym rozwiązaniem jest identyfikowanie użytkownika na podstawie unikatowego dla danego użytkownika w określonym czasie ciągu symboli. Żeby zapewnić większe bezpieczeństwo, identyfikator ten nie może ciągle pozostawać bez zmian.

JWT (*ang. JSON Web Token*) jest dobrym rozwiązaniem danego problemu. Oprócz zabezpieczenia aplikacji, JWT umożliwia przechowywanie informacji o użytkowniku, która nie może być podmieniona bez klucza, który jest przechowywany po stronie serwera. W taki sposób nie ma potrzeby przechowywania danego identyfikatora, ponieważ wszystkie niezbędne informacje o użytkowniku mogą być uzyskane z tego ciągu symboli [WWW-8, 2020].

2.4 HTML 5

HTML (*ang. Hyper Text Markup Language*) jest to język służący do tworzenia struktury aplikacji webowych. HTML zawiera szereg elementów, które są używane do tworzenia struktury strony oraz powiązań pomiędzy poszczególnymi elementami. Elementy HTML mogą być również używane do zmiany wyglądu tekstu tak jak kursywę, zwiększenie lub zmniejszenie czcionki itd.

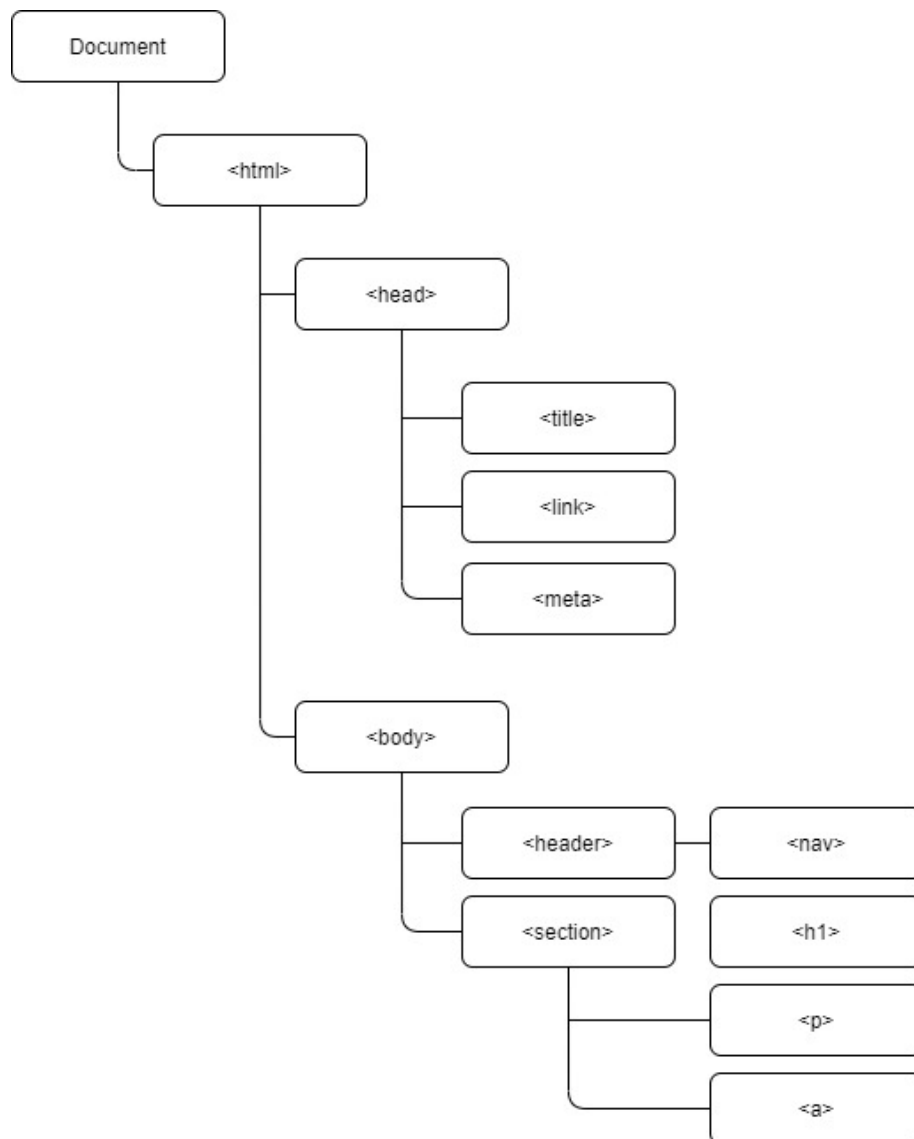
Podczas załadowania strony internetowych jest generowany DOM (*ang. Document object model- Obiektowy model dokumentu*). Struktura DOM powstaje z elementów HTML. Główną etykietą w strukturze dom jest to etykieta „html”. W następnej kolejności podawane są etykiety „head” oraz „body”. W środku etykiety „head” są ustawienia takie jak nagłówek strony, opis, słowa kluczowe oraz linki do plików z stylami (Rys. 5). Bardziej dokładnie struktura DOM oraz syntaksa HTML jest opisane w [WWW-13, 2021].

HTML jest w dniu dzisiejszym bardzo popularnym narzędziem do wytwarzania stron oraz aplikacji internetowych. Wytwarzanie stron internetowych bez użycia dodatkowych CMS narzędzi ma dużo zalet:

1. Witryny HTML mają znacznie mniejszą wagę.
2. Witryny „napisane” w języku HTML działają i ładują się znacznie szybciej, ponieważ zawierają tylko niezbędne elementy i pliki.
3. Więcej możliwości dla twórcy witryny.

Wady witryn korzystających ze statycznych stron HTML obejmują kilka punktów, które w niektórych sytuacjach mogą być bardzo istotne, a nawet decydujące.

1. Dłuższy proces wprowadzania zmian.
2. Brak panelu administracyjnego, w którym informacje o stanie serwisu lub dodatkowe koszty związane z dodaniem takiego panelu.
3. Utrzymywanie takiej witryny wymaga znajomości HTML oraz innych narzędzi, które są wykorzystywane w danym serwisie.



Rys. 5 Przykład prostej struktury DOM
Źródło: opracowanie własne

2.5 SASS

SASS (*ang. Syntactically Awesome Style Sheets*) jest to CSS preprocesor, służący do pisania stylów przy użyciu odpowiedniej składni danego narzędzia. Kod źródłowy SASS jest kompilowany do kodu CSS (*ang. Cascading Style Sheets*) z uwzględnieniem kompatybilności kodu z różnymi urządzeniami i wersjami CSS. SASS umożliwia wybór składni SCSS lub niezależnej składni SASS. W przypadku danego projektu będzie używana składnia SCSS ze względu na możliwość używania nawiasów klamrowych, które robią kod bardziej ustrukturyzowany.

1. Nie zależnie od wybranej składni język SASS udostępnia następujące narzędzia:
2. Zmienne do przechowywania danych takich jak wartości liczbowych, kolorów, czcionek itd.
3. Funkcje, które umożliwiają obliczanie niezbędnych wartości poprzez zdefiniowane przez programistę przeliczenia.
4. Dziedziczenie, dzięki któremu możemy używać wielokrotnie zdefiniowane przez nas style dla podobnych elementów w aplikacji.
5. Importowanie zewnętrznych stylów, funkcji i zmiennych z bibliotek zewnętrznych.

Dzięki narzędziu SASS pliki źródłowe projektu są mniejsze oraz bardziej czytelne. Wytwarzanie interfejsów użytkownika z użyciem danego narzędzia jest również szybsze oraz bardziej skuteczne [WWW-12, 2021].

2.6 TypeScript

TypeScript jest to język programowania rozwijany przez zespół Microsoft. TypeScript w przeciwieństwie do JavaScript, do którego TypeScript jest kompilowany, wprowadza dużą kontrolę nad kodem, zmniejsza ryzyko popełniania błędów i powoduje, że kod jest bardziej czytelny i prostszy w utrzymaniu.

Praca nad pierwszą wersją TypeScript była rozpoczęta w roku 2012 jako OpenSource projekt. Od początku język ten sprawił duże zainteresowanie wśród JavaScript programistów. Oczywiście jest dużo sceptyków jeżeli chodzi o język TypeScript, ponieważ kod wyjściowy jest to nadal JavaScript. Łatwo jednak zauważyć, że podczas wytwarzania JavaScript aplikacji z użyciem TypeScript ilość błędów logicznych jest znacznie mniejsza, ponieważ kompilator od razu pokazuje wszystkie pomyłki związane z niepoprawnym używaniem typów czy struktur danych.

Język TypeScript wprowadza wzór OOP oraz wzór programowania funkcjonalnego. To programista decyduje, z którego z nich chce korzystać, jednak podczas wytwarzania oprogramowania, które jest zgodne z zasadą SOLID, jest używany wzór OOP.



Rys. 6 Kompilacja kodu TypeScript do kodu JavaScript
Źródło: <https://devopedia.org/typescript>, z dnia 05.12.2020

Oprócz wyżej wymienionych zalet TypeScript warto wspomnieć również o tym, że skompilowany z TypeScript kod jest kompatybilny z większością przeglądarek. Programista jedynie musi podać wersję ECMAScript, do której kod powinien być skompilowany (Rys. 6).

TypeScript ma dużo zalet nad tradycyjnym językiem JavaScript. Główne atuty TypeScript są następujące:

1. TypeScript jest silnie typowanym językiem. Syntaksa TypeScript jest łatwiejsza do zrozumienia dla programistów Java i C #.
2. TypeScript implementuje wiele koncepcji OOP, takich jak dziedziczenie, polimorfizm, hermetyzacja i modyfikatory dostępu.
3. Potencjał języka sprawia, że wytwarzanie dużych i złożonych rozwiązań jest łatwiejsze i bardziej bezpieczne niż w przypadku używania standardowego JavaScript, ponieważ wiele błędów jest eliminowane w procesie kompilacji.

Do głównych wad TypeScript można odnieść następujące właściwości:

1. W procesie wytwarzania oprogramowania mamy do czynienia z takimi plikami jak „.ts”, „.d.ts”, „.map”, „.js”. Jest to bardzo dużo dodatkowych plików, co nie jest wygodne w przypadku małych projektów.

2. Nie wszystkie przeglądarki obsługują debugowanie TypeScript poprzez narzędzia deweloperskie, które udostępnia przeglądarka.
3. Wiele natywnych klas, o których programista musi być poinformowany. Na przykład zamiast jednej klasy Event istnieją takie klasy jak MouseEvent, TouchEvent, KeyboardEvent itd.

Typescript jest dobrym rozwiązaniem dla zarówno dużych jak i małych projektów, ponieważ zapewnia większą kontrolę nad kodem aplikacji [Fenton, 2017].

2.7 Angular 9

Angular jest to JavaScript framework do wytwarzania aplikacji internetowych. Angular aplikacje są szybkie oraz wydajne.

Angular jest technologią, która dostarcza nam narzędzi do budowy aplikacji internetowej wraz ze strukturą i organizacją kodu projektu. W przeciwieństwie do biblioteki ReactJS Angular dostarcza większość narzędzi niezbędnych do zaimplementowania całkowitej aplikacji.

Angular umożliwia podział aplikacji na komponenty oraz moduły. W taki sposób aplikacja jest podzielona na osobne niezależne elementy, które między sobą komunikują przy pomocy zdefiniowanych parametrów.

Oprócz wyżej wymienionych zalet narzędzi Angular warto wspomnieć o silniku do wstrzykiwania zależności. Dzięki DI kod wytwarzanej aplikacji jest zgodny z piątą zasadą SOLID - odwrócenia zależności. Narzędzia, które oferuje Angular wraz z przykładami oraz dobrymi praktykami związanymi z ich użyciem są opisane w [Murray, Coury i inni, 2018].

Angular jak i każde inne narzędzie ma swoje plusy i minusy. Główne zalety Angular są następujące:

1. Angular jest używany razem z Typescript.
2. Wiersz poleceń Angular (Angular CLI) oferuje prosty sposób do automatycznego generowania szablonów komponentów, modułów, serwisów i innych.
3. Narzędzie do generowania nowych projektów ze wszystkimi niezbędnymi ustawieniami i standardami.
4. Dobrej jakości dokumentacja, która pozwala deweloperowi uzyskać wszystkie niezbędne informacje w jednym miejscu.
5. MVVM (Model-View-ViewModel), który umożliwia programistom niezależną pracę nad podobnymi sekcją aplikacji przy użyciu odpowiedniego zestawu danych.
6. Dependency Injection. Jest to jedna z największych zalet Angulara nad innymi narzędziami i wspomaga przy implementacji dużych i skomplikowanych projektów.

Angular jest dobrym narzędziem do wytwarzania dużych, skalowalnych aplikacji. W dniu dzisiejszym Angular jest również często używany w mniejszych projektach, ponieważ jest ciągle rozwijany i najnowsze wersje zapewniają, że zbudowane aplikacje są mniejsze, jest on również bardziej wydajny ze względu na zmianę środowiska do śledzenia i wizualizacji zmian. Natomiast Angular ma również swoje wady. Wśród najbardziej zauważalnych minusów tego narzędzi jest:

1. Duża ilość rodzajów struktur (Injectables, Components, Pipes, Modules, itd.). Ze względu na to trudniejszy jest proces nauczania się tego narzędzi w porównaniu z React i Vue.js.
2. Stosunkowo niska wydajność, biorąc pod uwagę różne metryki. Z drugiej strony można to łatwo rozwiązać za pomocą tak zwanej strategii śledzenia zmian (Change Detection Strategy), która oferuje większą kontrolę nad procesem śledzenia zmian i ich wizualizacji.

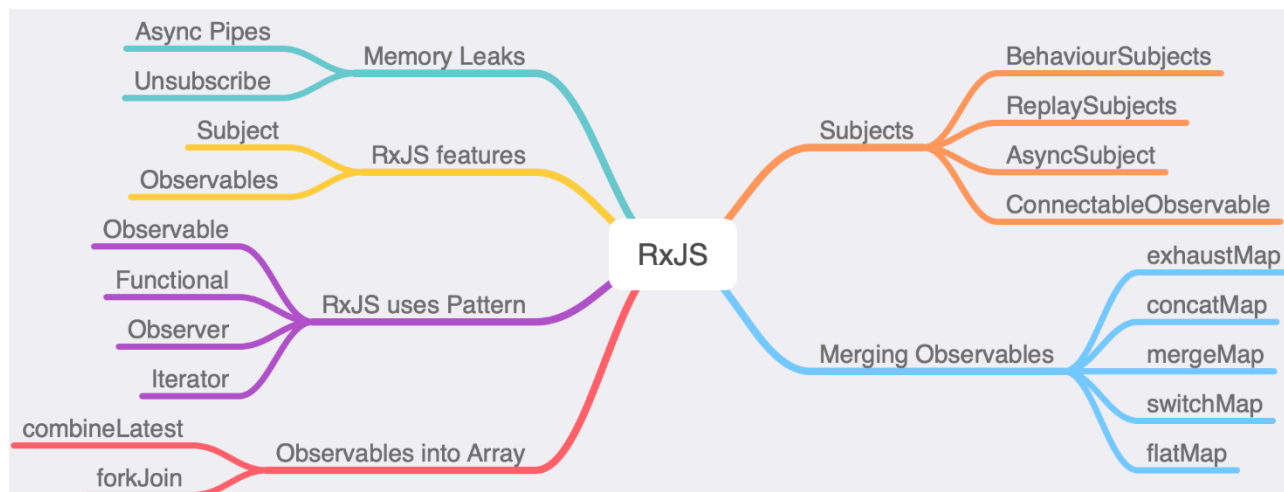
Dla doświadczonego programisty Angular jest bardzo dobrym narzędziem. Każdy framework ma swoje wady i zalety, natomiast umiejętność posługiwania się wszystkimi dostępnymi narzędziami, które oferuje Angular, zapewnia, że wytworzona aplikacja będzie bardziej skalowalna oraz wydajna [Murray, Coury i inni, 2018].

2.8 RxJS

RxJS (*ang. Reactive Extensions for JavaScript*) jest to narzędzie które umożliwia obsługę asynchronicznego JavaScript kodu na podstawie programowania reaktywnego.

Programowanie reaktywne jest to programowanie bazujące na asynchronicznych strumieniach danych. W przypadku aplikacji internetowych asynchronicznymi są np. kliknięcia, wprowadzanie tekstu do elementów formularza, przesunięcie kursora myszy, obsługa AJAX zapytań i inne.

RxJS oferuje duży zestaw funkcji do łączenia, tworzenia i filtrowania asynchronicznych strumieni danych, umożliwia wytwarzanie obsługi własnych strumieni danych itd. (Rys. 7)



Rys. 7 Podstawowe API RxJS

Źródło: <https://levelup.gitconnected.com/rxjs-a-deep-dive-with-angular-8-7fe9fd675aaa>, z dnia 27.11.2020

Obsługa kodu asynchronicznego przy użyciu RxJS zapewnia lepszą czytelność oraz przejrzystość kodu. W przypadku aplikacji, które bazują na dużej ilości asynchronicznych przepływach danych, RxJS jest dobrym rozwiązaniem [WWW-9, 2020].

2.9 IndexedDB

IndexedDB to obiektowy magazyn danych przeglądarki, który jest wydajniejszy oraz bardziej niezawodny od magazynów typu klucz – wartość takich jak LocalStorage oraz SessionStorage.

W ramach jednej aplikacji mogą być wytwarzane różne instancje baz IndexedDB. Każda baza danych ma unikalną nazwę. Każdy obiekt przechowywany w IndexedDB musi mieć klucz, przy pomocy którego pobierany z magazynu. IndexedDB może automatycznie generować unikalne klucze dla każdego obiektu dodanego do bazy danych.

Oprócz możliwości pobierania obiektów za pomocą klucza, istnieje również możliwość wyszukiwania danych według innych właściwości obiektu. Żeby umożliwić takie wyszukiwanie, niezbędne jest dodawanie indeksów na podstawie danej właściwości. Każdy indeks definiuje dodatkowy klucz dla przechowywanych obiektów.

IndexedDB gwarantuje niepodzielność operacji. Poszczególne działania na bazie danych są łączone w transakcje, więc albo wszystkie zakończą się sukcesem, albo żadna z nich nie powiedzie się, a baza danych nie będzie w częściowo zmienionym stanie [Camden, 2013].

2.10 Jest

Jest to środowiskiem uruchomieniowym do testów JavaScript kodu. Pakiet „Jest” dostarczany jest jako pakiet NPM. Dane narzędzie może być używane zarówno dla aplikacji React jak i dla aplikacji napisanych w Angular czy Vue.js oraz node.js. Ze względu na to „Jest” jest jednym z najbardziej popularnych narzędzi do testowania JavaScript aplikacji.

Kod testów jest zamieszczany w plikach z rozszerzeniem .spec. Taka konwencja nazewnictwa została zapożyczona w języku Ruby.

W plikach .spec testy są grupowane przy pomocy metody „describe”. Metoda ta najczęściej dzieli strukturę na mniejsze części: metody klas, elementy modułu. W środku metody describe są podawane wszystkie przypadki testowe przy użyciu odpowiedniej metody.

W celu zdefiniowania zdarzeń, które mają być wykonywane przed lub po każdym przypadku testowym, używamy funkcji takie jak „beforeEach”, „beforeAll”, „afterEach”, „afterAll”.

Dobłą praktyką w testach automatycznych (jednostkowych oraz integracyjnych) jest wykorzystywanie różnych rodzajów obiektów testowych w celu izolowania kodu.

Większość bibliotek, wymagają wcześniejszego zdefiniowania rodzaju danego obiektu, Jest natomiast reprezentuje wszystko jako jeden wszystko obiektem typu “Mock”.

„Mock” jest dostępny i może być używany na różne sposoby w kodzie w zależności od potrzeb, co w dużym stopniu upraszcza proces pisania testów automatycznych [WWW-14, 20201].

2.11 Node.js

Node.js jest to środowisko do uruchomienia JavaScript kodu. W dniu dzisiejszym Node.js jest jednym z najbardziej popularnych narzędzie do tworzenia wydajnych i skalowalnych REST API, aplikacji mobilnych, aplikacji desktopowych a również jest coraz częściej używany w dziedzinie AI oraz IoT. Zarówno przeglądarka jak i Node.js uruchamiają JavaScript kod w środowisku wykonawczym V8 [Herron, 2016].

Aplikacje internetowe napisane zgodnie z architekturą klient – serwer obsługuje każde zapytanie wysłane z aplikacji klienta, wysyłając odpowiednie zasoby. Po zakończeniu żądania połączenie jest zamykane. Przy otrzymaniu nowego żądania, tworzony jest nowy wątek do jego obsługi. Jeżeli serwer ma ograniczoną ilość potoków, które mogą być obsłużone jednocześnie, może dojść do sytuacji, gdy dla nowego żądania nie będzie dostępnego wątku.

Dany problem jest rozwiązany w środowisku Node.js. Dana platforma wykorzystuje model sterowany zdarzeniami. Node.js używa nieblokujących operacji wejścia – wyjścia, co znaczy, że operacje te nie blokują główny wątek. Ze względu na to serwer nadal będzie mógł obsługiwać żądania.

Ze względu na sposób obsługi zdarzeń asynchronicznych w Node.js, jest on bardzo szybki i jest dobrym rozwiązaniem dla dużej ilości problemów.

Node.js ma również swoje wady. Środowisko Node.js nie jest dobrym rozwiązaniem w przypadku skomplikowanych żądań, które wymagają dużej mocy obliczeniowej. Dokonując takiego rodzaju żądania, główny wątek jest zablokowany, co znaczy, że serwis nie jest w stanie obsługiwać kolejne zadania. W dniu dzisiejszym Node.js rozwiązuje dany problem, wprowadzając wielowątkowość [Brooks, Grow, Craig, Short, 2018].

2.12 SQL Server

SQL Server to relacyjny system do zarządzania bazami danych wytworzony i rozwijany przez zespół Microsoft.

SQL serwer udostępnia szereg serwisów, z dziedziny inżynierii baz danych oraz narzędzia związane z dziedziną analizy biznesowej.

Od niedawna firma Microsoft oferuje kilku odmian SQL Server, żeby klienci mogli wybrać taką, która im najbardziej odpowiada. Obecnie firma oferuje trzy odmiany produktu SQL Server, które wewnętrznie są nazywane ABC: A dla Appliance, B dla Box oraz C dla Cloud.

1. Appliances. Ideą tego produktu jest dostarczenie kompletnego rozwiązania obejmującego sprzęt, oprogramowanie oraz usługi.

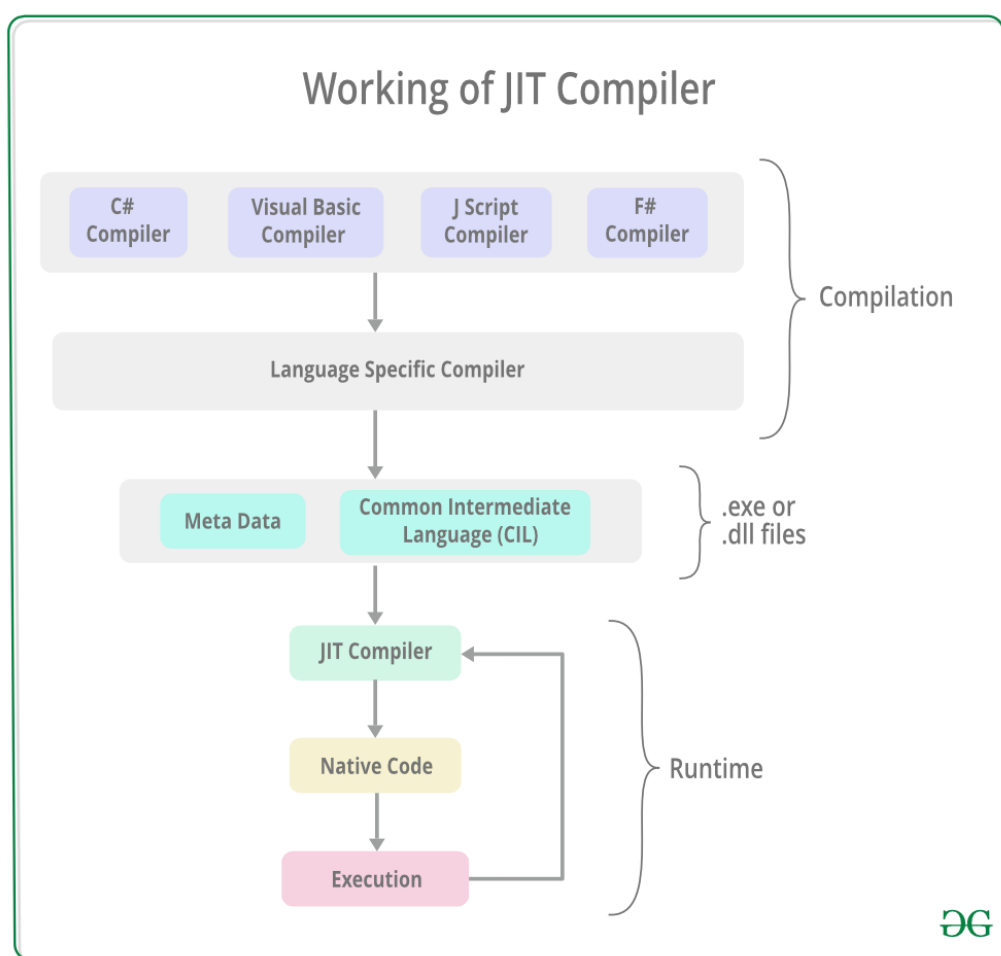
2. Box. W przypadku tego rodzaju produktu, który formalnie jest określany jako on-premise SQL Server, klient jest odpowiedzialny za wszystko – sprzęt, instalacje oprogramowania, i obsługę aktualizacji oraz tworzenie kopii zapasowych.
3. Cloud. Microsoft oferuje dwa rodzaje produktów SQL Server w chmurze: prywatny i publiczny. Chmura publiczna posiada nazwę Windows Azure SQL Database [Ben-Gan, 2012].

W ciągu ostatnich 10 lat SQL Server przeszedł od rozwiązania dla małych i średnich systemów zarządzania bazami danych do potężnej platformy. SQL Server uwzględnia wszystkie współczesne wymagania dotyczące pracy z i jest coraz bardziej popularnym narzędziem do budowy systemów bazodanowych.

Jedną z głównych wad używania Microsoft SQL Server przeciwieństwie do wielu innych systemów takich jak na przykład MySQL, PostgreSQL, są to koszty związane z licencją danego produktu. Choć korzystanie z danego narzędzi do celów programistycznych lub edukacyjnych jest bezpłatne, każdy rodzaj użytku biznesowego wiąże się z opłatą licencyjną [McQuillan, 2013].

2.13 C#

C# jest to język zorientowany obiektowo o składni podobnej do języka C, C++ oraz Java. Jako OOP język C # umożliwia polimorfizm, dziedziczenie oraz enkapsulację. Ze względu na podejście obiektowe oraz przejrzystą składnię C# pozwana rozwiązywać duże problemy, implementować zaawansowane systemy i dużych, ale jednocześnie elastycznych, skalowalnych i rozszerzalnych aplikacji. C # cały czas jest rozwijany wprowadza nową funkcjonalność, która upraszcza pracę programistom ale również upraszcza jakość kodu oraz oprogramowania.



Rys. 8 Schemat procesu JIT kompilacji

Źródło: <https://media.geeksforgeeks.org/wp-content/uploads/20190410185504/Working-of-JIT-Compiler1.png>,
z dnia 16.12.2020

Programy napisane w języku C# działają na platformie .NET. Platforma .NET wprowadza zestawie bibliotek klas, które nie są dostępne dla aplikacji napisanych w języku C#, ale również mogą być używane w programach w języku F#, C++ oraz innych językach platformy.

Kod C# jest kompilowany na części aplikacji z rozszerzeniami exe lub dll. W momencie, gdy aplikacja jest uruchamiana oraz kompilowana przy użyciu tak zwanej kompilacji JIT (Rys. 8). Kompilacja niezbędnej części kodu odbyła się w momencie odwołania się. Po zakończeniu danego procesu kod zostaje zapisany i przy ponownym odwołaniu się do kodu nie będzie ponownej kompilacji, co sprawia, że aplikacja jest bardziej wydajna [Skeet, 2013].

2.14 .NET Core

.NET Framework istnieje już od wielu lat. W czasie jego powstania obejmował on tylko .NET, przy pomocy którego można było tworzyć aplikacje desktopowe dla systemu operacyjnego Windows oraz aplikacje internetowe. Następnie pojawiły się inne implementacje .NET, takie jak Xamarin, które służyły do wytwarzania aplikacji mobilnych dla platform Android oraz IOS. Ze względu na to, że .NET Framework nie był wystarczająco elastyczny, zaczęto pracę nad standaryzacją.

.NET Core to najnowsza Open Source implementacja .NET, który wspiera wiele systemów operacyjnych. .NET Core umożliwia tworzenie wieloplatformowych aplikacji konsolowych, usług w chmurze oraz aplikacji webowych. W związku z tym, że projekt został zaprojektowany zgodnie z zasadami open source, nad platformą .NET Core pracowało około 10 000 programistów. Uwzględniając najlepsze praktyki .NET Core został przekształcony w platformę która jest dostępna i efektywna dla programistów.

.NET Core został zaprojektowany wieloplatformowa otwarta wersja .NET Framework, dużo rzeczy różni te 2 narzędzia. Aplikacje oparte na technologiach Windows nie są wspierane przez .NET Core., ale aplikacje konsolowe oraz webowe mogą być obsługiwane zarówno przez .NET Core, jak i przez .NET Framework.

.NET Core API jest mniejszy niż .NET Framework, ale będzie się rozwijał wraz z rozwojem. Ponadto .NET Core implementuje tylko niektóre podsystemy platformy .NET Framework w celu obsługi uproszczonego i elastycznego projektu platformy. Te różnice mogą w pewnym stopniu ograniczać .NET Core, ale korzyści związane z tym, że jest to platforma wieloplatformowego oraz z otwartym kodem źródłowym przeważają [Esposito - 2018].

2.15 Entity framework Core

Entity Framework Core to zorientowany obiektowo ORM (*ang. object-relational mapping*) firmy Microsoft. EF Core wprowadza wyższy poziom abstrakcji podczas pracy z bazami danych.

Zadaniem Entity Framework Core jest przechowywanie danych, reprezentujące obiekty .NET w bazie danych i ponowne ich późniejsze pobieranie. Innymi słowy, Entity Framework Core działa jako warstwa między aplikacją a bazą danych.

W celu przechowywania obiektów .NET w bazie danych, Entity Framework Core konwertuje obiekty do postaci, która może być wykorzystana w poleceniach SQL przez serwer bazodanowy. W celu zachowania spójności pomiędzy różnymi serwerami baz danych, Entity Framework Core dostarcza odpowiednie narzędzia dla poszczególnych wersji. Entity Framework Core, umożliwia również pobieranie danych i reprezentowanie ich w postaci obiektów NET. Entity Framework Core przy pomocy narzędzie LINQ do wykonywania zapytań w bazie danych, co umożliwia pracę z kolekcjami [Freeman, 2018].

3 Projektowanie systemu

W tym rozdziale opisany zostanie projekt danego systemu, zostaną zdefiniowane wymagania funkcjonalne oraz нефункционалне, zostaną przeanalizowane najważniejsze przypadki użycia oraz procesy występujące w systemie.

3.1 Analiza wymagań

Analiza wymagań jest to proces, który prowadzi do decyzji, co powinien oferować produkt końcowy. Od tego, czy dokładnie została ona przeprowadzona zależy cały projekt. W celu wydedukowania wymagań funkcjonalnych oraz нефункционалных, została przeprowadzona analiza świata rzeczywistego, analiza dobrych praktyk programowania oraz analiza zaleceń z tematu bezpieczeństwa.

Jako opcję dodatkową system powinien wprowadzić konwertowanie do typu XML, PHP Array. Dla wygody tłumaczy możliwe jest również generowanie oraz importowanie plików typu XLSX. Bezpośrednia integracja z API powinna być dostępna dla typu JSON oraz XML. Inne rodzaje mogą być tylko i wyłącznie importowane i eksportowane.

W trakcie wytwarzania oprogramowania programiści przygotowują pliki, które zawierają niezbędne klucze i wartości tłumaczeń. Ręczne wprowadzanie tych wartości do systemu jest czasochłonne i nieefektywne. Z tego powodu system powinien również umożliwiać importowanie przygotowanych wcześniej plików JSON.

Grupą docelową danego systemu są osoby zatrudnione w dziedzinie IT oraz tłumacze, którzy dokonują zlecenia związane z tą dziedziną. System powinien być wygodny dla obu rodzajów użytkowników. Z tego powodu została podjęta decyzja, że system powinien oferować interfejs, który udostępnia bardziej abstrakcyjny sposób pracy z plikami takiego rodzaju jak JSON oraz XML.

Nie wszystkie zmiany, które wprowadzi tłumacz mogą być wydane do aplikacji produkcyjnej, ponieważ jest ryzyko, że treść wprowadzona przez tłumacza jest niezgodna z tym, co oczekuje kierownik projektu. Z tego powodu jest zapotrzebowanie we wprowadzeniu dodatkowego poziomu zabezpieczenia takiej sytuacji, a mianowicie wprowadzenie ręcznej akceptacji zmian przez administratora projektu.

Dla uproszczenia pracy kierownika projektu, powinna być dostępna możliwość kontaktu z osobą czy firmą tłumaczącą oraz możliwość oceny pracy tłumacza dla ułatwienia wyboru dla innych osób, które również są zainteresowane w usługach tłumaczy. System taki powinien być wygodny dla każdego użytkownika. Interfejs nie powinien być zbyt techniczny. Każda osoba powinna być w stanie się nim posługiwać.

Produkt końcowy powinien być zrealizowany w postaci aplikacji internetowej, podzielonej na Frontend w postaci tak zwanego SPA oraz Backend w postaci REST API po stronie serwera. Takie podejście powinno zmniejszyć czas oczekiwania przed rozpoczęciem interakcji z interfejsem użytkownika. SPA również ma swoje wady, które również mają być uwzględniane, a mianowicie jest to problem z SEO oraz problem przy dzieleniu się linkami. System również powinien gwarantować bezpieczeństwo danych użytkownika. Hasła mają być przechowywane w bazie danych w sposób zgodny z zaleceniami ekspertów dziedziny bezpieczeństwa w sieci internetowej. Do tego należy przechowywanie haseł nie w postaci łańcucha oryginalnego, tylko w postaci tak zwanego hasza. Dla procesu haszowania haseł musi być używany unikalny łańcuch symboli oraz odpowiedniej ilości iteracji.

3.1 Specyfikacja wymagań

Wymagania funkcjonalne są podstawą każdego projektu informatycznego. W celu dostarczenia produktu końcowego niezbędna jest realizacja funkcji opisanych w tym obszarze. Wymagania funkcjonalne opisują działania, które mogą być wykonywane przez użytkowników danego systemu. W wyniku analizy została wydedukowana następująca lista wymagań funkcjonalnych:

Tab. 1. Wymagania funkcjonalne

ID	Opis wymagania	Priorytet
1	Uwierzytelnienie oraz rejestracja	Wysoki
2	Dodawanie projektu	Wysoki
3	Dodawanie użytkowników do projektu	Wysoki
4	Nadawanie uprawnień użytkownikom	Średni
5	Importowanie pliku .JSON, .XLSX z tłumaczeniami w postaci klucz-wartość	Mały
6	Dodawanie nowych rekordów do tłumaczeń	Wysoki
7	Dodawanie nowego języku do tłumaczenia	Wysoki
8	Usunięcie klucza do tłumaczenia	Średni
9	Wprowadzanie tłumaczeń	Wysoki
10	Zapisywanie zmian tłumaczeń	Wysoki
11	Przeglądanie historii zmian	Mały
12	Przeglądanie statystyki zmian	Mały
13	Eksportowanie tłumaczeń w postaci pliku XLSX	Średni
14	Eksportowanie tłumaczeń w postaci PHP zmiennych	Średni
15	Eksportowanie tłumaczeń w postaci JSON	Średni
16	Dodawanie komentarzy do tłumaczenia	Średni
17	Wyszukiwanie tłumaczy	Średni
18	Ocena pracy tłumaczy	Średni

Źródło: opracowanie własne

Wymagania niefunkcjonalne opisują, wymagania związane z częścią wizualną, z bezpieczeństwem aplikacji, ograniczeniami oraz wydajnością.

Tab. 2. Wymagania niefunkcjonalne

ID	Opis wymagania	Priorytet
	Hasła powinny być haszowane przy użyciu unikatowego ciągu symboli	Wysoki
2	Token autoryzacyjny powinien wygasać po upływie 1 dnia	Wysoki
3	System powinien być skalowalny	Średni
4	Aplikacja po stronie klienta powinna być zrealizowana w postaci SPA	Wysoki
5	Aplikacja powinna udostępniać podwójną walidację: po stronie Frontendu oraz po stronie API	Wysoki
6	Aplikacja webowa powinna być dostępna dla użytkowników przeglądarek Google Chrome (od wersji 12.0.0), Mozilla Firefox (od wersji 68.0.0)	Średni

Źródło: opracowanie własne

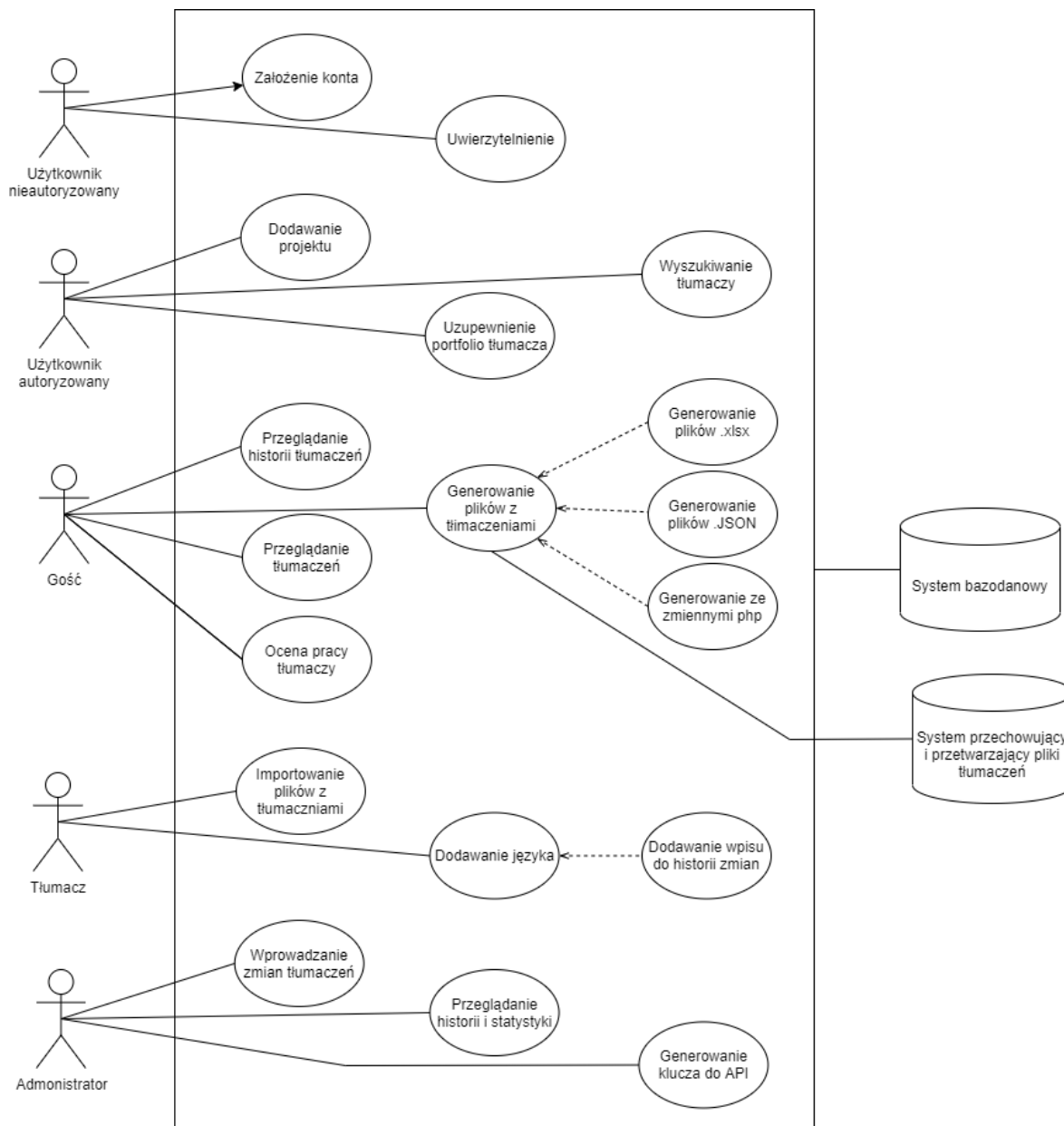
3.2 Diagram przypadków użycia

W następnej kolejności projektowania aplikacji został opracowany diagram przypadków użyciu, który jest bardzo ważnym elementem procesu projektowania aplikacji, ponieważ wizualizuje wszystkie przypadki, które muszą być uwzględnione podczas kolejnych etapów projektowania oraz podczas procesu implementacji rozwiązania.

W systemie występują użytkownicy końcowi, system bazodanowy oraz system do przechowywania i przetwarzania plików z tłumaczeniami. Użytkownicy, którzy nie są zarejestrowani i zalogowani w systemie nie mają uprawnień do głównych funkcjonalności. Z tego względu nie są w stanie dodawać nowe

projekty, nie mają dostępu do listy istniejących projektów, nie mają możliwości korzystać z edytora tłumaczeń, nie mają uprawnień do wyszukiwania i kontaktu z tłumaczami.

Po założeniu konta użytkownik ma dostęp do głównej deski użytkownika. Dzięki temu użytkownik może rozpocząć swoją pracę. Użytkownik taki może założyć nowy projekt. W tym przypadku użytkownik występuje w roli administratora projektu i ma uprawnienia do edytowania treści projektu, dodawania członków zespołu oraz zmiany ich uprawnienia. Administrator projektu ma również dostęp do statystyki oraz historii zmian projektu. Taki użytkownik ma również uprawnienia do modyfikacji ustawień projektu (Rys. 9).



Rys. 9 Diagram przypadków użycia
Źródło: opracowanie własne

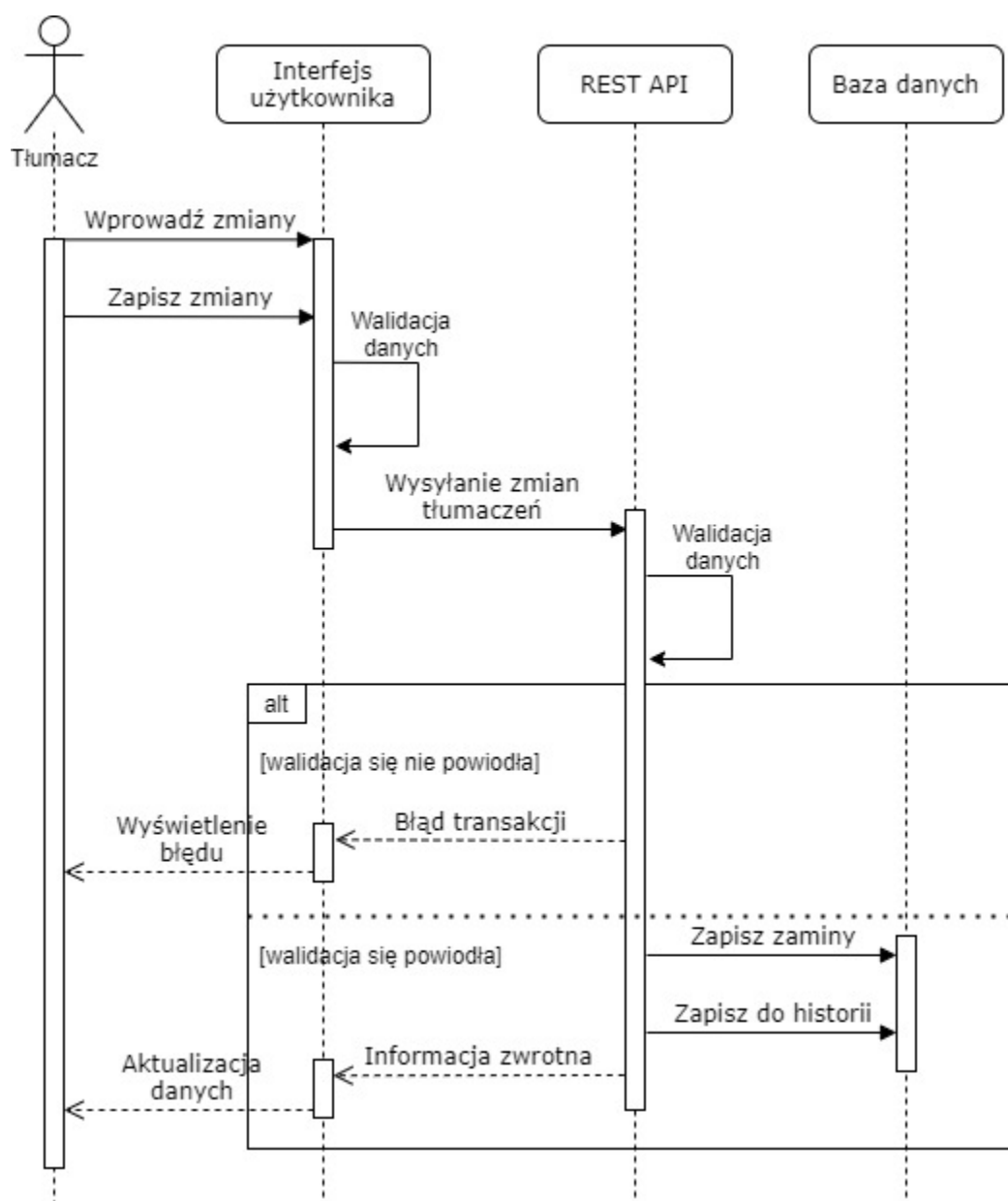
Użytkownik, który został dodany do projektu, może dokonywać jedną z następujących ról:

1. Gość projektu – jest to użytkownik, który nie może dokonywać żadnych modyfikacji w ramach danego projektu. Taki użytkownik ma uprawnienia tylko i wyłącznie do przeglądania treści tłumaczeń, generowania plików w odpowiednim formacie i dodawanie oraz czytanie komentarzy innych uczestników.

2. Tłumacz – jest to użytkownik, który ma wszystkie wyżej wymienione uprawnienia gości, ale może również dokonywać zmian w treściach tłumaczeń, zmieniać widoczność poszczególnych rekordów oraz usuwać rekordy. Po dokonaniu modyfikacji tłumaczeń poprzez interfejs użytkownika, dane są zapisywane do systemu bazodanowego.
3. Administrator – jest to użytkownik o najwyższych uprawnieniach. Administratorem projektu może być również użytkownik, który został dodany do projektu z wybraną rolą administratora. W taki sposób w przypadku zmiany zespołu, nie musi być dodawany nowy projekt, a w już istniejącym projekcie główną rolę będzie miał użytkownik z daną rolą.

3.3 Diagramy sekwencji

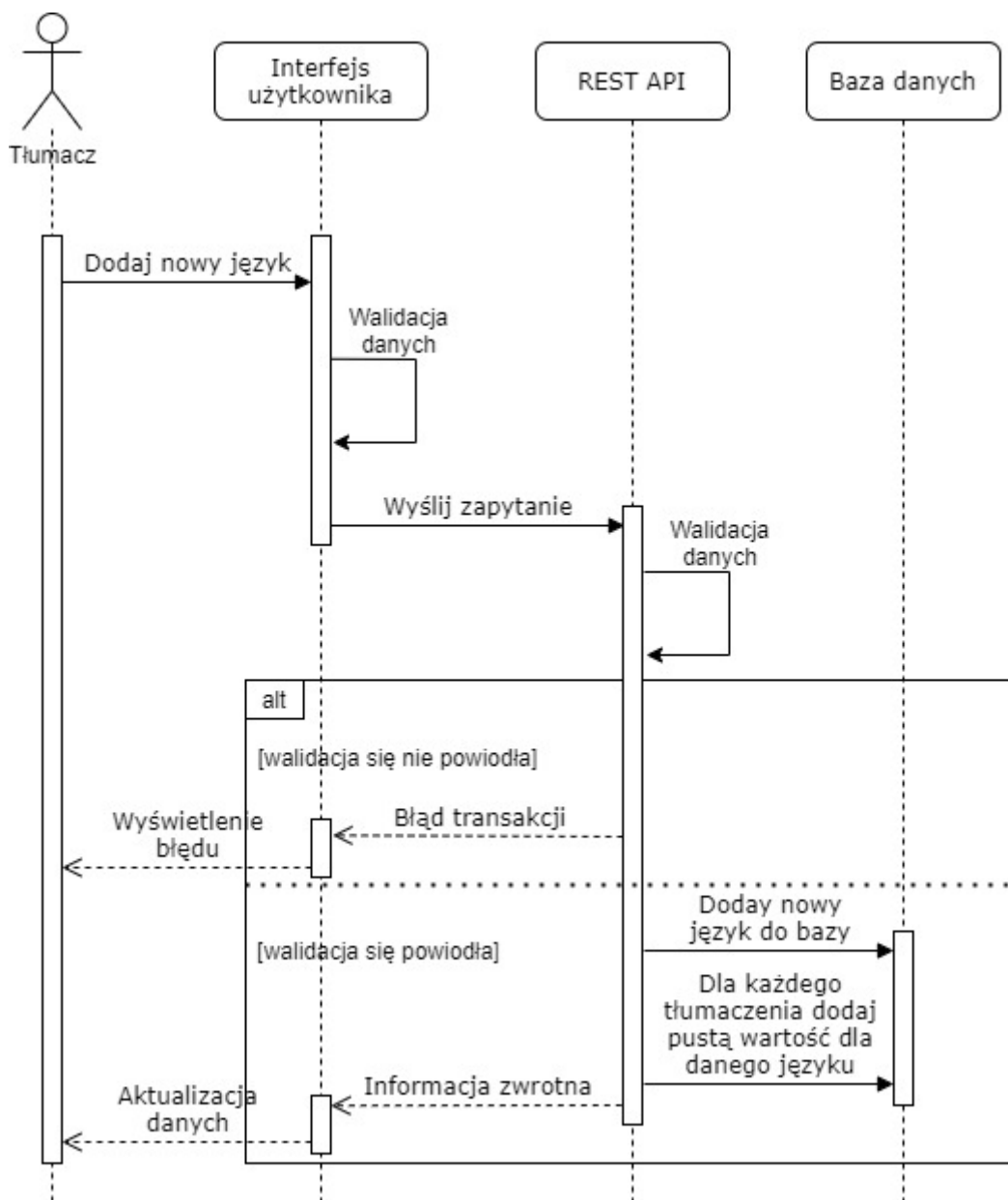
Po zaprojektowaniu i dokładnej analizie wszystkich przypadków użycia oraz wymagań do projektu, były analizowane poszczególne procesy pod kątem przepływu danych oraz interakcji pomiędzy systemami. W celu dokładnej analizy tych procesów zostały zaprojektowane diagramy sekwencji dla dwóch najważniejszych procesów.



Rys. 10 Diagram sekwencji: wprowadzanie zmian tłumaczeń
Źródło: opracowanie własne

Pierwszy proces, który jest najważniejszym procesem w danym projekcie, jest to wprowadzanie zmian tłumaczeń przy użyciu interfejsu użytkownika danego systemu. W danym procesie występuje tłumacz,

który wprowadza i zapisuje zmiany zapisuje do systemu przy pomocy funkcjonalności, które są dostarczane poprzez interfejs użytkownika. Po zapisaniu zmian poprzez odpowiedni przycisk aplikacja po stronie klienta wysyła dane do REST API. W tym momencie dokonywane są walidacje danych, sprawdzanie informacji o użytkowniku, sprawdzanie, czy jest użytkownik uprawniony do dokonywania takiego rodzaju czynności w ramach danego projektu. W przypadku gdzie wszystkie te sprawdzania się powiedą, zmiany tłumaczeń są zapisywane do bazy danych. Jednocześnie dodawany jest nowy rekord do historii zmian. Po zapisaniu wszystkich zmian, użytkownik zostaje poinformowany o tym, że dane zostały zaktualizowane a dane, które są widoczne dla użytkownika w tej chwili zostaną zaktualizowane. Na Rys. 10 jest podana wizualizacja danego procesu w postaci diagram sekwencji.



Rys. 11 Diagram sekwencji: przechowywanie zmian tłumaczeń
Źródło: opracowanie własne

Drugim ważnym procesem w danym systemie jest to dodawanie nowego języku. Nowy język może być dodawany przez użytkowników, którzy mają co najmniej rolę tłumacza. Po dodawaniu nowego języku po stronie klienta poprzez interfejs użytkownika, wysyłane są dane do REST API, gdzie są sprawdzane uprawnienia użytkownika oraz jest dokonywana walidacja danych. Podczas walidacji danych niezbędne jest sprawdzenie, czy język o takiej samej nazwie w danym projekcie. Jeżeli walidacja się powiedła,

dodawany jest nowy rekord do tabeli „Languages” oraz do tabeli „LanguageLangs”, gdzie są przechowywane przetłumaczone wartości dla odpowiedniego języka (Rys. 11).

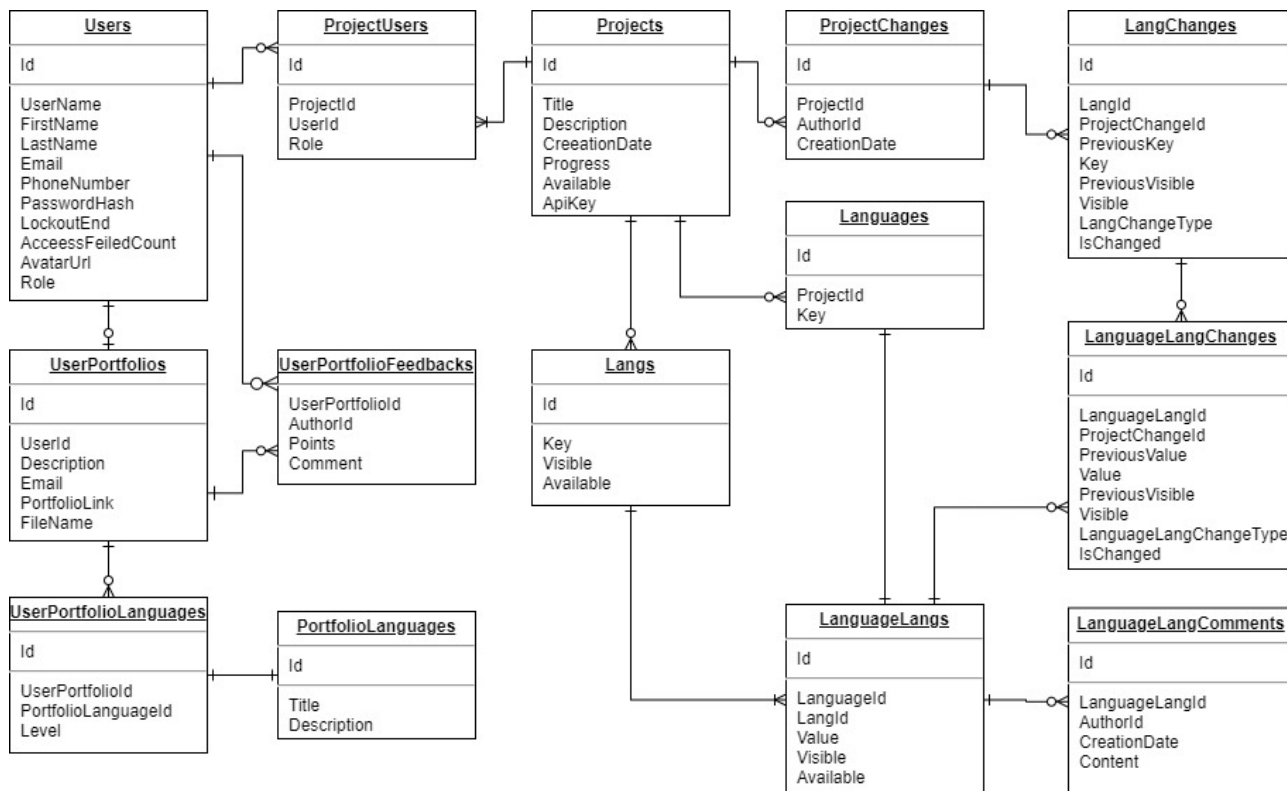
Oba procesy po zakończeniu działania zwracają informację zwrotną użytkownikowi systemu oraz są obsługiwane wyjątki w aplikacji.

3.4 Projekt bazy danych

Kolejnym etapem projektowania systemu jest to projektowanie struktury bazy danych (Rys. 12). Po analizie wszystkich wymagań zarówno funkcjonalnych jak i niefunkcjonalnych, została zaprojektowana struktura bazy danych, która będzie niezbędna do osiągnięcia postawionego celu.

Podczas projektowania bazy danych zostały uwzględnione wymagania zarówno funkcjonalne jak i niefunkcjonalne. Zostało zaprojektowano 14 tablic bazy z odpowiednimi relacjami pomiędzy nimi o następującej strukturze:

1. Tabela „Users”. W tej tabeli są wszystkie dane o użytkownikach systemu. Kluczem w tej tabeli jest Id, które będzie reprezentowany przez ciąg symboli tak zwany GUID (*ang. Global unique identifier*). Pole PasswordHash będzie zawierało hasło użytkownika w postaci zaszyfrowanej (pojęcie szyfrowania danych zostało opisano w [WWW-11, 2020]). Oprócz tego ze względu na bezpieczeństwo w systemie dla każdego użytkownika będzie przechowywana liczba prób zalogowania się podczas ostatniego logowania się.
2. Tabela „UserPortfolios”. Każdy użytkownik może dodać swoje portfolio w celu oferowania usług tłumacza.
3. Tabela „PortfolioLanguages”. Użytkownicy dodając swoje portfolio mogą wybrać odpowiednie języki, które są przechowywane w danej tabeli.
4. Tabela „UserPortfolioLanguages” jest to tabela, która łączy tabele „UserPortfolios” oraz „PortfolioLanguages”, ponieważ między nimi występuje relacja wiele do wielu.
5. Tabela „UserPortfolioFeedbacks” jest to tabela przechowująca wszystkie informacje zwrotne od innych użytkowników o danym tłumaczu.
6. Tabela „Projects”. W tej tabeli są przechowywane informacje o projektach.
7. Tabela „ProjectUsers” reprezentuje jako tabela łącząca projekty z użytkownikami. Relacja pomiędzy tabelą „Users” a tabelą „Projects” jest również relacją wiele do wielu.
8. Tabela „Languages”. W tabeli „Languages” są przechowywane języki, które zostały dodane w danym projekcie.
9. Tabela „Langs” przechowuje wszystkie rekordy, które są dodane w celu przetłumaczenia. Tabela ta reprezentuje wyłącznie klucze do danych rekordów, a wartości są przechowywane w tabeli „LanguageLangs”.
10. Tabela „LanguagesLangs” zawiera wszystkie wartości tłumaczeń dla poszczególnych rekordów z tabeli „Langs” oraz odpowiedniego języku (tabela „Languages”).
11. Tabela „ProjectChanges” przechowuje historię wraz z informacją o autorach zmian tłumaczeń w ramach konkretnego projektu.
12. Tabela „LangChanges” odpowiada za przechowywanie wszystkich zmian tłumaczeń z informacją o stanie wcześniejszym danego tłumaczenia.
13. Tabela „LanguageLangChanges” przechowuje wszystkie zmiany wartości z informacją o poprzednich wartościach tłumaczeń dla konkretnego języku.
14. Tabela „LanguageComments” przechowuje wszystkie komentarze dodane do rekordów z tabeli „LanguageLangs”.



Rys. 12 Diagram ERD tabel powiązanych z projektem
Źródło: Opracowanie własne

3.5 Projekt interfejsu użytkownika

Jednym z najbardziej popularnych podejść w procesie projektowania interfejsów użytkownika jest to projektowanie zorientowane na użytkownika. Proces taki ma dużo zalet, ale jest również bardzo czasochłonny, ponieważ wymaga długiej analizy, planowania i testowania.

W celu realizacji dobrej jakości aplikacji, z której będą chcieli korzystać użytkownicy końcowi, zostały użyte niektóre z etapów procesów HCD. System został zaprojektowany z uwzględnieniem dobrych praktyk projektowania UI oraz UX.

Projektowanie zorientowane na użytkownika jest strategią i procesem projektowania interfejsów, w którym potrzeby użytkowników końcowych są najważniejszym kryterium w procesie projektowania. Takie podejście jest procesem wieloetapowym, który wymaga od projektantów nie tylko analizowania i przewidywania sposobu korzystania z produktu przez użytkowników, ale także sprawdzania poprawności założeń dotyczących zachowania użytkownika w rzeczywistych testach z potencjalnymi użytkownikami. Takie testowanie jest bardzo ważnym etapem w tym procesie, ponieważ projektant nie jest zawsze w stanie zrozumieć, w jaki sposób użytkownicy będą posługiwać się danym narzędziem [Lowdermilk, 2013].

Ze względu na to, że na danym etapie wszystkie wymagania zostały już przeanalizowane, proces ten został pominięty w procesie projektowania interfejsu. Najwięcej czasu zostało przydzielone na proces empatyzacji, generowania pomysłów oraz prototypowania.

Proces empatyzacji polega na tym, że projektant analizuje dany problem pod względem użytkownika końcowego, wszystkie możliwe problemy, które mogą występować podczas obsługi systemu poprzez użytkowników końcowych. Uwzględniana w tym momencie jest najbardziej grupa docelowa, ponieważ projekt jest skierowany na osób, które mają do czynienia z wielojęzycznymi systemami informatycznymi oraz ich tłumaczeniami.

Po analizie i uzyskaniu wniosków po danym etapie, zostały wygenerowane pomysły rozwiązania danego problemu w postaci makiet interfejsów użytkownika (Rys. 13).



Rys. 13 Podstawowy makiet interfejsu użytkownika
Źródło: opracowanie własne

Prototypowanie to proces tworzenia niskiej lub wysokiej jakości makiet projektu aplikacji, które mogą być wykorzystywane w celu przetestowania użyteczności. Prototypowanie wspomaga, w wizualizacji koncepcji związanej z aplikacją, która zostanie wytworzona w przyszłości. W taki sposób można generować pomysły i je przechowywać w takiej postaci. Pominięcie etapu prototypowania może skutkować popełnieniem błędu na etapie implementacji, co prowadzi do większych wydatków i wymaga więcej czasu naprawy.



Rys. 14 Pierwszy prototyp interfejsu użytkownika
Źródło: opracowanie własne

Na tym etapie zostały zintegrowane wszystkie pomysły i został zaprojektowany pierwszy prototyp (Rys. 14). Po analizie danego prototypu oraz testów na niezależnych użytkownikach, były wykryte następujące błędy związane z użytecznością aplikacji:

1. Nieczytelność treści. Ze względu na wybraną kolorystykę, tekst w wielu miejscach jest nieczytelny i utrudnia to prace w systemie.
2. Problem z czytelnością informacji o użytkownikach. Po analizie danego problemu została przeniesiona sekcja z informacji o użytkowniku w bardziej widoczne miejsce – na dół sekcji bocznej.
3. Pozycjonowanie przycisku do wylogowania. Ze względu na to, że przycisk do wylogowania jest taki blisko innych elementów nawigacji, użytkownik błędnie wciskając na ten przycisk, musi przejść proces autoryzacji od nowa.

Na podstawie wyników testów, został zaprojektowany nowy układ interfejsu użytkownika (Rys. 15) oraz został wybrany nowy schemat kolorów w celu polepszenia widoczności tekstu oraz lepszego doświadczenia użytkownika (Rys. 16).



Rys. 15 Układ interfejsu użytkownika po analizie testów na użytkownikach
Źródło: opracowanie własne

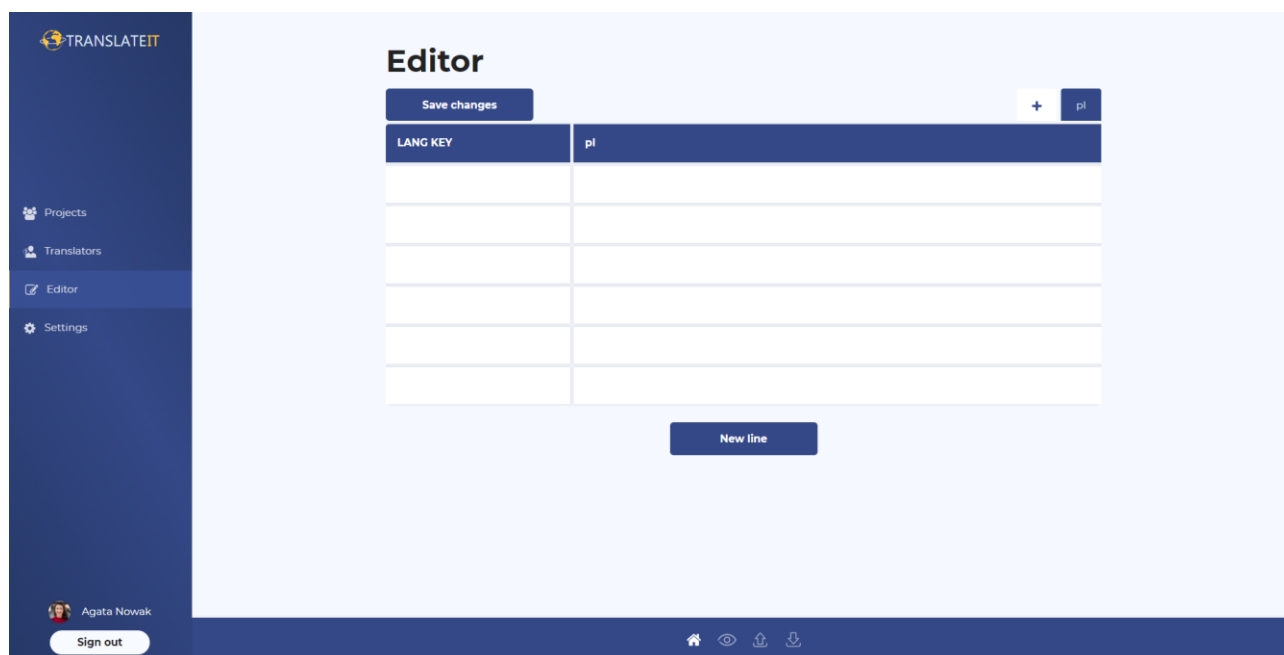


Rys. 16. Kolorystyka aplikacji
Źródło: opracowanie własne

Podczas projektowania wersji końcowej aplikacji, zostały również dodane nowe elementy, które są niezbędne do uzyskania aplikacji, która spełnia wszystkie wymagania funkcjonalne. Zostało również zaprojektowane logo aplikacji (Rys. 17). Na podstawie danego prototypu został zaimplementowany końcowy interfejs aplikacji (Rys. 18).



Rys. 17 Logo aplikacji
Źródło: opracowanie własne



Rys. 18 Drugi prototyp interfejsu użytkownika
Źródło: opracowanie własne

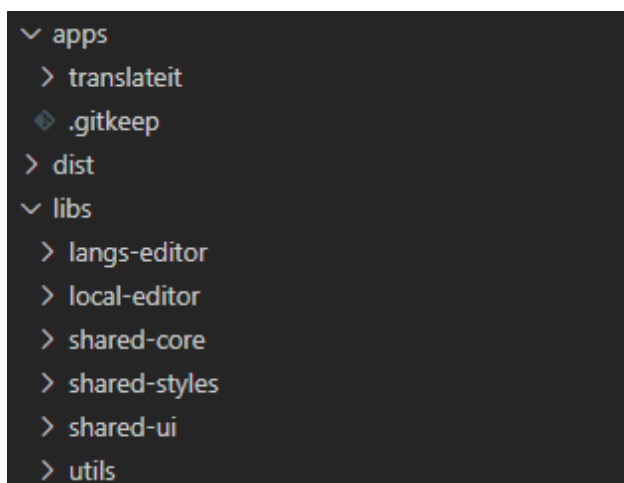
Po dokonaniu niezbędnej analizy oraz zaprojektowaniu wszystkich elementów oraz procesów danego systemu, został rozpoczęty proces implementacji. Ze względu na to, że wszystkie procesy zostały dokładnie przeanalizowane, podczas implementacji było popełniono znacznie mniej błędów.

4 Implementacja

W tym rozdziale zostanie opisany proces implementacji systemu. Ze względu na to, że system jest podzielony na 3 części, implementacja każdej z nich jest osobnym procesem, który będzie omówiony poniżej.

4.1 Implementacja aplikacji internetowej

Aplikacja kliencka została podzielona na projekt aplikacji głównej oraz projekty bibliotek, które również zostały zaprojektowane w ramach danego rozwiązania. Do bibliotek dodane są moduły, które mogą być użyte w całej aplikacji głównej oraz w przyszłości w innych rozwiązaniach oraz do dalszego rozwoju aplikacji (Rys. 19). Taka struktura jest zgodna z zasadami SOLID. Wszystkie elementy są odseparowane i mogą być wykorzystywane w potrzebnych modułach czy komponentach bez konieczności modyfikacji kodu źródłowego (zasada otwarte-zamknięte), a każdy komponent jest odpowiedzialny za konkretną funkcjonalność (zasada jednej odpowiedzialności).



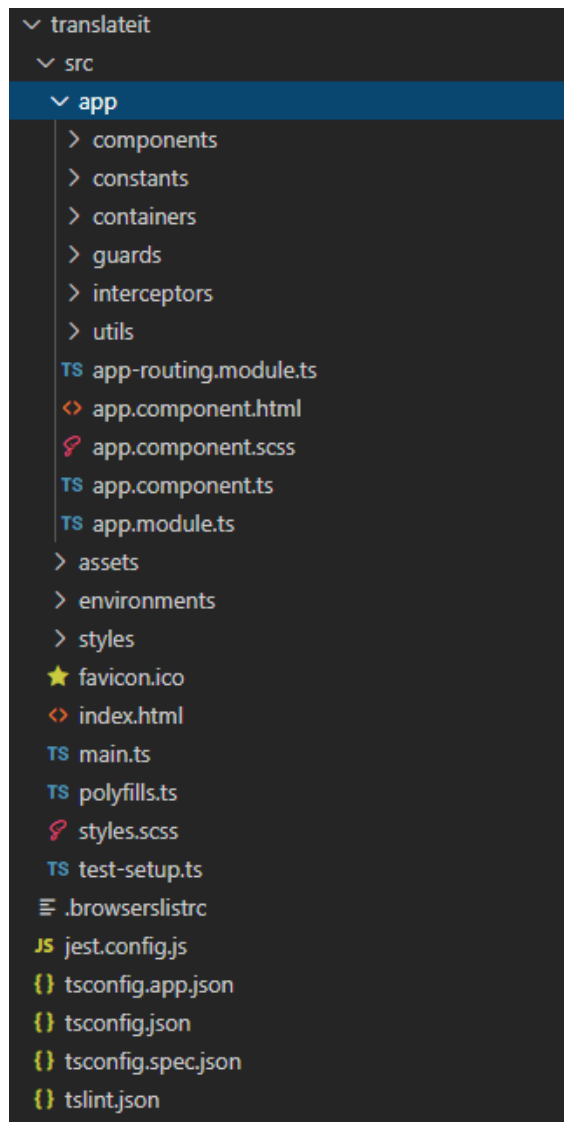
Rys. 19 Ogólna struktura aplikacji klienckiej
Źródło: opracowanie własne

W module „shared-ui” są przechowywane komponenty interfejsu użytkownika, które upraszczają proces wytwarzania aplikacji, ponieważ mogą być używane wielokrotnie, a w przypadku konieczności zmiany wyglądu lub logiki tych elementów nie musimy powtarzać te zmiany w każdym miejscu, w którym używamy dany komponent.

Moduł „langs-editor” jest to moduł, który zawiera wszystkie elementy edytora tłumaczeń oraz odpowiada za ich integrację. Moduł ten przewiduje możliwość rozszerzenia aplikacji. Dzięki temu, że Angular udostępnia narzędzia do DI (*ang. dependency injection* - *wstrzykiwanie zależności*) w modułach, możemy rozszerzać te moduły podając własny serwis do obsługi procesów związanych z przechowywaniem i pobieraniem tłumaczeń. W przypadku danego rozwiązania zostały zaimplementowane 3 rodzaje wyżej wspomnianych serwisów:

1. Serwis do pobierania i przechowywania tłumaczeń w pamięci przeglądarki – LocalEditorService.
2. Serwis do pobierania i przechowywania tłumaczeń należących do konkretnego projektu z REST API po stronie serwera – ProjectEditorService.
3. Serwis do pobierania i przechowywania tłumaczeń należących do konkretnego rekordu z historii zmian z REST API po stronie serwera – ProjectHistoryEditorService.

Serwisy te zostały użyte odpowiednio na stronach edytora tłumaczeń lokalnego, edytora tłumaczeń projektu oraz strony z podglądem zmian w projekcie.



Rys. 20 Struktura głównego modułu aplikacji
Źródło: opracowanie własne

Implementacja aplikacji głównej została przeniesiona do folderu „apps/transateit” (Rys. 20). W głównym folderze aplikacji są przechowywane pliki konfiguracyjne oraz podkatalogi takie jak „components”, „containers”, „interceptors”, „guards”, „utils”, „constants”, „styles”, „environments” oraz „assets”. W folderze „components” są przechowywane mniejsze elementy interfejsu użytkownika, które mogą być używane w różnych widokach aplikacji. Widoki, które są połączeniem komponentów oraz elementów, które są zaimplementowane w wyżej wspomnianych bibliotekach, są przechowywane w folderze „containers”. Każdy z tych widoków są odrębnymi modułami, które są importowane do modułu głównego aplikacji przy użyciu narzędzia do tak zwanego leniwego ładowania (*ang. lazy loading*).

Folder „components” zawiera nie duże fragmenty widoków, które są używane w kontenerach aplikacji. Do takich komponentów należą na przykład komponent fragmentu bocznego z nawigacją na stronie profilu użytkownika. Komponenty te nie muszą być wynoszone do globalnego folderu, ponieważ nie są używane wielokrotnie w różnych miejscach i ewentualnie różnych aplikacjach.

W folderze „containers” są przechowywane duże elementy widoków, które dzielą całą aplikację na podstrony i są od siebie niezależne. Większość z elementów danego folderu reprezentują osobny widok aplikacji.

Niektóre kontenery definiują ścieżki do poszczególnych widoków aplikacji. Przykładem takiego modułu jest „ProjectPageModule”. Moduł ten odpowiada za kierowanie użytkownika do widoków związanych

z projektami. Jednym z takich widoków jest to widok z formularzem do dodawania nowego projektu, kolejnym jest lista wszystkich projektów oraz widok z szczegółami oraz edytorem danego projektu. Oprócz tego moduł ten prowadzi również do widoków z historią zmian, statystyką projektu oraz formularza do edytowania danych projektu (Rys. 21).

```
const routes: Routes = [
  {
    path: '',
    component: ProjectPageComponent,
    children: [
      {
        path: '',
        pathMatch: 'full',
        loadChildren: () =>
          import('./components/project-list').then((m) => m.ProjectListModule),
      },
      {
        path: 'new',
        pathMatch: 'full',
        loadChildren: () =>
          import('./components/project-form').then((m) => m.ProjectFormModule),
      },
      {
        path: ':projectId',
        loadChildren: () => import('./components/project-page/project-page.module').then(m => m.ProjectPageModule)
      },
      {
        path: ':projectId/history',
        loadChildren: () => import('./components/project-history/project-history.module').then(m => m.ProjectHistoryModule)
      },
      {
        path: ':projectId/statistics',
        loadChildren: () => import('./components/project-statistics-page/project-statistics-page.module').then(m => m.ProjectStatisticsPageModule)
      }
    ]
  },
];
```

Rys. 21 Ścieżki do widoków modułu „ProjectPage”

Źródło: opracowanie własne

W celu zarządzania uprawnieniami użytkownika, są dodane klasy tak zwani strażnicy uprawnień (*ang. guards*), które implementują interfejs „CanActivate”. Są to wbudowane interfejsy Angular, które są obsługiwane przez router na podstawie stanu aplikacji oraz podanej ścieżki. Najważniejsze tego rodzaju klasy, które zostały dodane są to „AuthGuard” oraz „UnAuthGuard”, które odpowiednio służą do kontroli uprawnień do widoków prywatnych oraz widoków dostępnych wyłącznie użytkownikom niewie-ryzelnionym.

Folder „Interceptors” przechowuje klasy, które służą do przechwytywania zapytań REST API. W przypadku danego projektu zostały zaimplementowane klasy „TokenInterceptor” oraz „ResponseInterceptor”. Pierwsza z klas służy do przekazywania identyfikatora JWT w nagłówkach przy każdym zapytaniu wysłanym do REST API. Drugi interceptor służy do obsługi zwróconych danych w odpowiedzi. W tej klasie są również obsługiwane błędy. W przypadku błędu 401 wysyłane jest zapytanie o nowy identyfikator dla użytkownika. Jeżeli zapytanie się nie powiodło, użytkownik jest przekierowywany do strony logowania.

Taka struktura aplikacji zapewnia, że większość elementów może być używana wielokrotnie bez konieczności zmiany kodu źródłowego. Każdy element aplikacji w łatwy sposób może być odizolowany, jest łatwy do błędów oraz debugowanie.

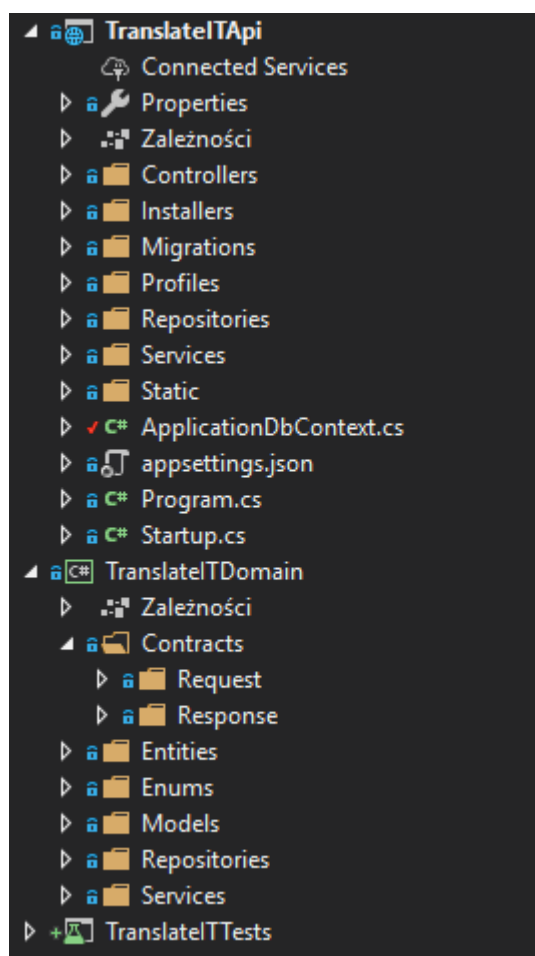
Podczas implementacji aplikacji klienckiej wystąpił problem związany z wydajnością. Aplikacja działała wolno w miejscach, gdzie jest wizualizacja dużej ilości danych. Ze względu na to została zmieniona strategia śledzenia zmian, która udostępnia Angular.

Strategia podstawowa, która jest używana w Angular dokonuje przerysowywania całej strony po każdym zdarzeniu takim jak kliknięcie, najeżdżenie myszy, wprowadzaniu wartości do pól tekstowych itd. W przypadku małych widoków z małą liczbą elementów taka strategia może być używana, ale w przypadku dużej liczby dynamicznych elementów, gdzie dokonywano dużo obliczeń oraz jest dużo interakcji ze strony użytkownika, strategia taka nie jest dobrym rozwiązaniem.

Po zmianie strategii śledzenia zmian aplikacja jest bardziej płynna, co ma duży wpływ na doświadczenie użytkownika.

4.2 Implementacja REST API

Aplikacja po stronie serwera została zaimplementowana w postaci REST API przy użyciu technologii .NET Core 3.1. Główne rozwiązanie zostało podzielone na 3 projekty (Rys. 22).



Rys. 22 Struktura plików rozwiązania REST API
Źródło: opracowanie własne

Projekt „TranslateITDomain” zawiera wszystkie definicje modeli, które występują w całej aplikacji.

W folderze „Contracts” tego projektu zostały dodane modele do obsługi komunikacji z API a aplikacją kliencką. Modele te zawierają tylko niezbędne elementy modeli rzeczywistych. Dane, które są zwracane użytkownikom są przechowywane w podkatalogu „Responses”, a modele, które API przyjmuje jako dane w odpowiednich punktach końcowych, są dodane do podkatalogu „Requests”.

„Entities” są to rzeczywiste modele, które służą do zapisu oraz odczytu w bazie danych.

Projekt „TranslateITTests” jest to projekt dla testów jednostkowych. W ramach danego projektu są dokonywane testy jednostkowe najważniejszych funkcjonalności aplikacji.

Główny projekt aplikacji oraz projekt uruchomieniowy jest to projekt „TranslateITApi”. Klasa Program jest klasą wstępną, która uruchamia projekt. W pliku Startup.cs danego projektu są dodane wszystkie niezbędne serwisy, które będą wykorzystywane w aplikacji przy pomocy wstrzykiwania zależności oraz zostały dodane ustawienia projektu w zależności od trybu uruchomienia aplikacji. Plik appsettings.json służy do przechowywania wartości konfiguracyjnych aplikacji, które nie mogą być dostępne publicznie.

ApplicationDbContext.cs jest to klasa, która dziedziczy od klasy „DbContext” Entity Framework Core i zawiera deklaracje wszystkich tabel, które powinny występować w bazie oraz zależności pomiędzy nimi.

W folderze „Static” są przechowywane pliki statyczne takie jak różnego rodzaju grafiki. Folder „Services” oraz „Repositories” są przechowywane serwisy, które są dostarczane do kontrolerów (folder „Controllers”) poprzez wstrzykiwanie zależności, które jest konfigurowane w sposób automatyczny w plikach z folderu „Installers”.

Folder „Migrations” służy do przechowywania wszystkich migracji bazy danych z możliwością powrotu do stanu bazy w odpowiednim momencie czasu. W trakcie implementacji aplikacji było wygenerowano 12 migracji bazy.

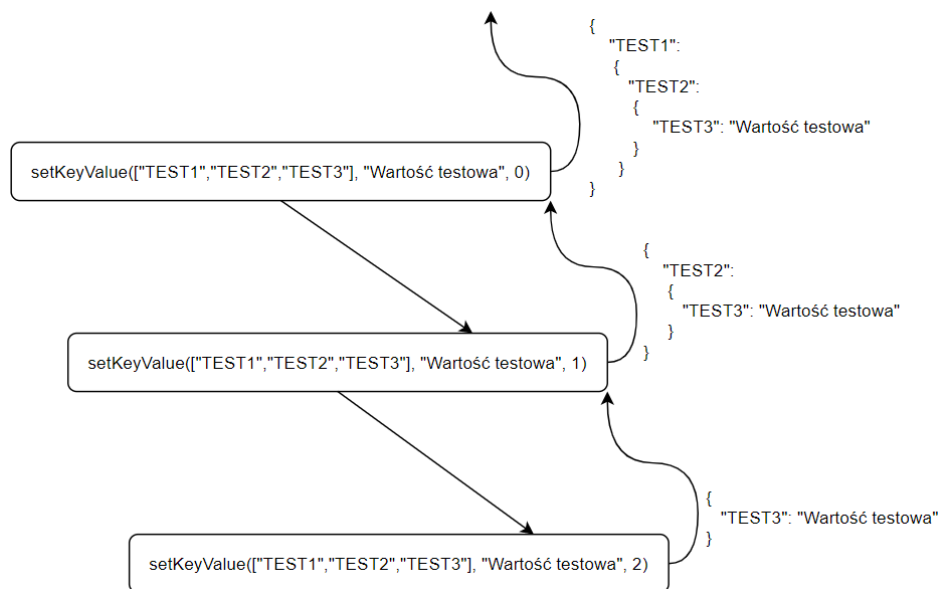
4.3 Implementacja serwisów do przechowywania tłumaczeń

Dodatkowo został zaimplementowany serwis do obsługi plików z tłumaczeniami w języku TypeScript przy użyciu platformy Node.js. Serwis ten odpowiada za generowanie plików z tłumaczeniami z rozszerzeniem .json, .excel oraz plików ze zmiennymi PHP.

Serwis ten dostarcza dwie główne funkcjonalności:

1. Generowanie plików z tłumaczeniami
2. Pobieranie obiektów tłumaczeń z plików z tłumaczeniami o rozszerzeniach .json oraz .excel.

Podczas generowania plików z rozszerzeniem .json są dostępne do wyboru opcje, czy plik powinien być minimalizowany, czy zagnieżdżony. W celu dostarczenia takiej funkcjonalności został zaimplementowany algorytm rekurencyjny, który dzieli klucz na części na podstawie symbolu do oddzielania – kropki. W każdym kroku algorytmu sprawdzana jest wartość parametru obiektu JSON. Jeżeli podany parametr obiektu nie była jeszcze dodana do obiektu końcowego, dodawany jest nowy parametr z wartością pustą. Jeżeli atrybut jest ostatni w kolejności, jest on dodawany do obiektu końcowego, a jego wartość jest to wartość, która była przekazana na wejściu (Rys. 23).



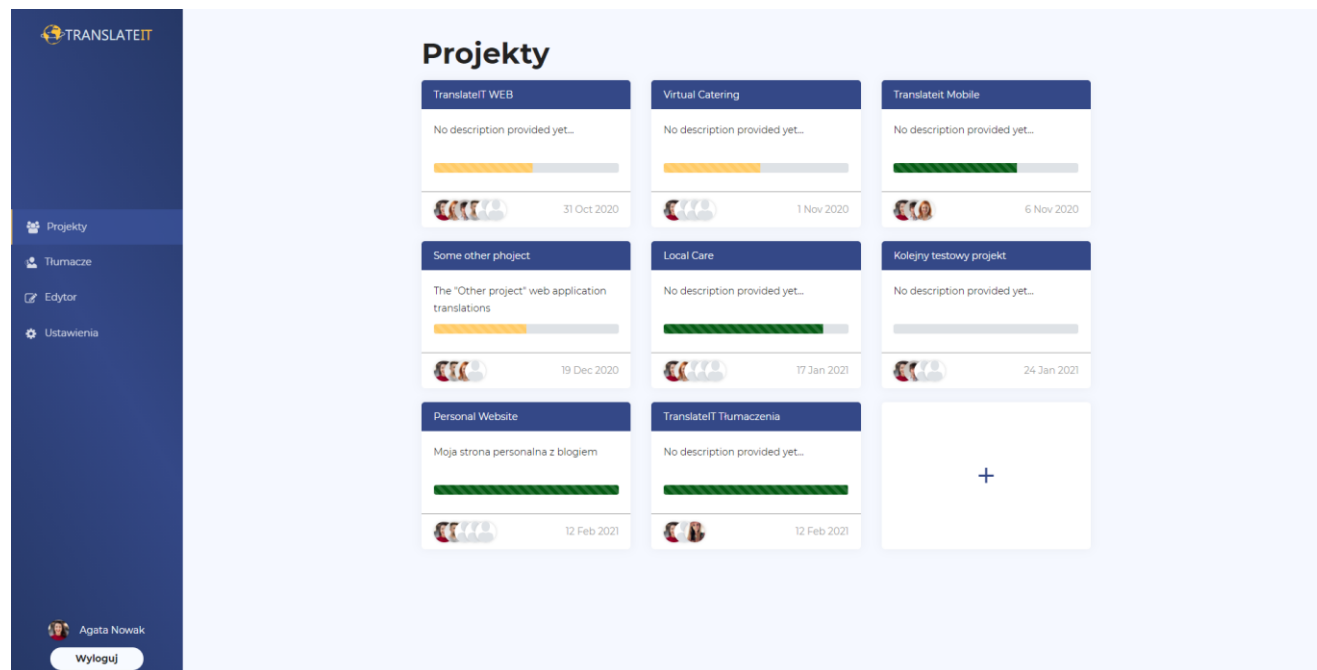
Rys. 23 Przebieg funkcji rekurencyjnej do generowania JSON obiektów z zagnieżdżeniami

Źródło: opracowanie własne

4.4 Opis działania aplikacji

Główny widok aplikacji jest to strona główna, która zawiera informację o systemie, przycisk do zmiany języku interfejsu oraz link do strony logowania oraz profilu użytkownika.

Po zalogowaniu do aplikacji użytkownik zostanie przekierowany do strony z listą aktualnych projektów, w których on uczestniczy (Rys. 24). Każdy element z tej listy zawiera krótką informację o projekcie taką jak tytuł, opis, indyktor progresu tłumaczeń oraz uczestników. Każdy z elementów listy jest jednocześnie odniesieniem do strony z dokładną informacją o projekcie oraz edytorem z tłumaczeniami. Lista na stronie tej zawiera również przycisk do przejścia do formularza nowego projektu.



Rys. 24 Widok z listą projektów
Źródło: opracowanie własne

Po lewej stronie danego widoku widać basek boczny z odniesieniami do odpowiednich podstron takich jak „Lista projektów”, „Tłumaczy”, „Edytor”, „Ustawienia”, informacje o zalogowanym użytkowniku oraz przycisk do wylogowania.

Po przejściu do widoku projektu, w którym już są dodane tłumaczenia, dostępny jest widok edytora ze wszystkimi niezbędnymi narzędziami do pracy z tłumaczeniami.

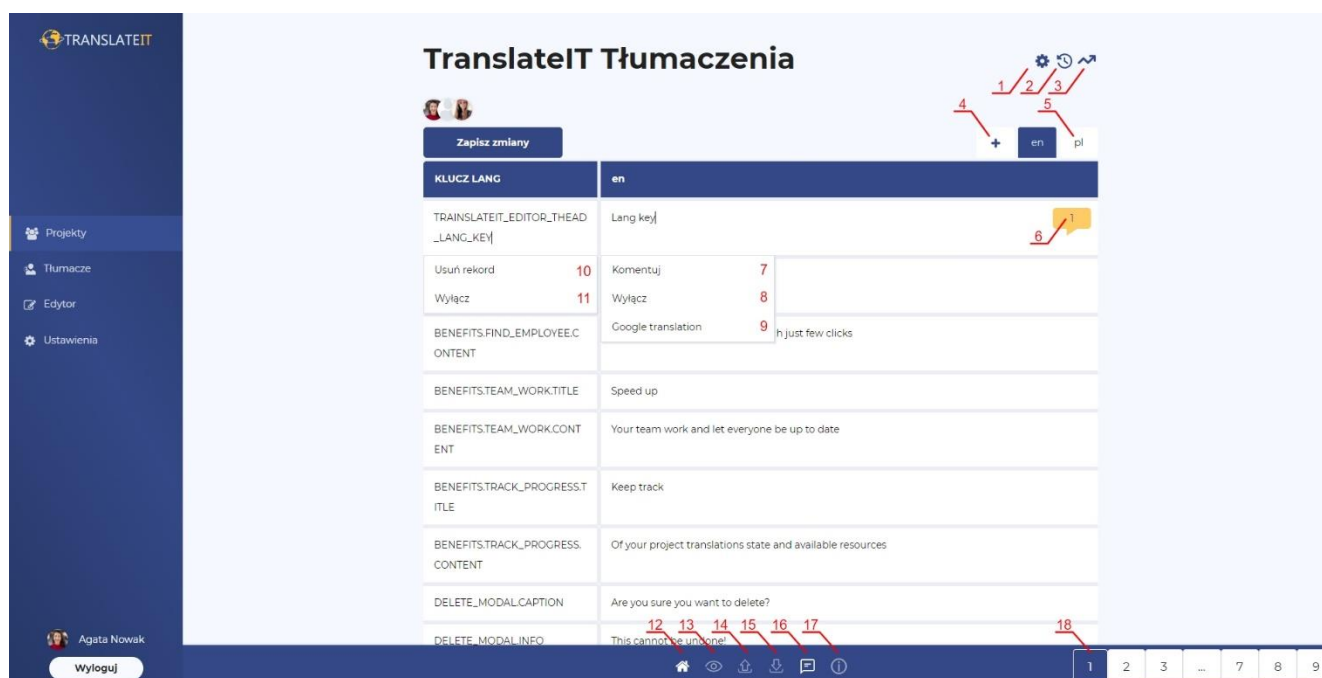
Na górze tego widoku jest informacja o projekcie. Wyświetlany jest tytuł, opis (opcjonalnie) oraz uczestnicy projektu. Po prawej stronie jest nawigacja do ustawień projektu, historii zmian tłumaczeń oraz statystyki projektu (Rys. 25, 1-3).

Pod sekcją nawigacji znajdują się przyciski do zmiany języku tłumaczeń, bezpośrednio w każdej z komórek edytora może być otwierane menu kontekstowe, które zawiera dodatkową funkcjonalność taką jak dodawanie komentarza, włączanie oraz wyłączanie całego wiersza lub wartości dla konkretnego języku, przejście do strony Google Translator z podanymi parametrami do przetłumaczenia oraz opcja do usuwania rekordu (Rys. 25, 7 – 11). Po prawej stronie dostępny jest przycisk do przeglądania komentarzy do danego tłumaczenia (Rys. 25, 6).

Po dokonaniu zmian użytkownik musi skorzystać z przycisku do zapisywania zmian, żeby dane nie zostały stracone.

Na dole widoku są dostępne przyciski do podglądu wersji finalnej tłumaczeń, do załadowania plików z tłumaczeniami, do pobierania plików z tłumaczeniami oraz przycisk do włączania i wyłączania komentarzy oraz dodatkowej informacji do tłumaczeń (Rys. 25, 12 – 17) oraz przyciski do nawigacji po podstronach tłumaczeń (Rys. 25, 18)

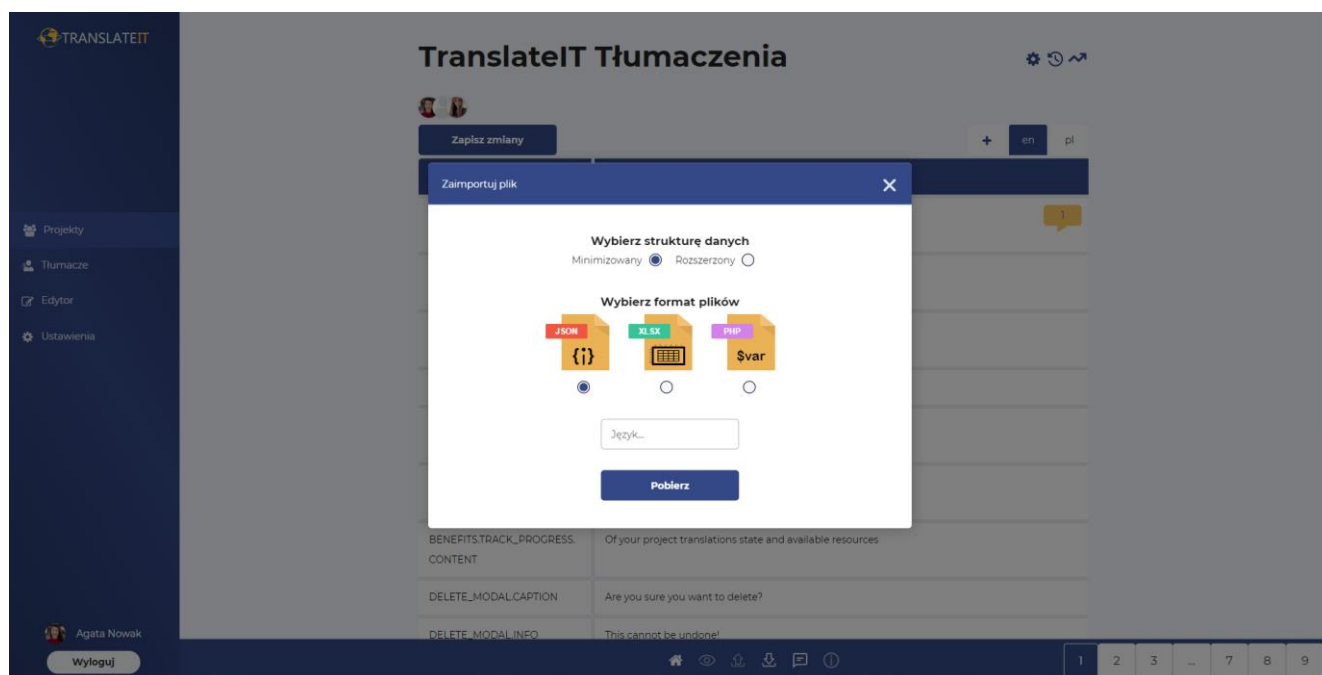
W przypadku wciśnięcia przycisku pobierania plików z tłumaczeniami, pokazywane jest okienko z opcjami formatu danego pliku. Użytkownik może wygenerować plik Excel, plik JSON lub skompresowany lub zagnieżdżony oraz plik ze zmiennymi w języku PHP (Rys. 26).



Rys. 25 Widok projektu z tłumaczeniami
Źródło: opracowanie własne

Po wyborze odpowiedniego formatu użytkownik może pobrać plik i wykorzystywać w celu zarówno podglądu jak i na produkcji w aplikacji. Pobrane pliki Excel oraz JSON mogą być również importowane do systemu w celu dodania tłumaczeń z już istniejących plików.

Po zakończeniu danego procesu okienko z opcjami wyboru pliku do pobierania, jest zamykane. W przypadku wystąpienia problemu na serwerze lub podczas próby łączenia się z serwerem, użytkownik zostaje poinformowany.



Rys. 26 Widok generowania plików z tłumaczeniami
Źródło: opracowanie własne

Oprócz wyżej wymienionych funkcjonalności są dostępne widoki aplikacji takie jak lista tłumaczy, edytor lokalny oraz ustawienia aplikacji.

W wyniku implementacji danego systemu zostały wytworzone trzy główne moduły:

1. Frontend w postaci SPA, który umożliwia korzystanie ze wszystkich funkcjonalności, które oferuje system.
2. Backend w postaci REST API.
3. Serwis do zarządzania plikami z tłumaczeniami w postaci REST API.





Po zakończeniu procesu implementacji projekt był gotowy do kolejnego etapu – testów systemowych.

5 Testowanie

W trakcie implementacji oraz po zakończeniu implementacji poszczególnych modułów aplikacji, były przeprowadzone testy automatyczne oraz manualne.

Funkcjonalność serwisu do przetwarzania plików z tłumaczeniami została pokryta testami jednostkowymi. Udało się osiągnąć ponad 97% pokrycia głównych serwisów aplikacji, co pomogło w wyeliminowaniu dużej ilości błędów logicznych oraz zapewniło kompatybilność wstecz podczas dalszej implementacji aplikacji (Rys. 27).

Testy jednostkowe do danego serwisu zostały napisane przy użyciu biblioteki do uruchomienia testów w języku JavaScript Jest.

File ▲		Statements ▾	Branches ▾	Functions ▾	Lines ▾
enums		100%	16/16	100%	6/6
mocks		100%	5/5	100%	0/0
models		100%	10/10	100%	8/8
services		97.14%	68/70	91.43%	32/35

Rys. 27 Testy jednostkowe serwisu do zarządzania plikami z tłumaczeniami
Źródło: opracowanie własne

W ramach rozwiązania REST API zostały napisane testy jednostkowe przy użyciu biblioteki XUnit. Testami zostały pokryte główne ścieżki występujące w „ProjectsRepository” oraz „UsersRepository”. Podczas testów jednostkowych został wykryty błąd logiczny w metodzie do wyliczania postępu projektów. Po poprawieniu danego błędu i ponownym uruchomieniu testów jednostkowych, dany test, jak i wszystkie inne testy się powiodły (Rys. 28).

▲ ✓ TranslateITests (15)	4.1 s
▲ ✓ TranslateITests (15)	4.1 s
▲ ✓ ProjectsRepositoryTest (6)	2.2 s
✓ AddProjectAsync_ShouldHave1...	1.6 s
✓ AddProjectUsers_ShouldHavePr...	155 ms
✓ GetAll_ShouldReturnProjectsFro...	185 ms
✓ GetById_ShouldReturnProjectFro...	78 ms
✓ UpdateProjectProgress_Should...	154 ms
✓ UpdateUpdateAsync_ShouldHa...	17 ms
▲ ✓ UsersRepositoryTest (9)	1.9 s
✓ AddUserAsync_ShouldHave1Ap...	56 ms
✓ AddUserAsync_ShouldHave2Us...	1.6 s
✓ AddUserAsync_ShouldReturnTrue	2 ms
✓ GetUserAsync_ShouldReturnAA...	36 ms
✓ GetUsersAsync_ShouldReturnE...	2 ms
✓ GetUsersAsync_ShouldReturnFil...	110 ms
✓ GetUsersAsync_ShouldReturnLi...	5 ms
✓ UpdateUserAsync_ShouldRetur...	3 ms
✓ UpdateUserAsync_ShouldUpdat...	34 ms

Rys. 28 Wyniki testów jednostkowych REST API
Źródło: opracowanie własne

Po zakończeniu implementacji oraz po każdym cyklu dodawania nowej funkcjonalności były przeprowadzane testy „end to end” całego systemu. Dzięki testom jednostkowym oraz systemowym udało się wykryć większość błędów logicznych oraz zostały naprawione błędy związane z użytecznością systemu. W momencie testowania zostały wykryte problemy związane z wydajnością, które zostały naprawione i przetestowane ponownie w kolejnym cyklu. Dzięki temu system końcowy spełnia wymagania funkcjonalne oraz нефункционалне.

Zakończenie

W ramach pracy dyplomowej został zaprojektowany i zaimplementowany system do zarządzania tłumaczeniami, który spełnił wymagania zarówno funkcjonalne jak i нефunkcjonalne. Serwis rozwiązuje problem opisany w pierwszym rozdziale pracy. Takie rozwiązanie w dniu dzisiejszym ma duży potencjał na rynku, a dzięki strukturze może być rozwijany i rozszerzany.

W trakcie projektowania oraz implementacji rozwiązania było przeanalizowano wiele dostępnych na rynku narzędzi, zostały porównane różne technologie do wytwarzania aplikacji internetowych typu SPA oraz aplikacji w postaci REST API. Ze względu na to oraz na nietypowy problem, który był rozwiązywany, została wzbogacona wiedza nie tylko w zakresie wytwarzania oprogramowania ale również w zakresie tłumaczenia interfejsów użytkownika i wpływu danego procesu na dostępność oraz użyteczność dostarczanego systemu.

Oprócz nawyków technologicznych, które były wzbogacone podczas wykonywania pracy dyplomowej, ważnym zadaniem było również nauczenie się projektowania użytecznych oraz nowoczesnych interfejsów użytkownika.

Aplikacja, która została zaimplementowana, spełnia również wymagania dotyczące skalowalności aplikacji i możliwości dalszego rozwoju. Dzięki podziałowi systemu na osobne moduły, zapewniona jest niezależność od platformy oraz technologii. Ze względu na strukturę aplikacji internetowej, na podstawie kodu danego projektu może być w prosty sposób wytworzona aplikacja desktopowa z użyciem platformy Electron, która umożliwia wytwarzanie takich aplikacji na podstawie kodu HTML, JavaScript oraz CSS. W celu wytworzenia prostego i lekkiego narzędzi do obsługi tłumaczeń może być wykorzystany tylko moduł edytora, który może działać niezależnie od całej aplikacji.

Literatura

[Ben-Gan, 2012]

Itzik Ben-Gan, Microsoft SQL Server 2012 T-SQL Fundamentals
Pearson Education, 2012

[Brooks, Grow, Craig, Short, 2018]

Charles J. Brooks, Christopher Grow, Philip Craig, Donald Short, *Cybersecurity Essentials*
Canada 2018, s. 14 – 21.

[Camden, 2013]

Raymond Camden, *Client-Side Data Storage. Keeping It Local*
O'Reilly, 2013.

[Esposito - 2018]

Dino Esposito, *Programming ASP.NET Core*
Microsoft Press, 2018, s. 25 – 42.

[Freeman, 2018]

Adam Freeman, *Pro Entity Framework Core 2 for ASP.NET Core MVC*
Apress, 2018, s. 3 – 7.

[Fenton, 2017]

Steve Fenton, *Pro TypeScript: Application-Scale JavaScript Development*
Apress, 2017, s. 1 – 7.

[Herron, 2016]

David Herron, *Node.js Web Development Third Edition*
Packt Publishing, Birmingham 2016, s. 235

[Hofmann-Delbor, Bartnicka, 2017]

Agenor Hofmann-Delbor, Marta Bartnicka, *Programiści i tłumacze. Wprowadzenie do lokalizacji oprogramowania*
Helion, 2017, s. 39 – 65.

[Lowdermilk, 2013]

Travis Lowdermilk, *User-Centered Design*
O'Reilly, 2013.

[McQuillan, 2013]

Mike McQuillan, *Introducing SQL Server*
Apress, 2015, s.1 – 33.

[Murray, Coury i inni, 2018]

Nate Murray, Felipe Coury, Ari Lerner, Carlos Taborda, *Ng-book: The Complete Guide to Angular*
CreateSpace Independent Publishing Platform, 2018.

[Richardson, Ruby, 2011]

Leonard Richardson, Sam Ruby, *RESTful Web APIs*
O'Reilly, 2011.

[Skeet, 2013]
Jon Skeet, *C# in Depth*
Manning, 2013

[Yunker, 2017]
John Yunker, *Think Outside the Country*
Byte Level Books, 2017, s. 13 – 22.

Źródła internetowe (WWW)

[WWW-1, 2020]
<https://www.smartling.com/resources/101/internationalization-vs-localization/>, z dnia 24.04.2020.

[WWW-2, 2020]
<https://kissdigital.com/pl/blog/single-page-application-jak-dziala-spa-i-czym-sie-rozni-od-mpa>, z dnia 14.06.2020

[WWW-3, 2020]
<https://docs.microsoft.com/pl-pl/azure/architecture/best-practices/api-design>, z dnia 14.06.2020

[WWW-4, 2020]
<https://pagely.com/blog/multilingual-improve-seo/>, z dnia 16.06.2020.

[WWW-5, 2020]
<https://www.internetworldstats.com/stats7.htm>, z dnia 17.06.2020.

[WWW-6, 2020]
https://w3techs.com/technologies/overview/content_language, z dnia 17.06.2020.

[WWW-7, 2020]
<https://blog.yellowant.com/rest-api-calls-made-easy-7e620e4d3e82>, z dnia 23.06.2020

[WWW-8, 2020]
<https://sniacnajavie.pl/json-web-token/>, z dnia 24.06.2020

[WWW-9, 2020]
<https://rxjs-dev.firebaseio.com/guide/overview>, z dnia 02.07.2020

[WWW-10, 2020]
<https://www.w3.org/International/questions/qa-i18n>, z dnia 12.04.2020.

[WWW-11, 2020]
<https://crackstation.net/hashing-security.htm>, z dnia 16.04.2020.

[WWW-12, 2021]
<https://sass-lang.com/documentation>, z dnia 05.01.2021

[WWW-13, 2021]
https://www.w3schools.com/html/html_intro.asp, z dnia 06.01.2021

[WWW-14, 20201]
<https://jestjs.io/docs/en/getting-started>, z dnia 12.02.20201

Streszczenie

Wyższa Szkoła Informatyki i Zarządzania z siedzibą w Rzeszowie

Kolegium Informatyki Stosowanej

Streszczenie pracy dyplomowej

Systemu do zarządzania tłumaczeniami zawartości aplikacji internetowych

Autor: Yuliia Hrynyk

Promotor: dr inż. Mariusz Wrzesień

Słowa kluczowe: tłumaczenia treści stron, zarządzanie wielojęzycznością, aplikacje jednostronicowe, REST API

Celem danej pracy dyplomowej jest implementacja aplikacji internetowej, która uprości proces internacjonalizacji aplikacji internetowych. Serwis, który jest produktem końcowym jest to aplikacja jednostronicowa po stronie klienta oraz REST API w technologii .NET Core 3.1 po stronie serwera. W ramach pracy dyplomowej zostały przeanalizowane dostępne narzędzia dla rozwiązania danego problemu, wady oraz zalety danych narzędzi i na podstawie tego został wybrany odpowiedni stos technologii. Aplikacja po stronie klienta jest oparta na framework Angular, który jest używany z językiem TypeScript. W celu uproszczenia dodawania stylów do aplikacji została użyta biblioteki SCSS oraz niektóre moduły framework bootstrap, które były dopasowane do własnych potrzeb na podstawie zmiennych SCSS.

Załączniki

Załączniki zostaną zamieszczone na płycie CD, na której będą dodane:

1. Folder z kodem źródłowym do aplikacji internetowej.
2. Folder z kodem źródłowym do serwisu do zarządzania tłumaczeniami.
3. Folder z kodem źródłowym do REST API.