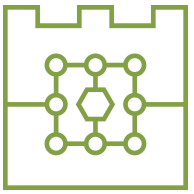
Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji



Tomasz Torbus

Numer albumu: 134237

Serwis informatyczny do identyfikacji i zapobiegania interakcjom lekowym w celu poprawy bezpieczeństwa farmakoterapii

IT service for identifying and preventing drug interactions to improve safety of pharmacotherapy

Praca Inżynierska
na kierunku Informatyka

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Kraków, 2023

1. Streszczenie

Prezentowana praca inżynierska skupia się na opracowaniu nowoczesnego serwisu internetowego, który wspomagać będzie użytkownika i lekarza w zakresie odpowiedzialnego doboru lekarstwa i uniknięcia szkodliwych interakcji pomiędzy nimi. W projekcie wykorzystano technologie takie jak Vue.JS, Python, MongoDB wraz z bibliotekami Vuetify, FastAPI. W ramach projektu zostało utworzone API, które służy do komunikacji pomiędzy warstwami frontendu i backendu, odpowiada one za wyszukiwanie leków, interakcji pomiędzy nimi i również rejestracją, logowaniem użytkowników i obsługą funkcji związanych z posiadanym kontem w systemie. Wyniki testów wykazały, że projekt spełnia swój cel i zostały osiągnięte wcześniej ustalone założenia, zwracane wyniki są trafne, a czas oczekiwania na nie jest niewielki, około 0,17 sekundy.

Abstract

The presented engineering work focuses on the development of a modern web service that will support the user and doctor in the responsible selection of medicines and avoiding harmful interactions between them. In the project, technologies such as Vue.JS, Python, MongoDB along with Vuetify, FastAPI libraries were used. As part of the project, an API was created, which is used for communication between the frontend and backend layers, it is responsible for searching for drugs, interactions between them, and also for registering, logging in users, and handling functions related to the account owned in the system. The test results showed that the project fulfills its purpose and the previously established assumptions have been achieved, the returned results are accurate, and the waiting time for them is small, about 0.17 seconds.

Spis treści:

2. Wstęp

2.1. Cel Pracy

Celem niniejszej pracy jest zaprojektowanie i wykonanie serwisu internetowego przeznaczonego dla urządzeń przenośnych oraz komputerów. Usługa ta na celu będzie mieć wspomaganie pacjenta i lekarza w podejmowaniu istotnej decyzji, jaką jest dobór nowego leku dla pacjenta. Pierwszorzędnym celem tego serwisu jest identyfikacja potencjalnych, niepożądanych, szkodliwych interakcji między lekami przyjmowanymi przez pacjenta. Istotą systemu jest intuicyjny, minimalny, napisany prostym językiem, minimalizujący ryzyko popełnienia błędów interfejs użytkownika, dzięki któremu korzystanie z serwisu nie będzie sprawiać problemu nawet osobom starszym.

Dodatkowym aspektem, który ma na celu ułatwić proces korzystania z serwisu poprzez personalizację ustawień dla użytkownika, jest możliwość utworzenia swojego konta i utworzenia swojej ‘apteczki’ czyli zbioru aktualnie przyjmowanych leków. Funkcjonalność ta jeszcze bardziej zwiększa wygodę użytkowania poprzez eliminację potrzeby wpisywania już przyjmowanych leków przy każdorazowym użytkowaniu wyszukiwarki.

Kluczową cechą serwisu jest zapewnienie szybkości działania przez zminimalizowanie wielkości zasobów przesyłanych pomiędzy serwerem a użytkownikiem i zmniejszenie złożoności obliczeniowej [37] algorytmu do wyszukiwania interakcji. Szybkość dostępu do informacji gra kluczową rolę w celu zapewnienia pozytywnych doświadczeń użytkownika.

2.2. Motywacja

Motywacją do podjęcia się tematu tego projektu jest nieustanny wzrost polifarmacji, czyli przyjmowaniu wielu leków jednocześnie [1], najczęściej przez osoby starsze. Na przestrzeni dwóch dekad procent osób starszych powyżej 65 roku życia, które przyjmują więcej niż 5 leków wzrosła z 12% do 49% w Anglii [2] oraz między latami 1999-2012 z 24% do 39% w Ameryce [3]. Te alarmujące liczby sprawiły, abym podjął się próby wykonania serwisu, który będzie chronił przed złym doborem medykamentów.

Pomimo istnienia już różnych serwisów oferujących sprawdzenie interakcji między lękowych, brakuje nieskomplikowanego rozwiązania, które byłoby łatwe i intuicyjne w obsłudze przez wszystkich.

2.3. Zakres pracy

Do zakresu pracy należy przegląd rozwiązań dostępnych publicznie w internecie, które oferują porównanie interakcji między lekami następnie poddanie analizie ich w celu znalezienia dobrych i słabych stron, aby stworzyć konkurencyjne rozwiązanie. Potrzebne również było pozyskanie danych kluczowych do wykonania projektu. Następnie mając pewność wiarygodności pozyskania danych, wykorzystując osobistą wiedzę i publikacje zaprojektować prosty w obsłudze interfejs do interakcji z danymi. Dla użytkownika końcowego serwisu potrzebne jest zaprojektowanie wygodnego w obsłudze interfejsu graficznego a dla dewelopera i przyszłości projektu, zaprojektowanie API. Po wykonaniu powyższych komponentów wymagana jest analiza czy finalny projekt spełnia wymagania.

2.4. Metodyka

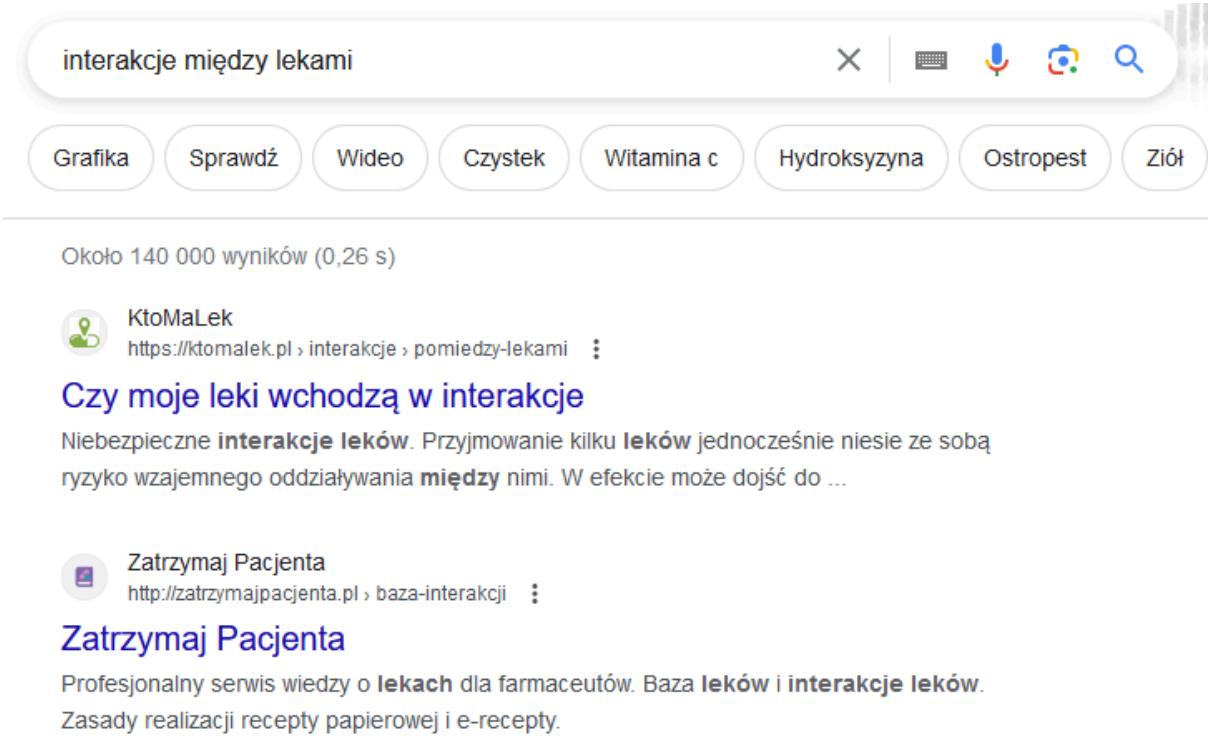
Analizy przydatności i jakości danych w pierwszym etapie zostały wykonane manualnie. Mowa tu o próbie sprawdzenia, czy dane pomiędzy dwoma zebranymi bazami danych, są możliwe do połączenia między sobą. Baza danych leków dostępnych na polskim rynku pobrana z [38] wymagała analizy czy możliwe jest zautomatyzowane przetłumaczenie składników leków, by pasowały one do bazy udostępnionej przez DrugBank [35]. Po skutecznych testach zaczęto projektować rozwiązanie dla pozyskiwania i możliwości przetwarzania danych. Zaczęto od utworzenia interfejsów do bazy danych, aby swobodnie pozyskiwać dane, następnie zaprojektowano API, które oferuje dla etapu tworzenia interfejsu użytkownika łatwy sposób do pozyskiwania danych, a w następnej kolejności utworzono interfejs użytkownika, dzięki któremu przez przeglądarkę internetową można wejść w interakcję z serwisem.

2.5. Analiza konkurencji

Ponieważ na rynku istnieją już podobne rozwiązania, konieczne jest wykonanie analizy oferujących te same usługi. W porównaniu zostaną rozpatrzone kryteria:

- Funkcjonalności usługi,
- Czytelność wyników,
- Dodatkowe funkcjonalności,
- Łatwość w zrozumieniu strony.

Na Rys. 1. przedstawione są wyniki zapytania w wyszukiwarce Google o frazę „interakcje między lekami”, otrzymujemy dwa serwisy, które takie usługi oferują.



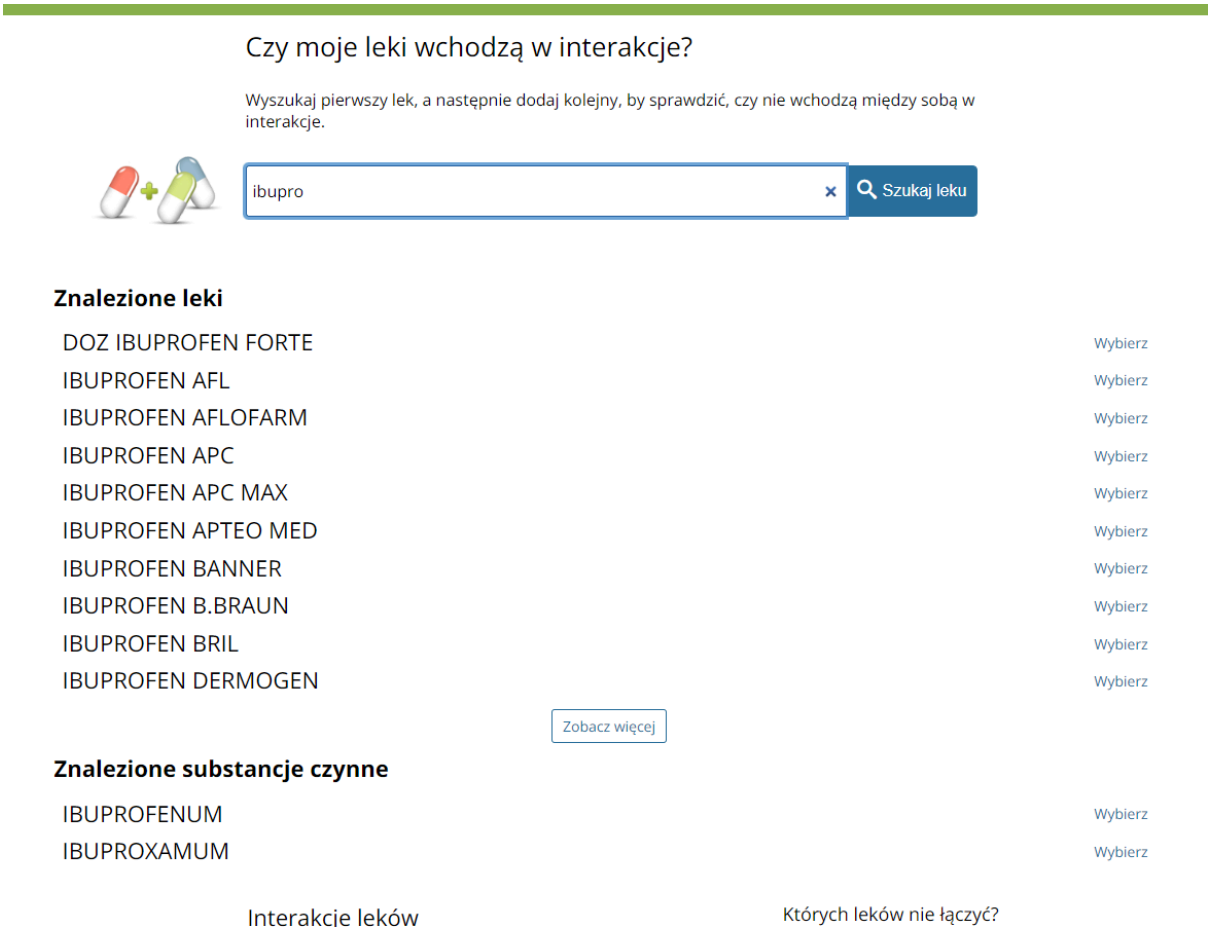
Rys. 1. Wyniki wyszukiwania w Google.pl frazy „interakcje między lekami”.

W wyszukiwarce jako trzeci wynik również pojawia się następna strona (https://leki.pl/interakcje/), lecz przez czas tworzenia niniejszej pracy miała przerwę techniczną, więc została pominięta w porównaniu. Do porównania zostanie również zawarta zostanie anglojęzyczna strona DrugBank [35], która oprócz dostarczenia danych do projektu niniejszej pracy, oferuje również porównanie na swojej stronie.

Analiza pierwszej strony — https://ktomalek.pl/l/interakcje/pomiedzy-lekami:

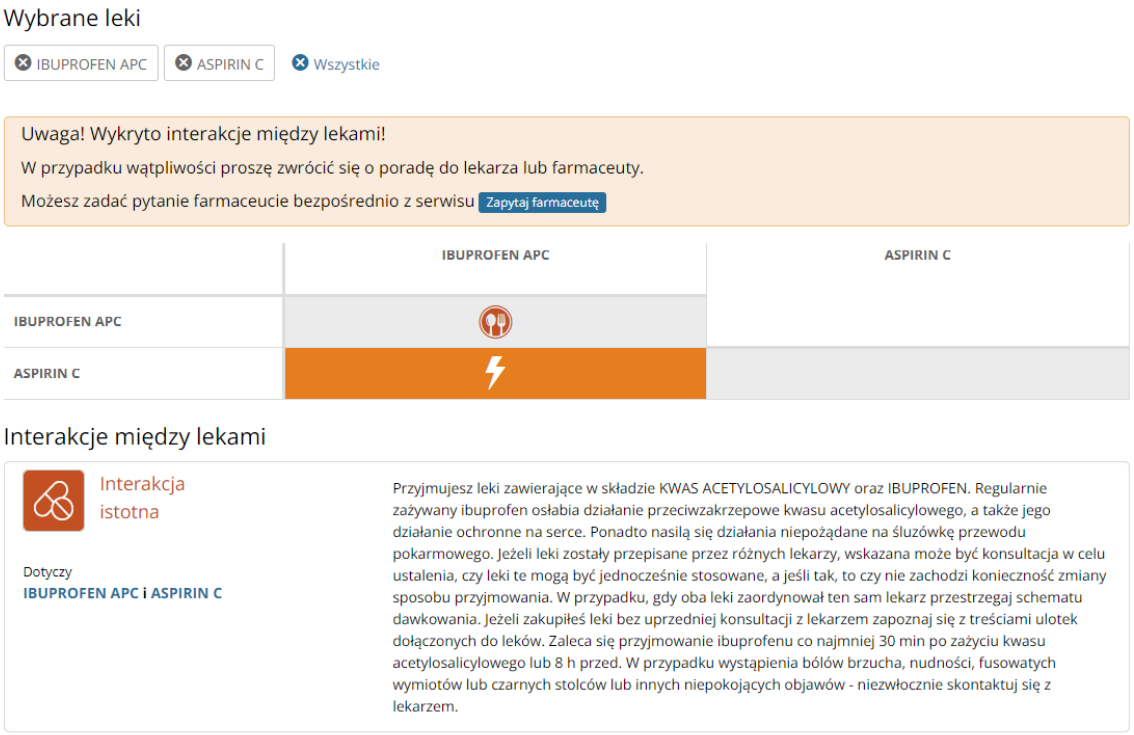
Rys. 2. Widok strony https://ktomalek.pl/l/interakcje/pomiedzy-lekami.

Pierwsza strona, widoczna na Rys. 2., w naszym porównaniu jest elementem składowym strony, która oferuje wyszukiwanie leków w pobliskich aptekach. Strona jest przejrzysta, responsywna i dodatkowo oferuje mały tekst edukujący na temat interakcji między lekowych. Dodatkowym atutem jest tutaj możliwość porównania leków z alkoholem i żywnością. Samo wyszukiwanie leków dostępnych do porównania oferuje bardzo wiele wyników, chociaż nie jest ono do końca przejrzyste, ponieważ wygląda to jak zawartość strony i nie jest jasno widoczne powiązanie listy wyników z naszym wyszukaniem:



Rys. 3. Wyszukanie na stronie https://ktomalek.pl/l/interakcje/pomiedzy-lekami.

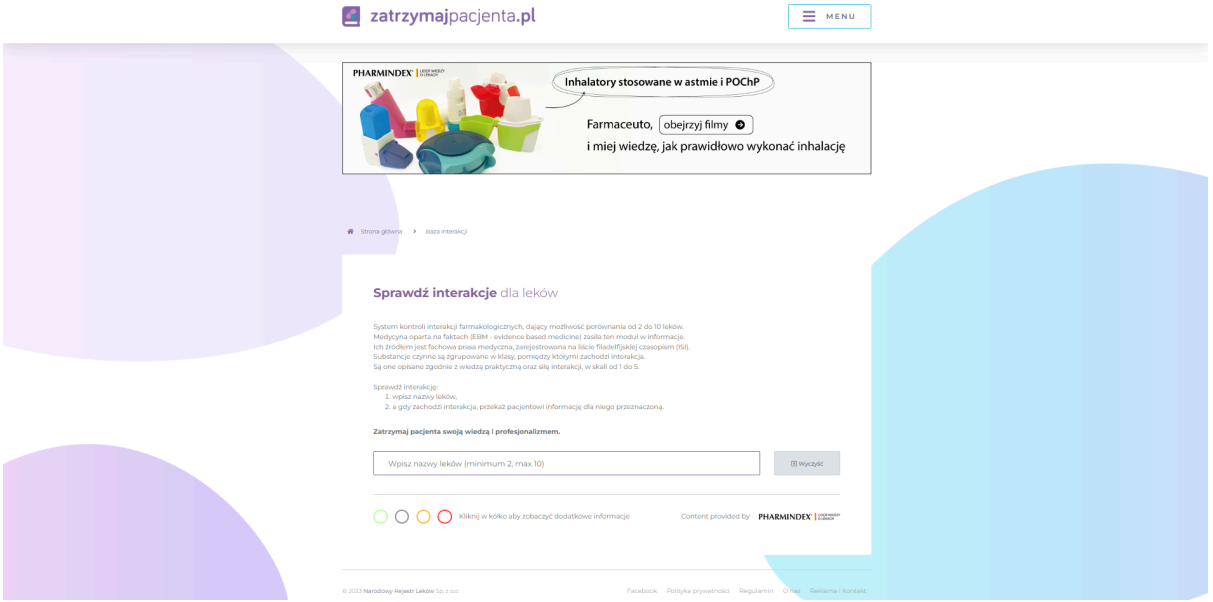
Wyniki wyszukiwania, przedstawione na Rys. 3., prezentują się jako tabela, gdzie przecinające się kolumny z wierszami wskazują, pomiędzy jakimi lekami następuje interakcja. Niestety dużym minusem tego wyszukana jest fakt, że przy każdym wyborze leków strona jest przeładowywana, co wpływa na doświadczenia użytkownika.



Rys. 4. Wyniki wyszukania na stronie https://ktomalek.pl/l/interakcje/pomiedzy-lekami.

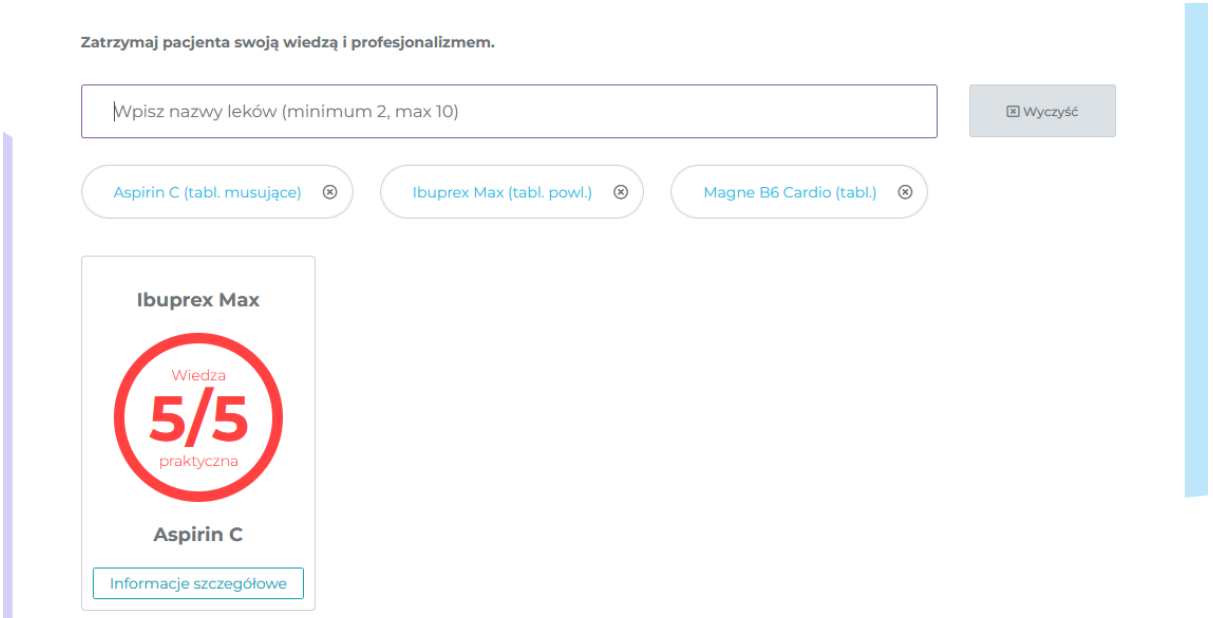
Podsumowując porównanie, warto zwrócić jeszcze uwagę na elementy, które skutecznie odwracają uwagę użytkownika od najważniejszego celu strony — wyszukiwania interakcji między lekami. Pierwszym elementem, który jest zauważalny na stronie, jest wielka, czerwona reklama, dodatkowo jest ona ruchoma, co jeszcze bardziej przyciąga do niej uwagę. Zaraz pod reklamą znajduje się niebieski przycisk „Powrót", którego jedynym zadaniem jest wykonanie skryptu JavaScript do powrotu na poprzednio odwiedzaną stronę, nawet jeśli nie jest to strona z domeny KtoMaLek.pl. Nie tylko może to zdezorientować użytkownika, który dla testu użyje tego przycisku, ale została utracona szansa na możliwość użycia tego przycisku do powrotu na główną stronę, by zatrzymać użytkownika na dłużej. Strona oferuje usługi kontaktu z farmaceutami w różnych etapach korzystania ze strony, od wspomnienia o tym w wynikach po nieustannie widoczny widget po prawej stronie ekranu, podczas gdy nie jest to rzecz szkodliwa, ponieważ dzięki temu użytkownik nie narazi się na problem zażywania złego leku, informację tą można zawrzeć tylko w jednym miejscu, by nie była ona tak natarczywie promowana. Ponadto zauważono dziwne, niezidentyfikowane działanie strony podczas cofania się w przeglądarce, podczas niego lądujemy na stronie "about:blank", która na nic nie wskazuje.

Analiza drugiej strony — http://zatrzymajpacjenta.pl/baza-interakcji#



Rys. 5. Widok strony http://zatrzymajpacjenta.pl/baza-interakcji#.

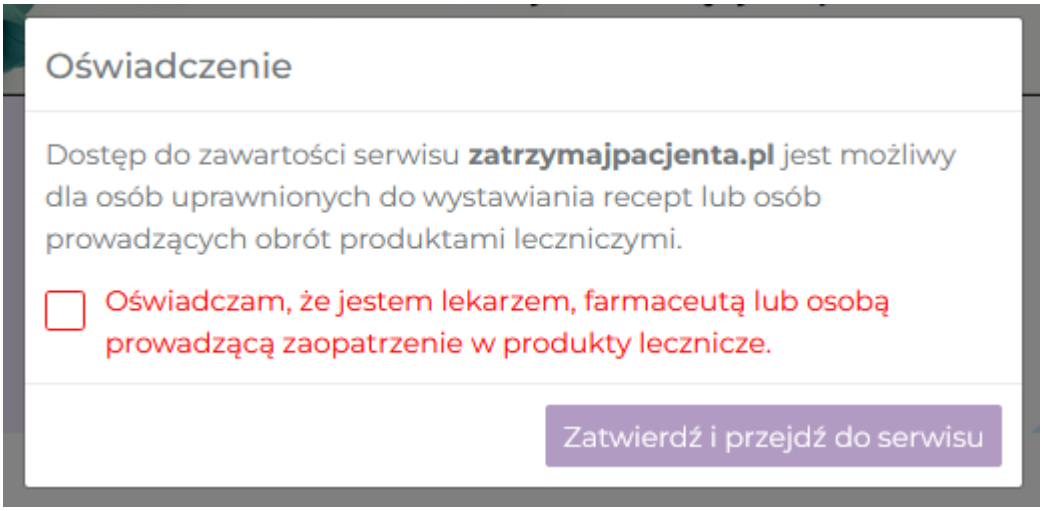
Pierwszą rzeczą, którą można zauważyć podczas korzystania ze strony, jest jej nowoczesny, przyjazny dla oka wygląd. Strona jest responsywna i bardzo czytelna z zapewnieniem, że komponenty na stronie mają jasno wyrażone intencje i nie zwodzą użytkownika. Dodatkowo strona oferuje krótki tekst edukujący na temat tego, skąd wzięte są dane.



Rys. 6. Wyszukanie na stronie http://zatrzymajpacjenta.pl/baza-interakcji#.

Podczas wyszukiwania wyniki prezentowane są na bieżąco, bez potrzeby wciskania przycisku wyszukaj, co znacząco zwiększa komfort korzystania ze strony. Wyniki są czytelne, z wyraźnym zaznaczeniem jak poważna interakcja może wystąpić, użytkownik nie jest przeciążony nadmiarem informacji, a w razie potrzeby przeczytania szczegółowych wyników, bez problemu można je uzyskać przez rozwinięcie.

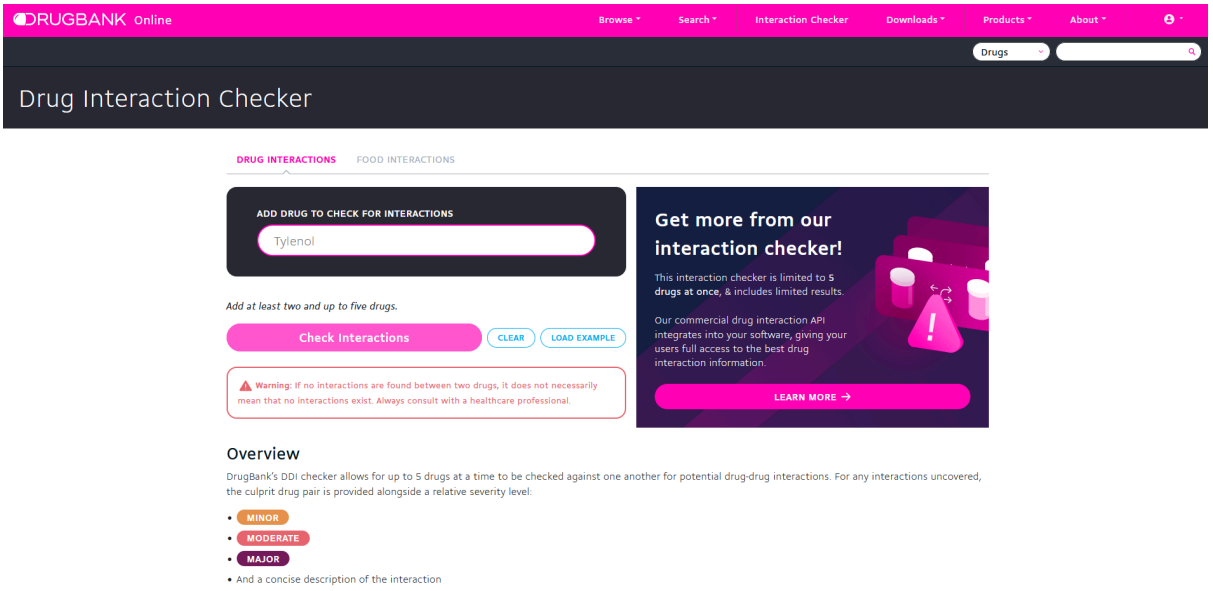
Podsumowując porównanie strona wypada bardzo dobrze w odbiorze wizualnym jak i funkcjonalnym. Jednak bardzo ważnym elementem, który trzeba wziąć pod uwagę, jest komunikat, który widoczny jest na Rys. 7.



Rys. 7. Oświadczenie przy pierwszym wejściu na stronę http://zatrzymajpacjenta.pl/baza-interakcji#.

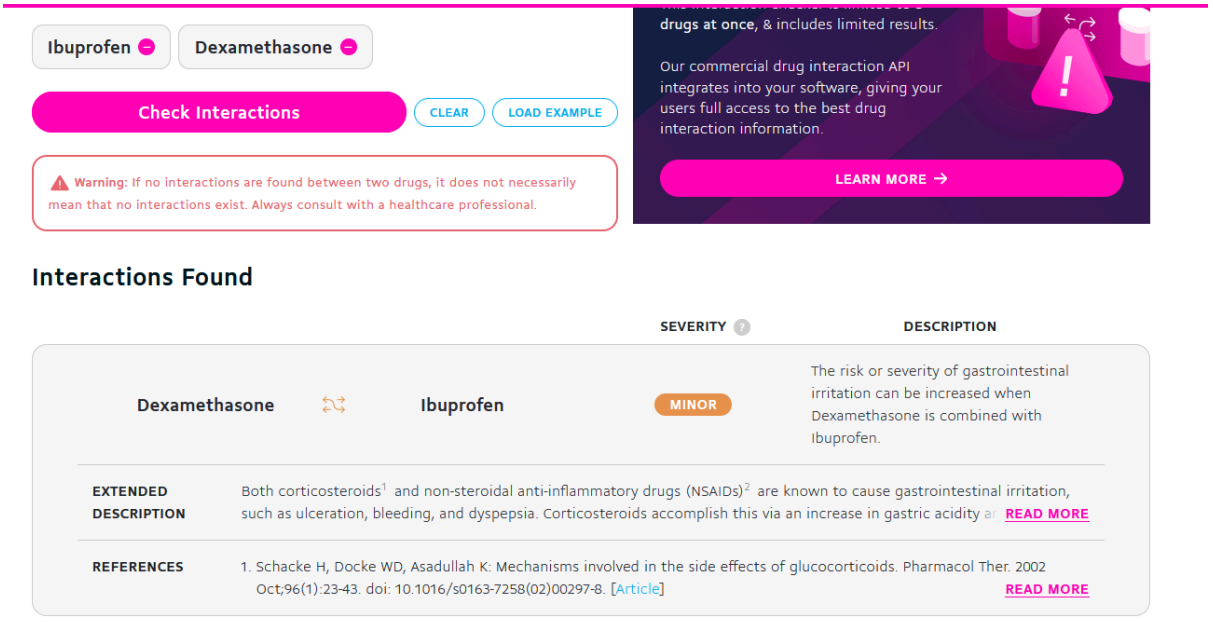
Jest to jednoznaczna informacja, że strona nie powinna być do użytku przez osoby nieuprawnione, za czym idzie, według jej zasad nie powinniśmy z niej korzystać jako zwykli pacjenci.

Analiza trzeciej trony — https://go.drugbank.com/drug-interaction-checker



Rys. 8. Widok strony https://go.drugbank.com/drug-interaction-checker.

Jest to strona, która zostanie opisana bardziej szczegółowo w rozdziale 4.5.1. Po wejściu na stronę otrzymujemy dostęp do sprawdzenia interakcji między składnikami leków i między składnikami a jedzeniem. Strona zawiera krótką informację, w jaki sposób interpretować wyniki i również krótki tekst opisujący czym są interakcje między lekowe. Strona jest oferowana jedynie w języku angielskim, a Polskie leki nie są zawarte w ich bazie. Pomimo tego braku strona oferuje rzetelne źródło informacji na temat interakcji pomiędzy składnikami wraz z podaniem publikacji na temat tejże interakcji.



Rys. 9. Wyszukanie interakcji na stronie https://go.drugbank.com/drug-interaction-checker.

Podsumowując porównanie, strona zawiera bardzo szeroki zakres wiedzy na temat interakcji, lecz niestety poprzez niedostępność jej w języku polskim i brak zawartych w niej Polskich leków nie może być skutecznie używana przez osoby z niewielką umiejętnością angielskiego. Dodatkowo strona nie jest przejrzysta a całe mnóstwo innych informacji, odciąga użytkownika od jego głównego celu, czyli wyszukania interakcji między lekami.

Część teoretyczna

3.1. Polifarmacja

Polifarmacja to termin, który jest używany w medycynie w odniesieniu do sytuacji, w której pacjent przyjmuje więcej niż jeden lek jednocześnie [1]. Termin ten jednak nie został nigdy oficjalnie zdefiniowany i przez to nie wskazuje on na minimalną ilość leków, które muszą być przyjmowane, tak więc termin ten zależy od interpretacji [4]. W tej pracy jako polifarmację uznawać będę przyjmowanie dwóch i większej ilości leków.

Polifarmacja najczęściej jest konsekwencją przyjmowania leków w celu zwalczenia występujących dwóch lub większej ilości długotrwałych schorzeń, ponieważ ich występowanie najczęściej prowadzi do konsekwencji przyjmowania wielu osobnych środków, aby zwalczyć wszystkie występujące choroby. Gdy pacjentowi przepisywana jest nowy lek, lub w przypadku gdy pacjent sam kupuje leki bez recepty, często system zdrowia nie bierze pod uwagę wszystkich potrzeb osoby, tylko skupia się na pojedynczym schorzeniu lub chorobie [5]. Ponieważ termin polifarmacji nie jest jednoznacznie określony [4], szacuje się, że występowanie zjawiska polifarmacji to od 10% do aż 90% w różnych populacjach [6].

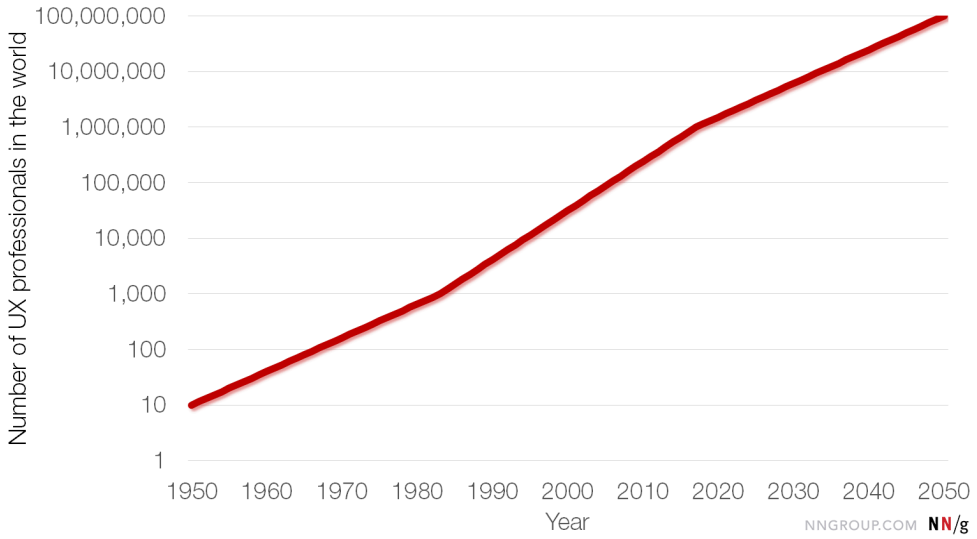
Zjawisko to jest konsekwencją rosnącej ilości schorzeń współistniejących pośród populacji starzejącej się na całym świecie. Według badań [7] około 11% niezaplanowanych wizyt w szpitalu wiąże się ze szkodą wyrządzoną poprzez zażywanie wielu leków, a ponad 70% niezaplanowanych wizyt starszych osób wiąże się ze szkodą wyrządzoną również poprzez zażywanie wielu leków. Aktualnie pacjenci z wieloma schorzeniami to jedno z najcięższych wyzwań dla ochrony zdrowia.

W Polsce na podstawie badań [8] osoby starsze (powyżej 65 roku życia) do przyjmowania więcej niż 5 leków przyznaje się 53.5% badanych, a do przyjmowania więcej niż 10 leków jednocześnie przyznaje się 8.7% gdzie choroby współistniejące to główny czynnik, który powoduje przyjmowanie tak wielu leków [8].

3.2. UI i UX

3.2.1 Czym jest UX?

UX (ang. User Experience) doświadczenia użytkownika, jest to proces definiujący doświadczenia użytkownika wiążące się z użytkowaniem danego produktu. Coraz więcej osób poświęca się zostaniu projektantem doświadczeń użytkownika, który pomaga projektować pozytywne doznania z produktem.



Rys. 10. Profesjonaliści doświadczeń użytkownika na całym świecie. Historia i przewidywania [10].

Z założenia każda zaprojektowania interakcja z produktem lub usługą ma być pozytywnie oceniona przez użytkownika. Sam proces „ścieżki doświadczenia" klienta, który wchodzi w taką interakcję, jest bardzo rozbudowany. Dla zobrazowania sytuacji pod uwagę weźmy dwie opcje przykładowej ścieżki zakupu telewizora wraz z opisem doświadczenia użytkownika o produkcie na danym etapie:

Opcja pierwsza:

1. Powstaje potrzeba kupna telewizora — zwykła chęć lub poprzedni uległ awarii.
2. Wyszukanie w Internecie informacji o telewizorach.
3. Wejście na stronę sklepu RTV AGD lub strony producenta — **to tutaj następuje już pierwszy kontakt z produktem.**
4. Przejrzenie oferty, wybór najlepiej pasującego i najlepiej prezentującego się telewizora dla klienta — jeżeli wykonywane na stronie producenta, ważna jest łatwość dostępu do informacji, nawigacji po stronie i dokonania zakupu.

5. Zamówienie, dostawa do domu/odbiór.
6. Otrzymanie pudła — tutaj jest oceniany sposób, w jaki prezentuje się opakowanie, czy nie jest uszkodzone.
7. Odpakowanie produktu — Czy produkt został dobrze zapakowany? Czy w łatwy i intuicyjny sposób wyciąga się z pudełka? Czy materiały, w które był telewizor zapakowany, sprawiają wrażenie porządnych?
8. Pierwszy kontakt z produktem — Czy nie jest on uszkodzony? Czy jest zrobiony z porządnych materiałów? Jak pachnie?
9. Czytanie instrukcji obsługi — Czy jest ona napisana prostym i zrozumiałym językiem? Czy jest zachowana hierarchia, dzięki której łatwo się odnaleźć? Czy wszystkie pytania są w niej odpowiedziane?
10. Montaż i podłączenie telewizora — Czy wszystkie kable i elementy montażowe są dostarczone wraz z telewizorem? Czy elementy montażowe są godne zaufania i udźwignąą telewizor? Czy w instrukcji jest o tym napisane?
11. Pierwsze uruchomienie telewizora — Czy przyjaźnie nas „wita”? Czy proces wdrażania jest intuicyjny? Czy interfejs użytkownika jest jasny? Czy jest on w naszym języku?
12. Użytkowanie telewizora — Czy dobrze grają głośniki? Czy obraz jest dostatecznie jasny i wyraźny? Czy posiada on funkcje, które były deklarowane już na stronie producenta? Czy pilot ma dobry zasięg i ma wygodne rozłożenie przycisków?
13. Długotrwałe użytkowanie telewizora — jak zachowuje się po latach? Czy wszystko jest dalej tak samo sprawne? Czy w razie awarii, dostarczone w instrukcji lub na stronie producenta instrukcje były jasne? Czy wraz ze wzrostem rozwoju technologicznego, dalej może on być określany za dobry?
14. Koniec życia telewizora — Czy w instrukcji lub na stronie producenta było podane gdzie go wyrzucić? Czy Zrobione zostało wszystko, co można by go utrzymać przy sprawności? Czy jest może zniżka, która pozwoli wymienić stary telewizor tej marki na nowy tej samej marki?

Opcja druga:

1. Powstaje potrzeba kupna telewizora — zwykła chęć lub poprzedni uległ awarii.
2. Pójście do najbliższego sklepu w celu przejrzenia telewizorów.
3. Pierwsze zetknięcie się z produktami — Który z nich prezentuje się najlepiej w określonym budżecie? Czy któryś może wyróżnia się na tyle, że jesteśmy gotowi dać

więcej? Czy pracownik prezentujący telewizory rozwiał nasze wątpliwości, a może polecił coś lepszego?

4. Zakup produktu — Czy produkt będzie dobrze zapakowany do transportu do domu?
5. W tym momencie powtarzają się punkty z opcji pierwszej od 7 do 14.

Ścieżki te zostały lekko uproszczone, ponieważ nie wzięliśmy pod uwagę faktu takich jak marketing, przeglądanie jednoczesne ofert w sklepach stacjonarnych i internecie, opinii znajomych, i tym podobnych. Jak widać, jest wiele punktów, o które dobry projektant UX musi zadbać, by dać jak najlepsze wrażenie produktu.

UX design w przeciwieństwie do *UI (ang. user interface) Interfejsu użytkownika* designu nie kieruje się tylko estetyką, lecz użytecznością [39], zapotrzebowaniem, marką i wieloma innymi. UX najczęściej jest podparty również badaniami. Badania takie często opierają się na ankietach, testach wyboru czy użytkownik woli projekt A lub projekt B i odpowiedzi zwrotnej dostarczonej bezpośrednio przez użytkownika.

3.2.2. Czym jest UI?

Interfejs użytkownika (UI), jest to punkt interakcji człowieka z maszyną. Pojęcie to jest to bardzo obszerne, ponieważ pokrywa ono interakcję człowieka z maszyną za pomocą ekranów, klawiatur, myszek, mikrofonów, kamer, języka, stron internetowych. Celem interfejsu użytkownika jest zapewnienie skutecznej, łatwej, efektywnej interakcji człowieka z maszyną, która równocześnie daje informację zwrotną użytkownikowi, która może zachodzić w formie graficznej, dźwiękowej, dotykowej. Wyróżnia się parę głównych interfejsów użytkownika [11]:

- GUI (Graphical User Interface) — graficzny interfejs użytkownika, mogą to być strony internetowe, aplikacje mobilne
- CLI (Command Line Interface) — interfejs poprzez wiersz poleceń, głównie do interakcji z komputerem i maszynami za pomocą poleceń tekstowych
- Menu-driven User Interface — interfejs użytkownika oparty na menu, może to być menu w restauracji, menu telewizora, menu kasy samoobsługowej

- Natural language user interface — interfejs użytkownika oparty na interakcji poprzez język naturalny, może to być na przykład wyszukiwarka semantyczna lub współczesne modele sztucznej inteligencji

Jak widać, sposobów na interakcję człowieka z maszyną może być więcej, niż tylko graficzne, istnieją również interakcje głosowe, za pomocą gestów, za pomocą ruchów. Istnieje wiele zasad dotyczących projektowania dobrego interfejsu użytkownika, ale jedną z najbardziej charakterystycznych jest 10 heurystyk Nielsena [12]. Jest to ogólny zbiór zasad interakcji człowieka z maszyną, które pozwalają zaprojektować i ocenić użyteczność danego interfejsu. Nazwa wzięła się od nazwiska autora tych Heurystyk: Jakoba Nielsena.

Prezentują się one następująco [12]:

1. **Widoczność statusu systemu.**

   Zasada ta mówi o ciągłym zapewnieniu użytkownika informacji zwrotnej. Informacja powinna prezentowana być w możliwie jak najszybszym czasie i w jak najbardziej oczywisty sposób. Przykładami są statusy ładowania strony, procentu postępu wgrywania pliku, po dodaniu produktu do koszyka, na koszyku pojawia się liczba. Gdyby podczas użytkowania interfejsu, „zamarzłby" on w miejscu, moglibyśmy założyć, że coś się zacięło lub nie działa, prawda mogłaby być natomiast inna, operacja wywołana przez użytkownika wymaga czasu.

2. **Zachowaj zgodność pomiędzy systemem a rzeczywistością.**

   Zasada ta mówi o tym, że design interfejsu powinien „przemawiać językiem użytkownika". Język powinien być dostosowany do użytkownika, bez potrzeby dla użytkownika by szukał definicji słowa. Design elementów (graficznych) powinien odwzorowywać jak najbliżej nasze realne środowisko. Przykładem jest kosz w systemach operacyjnych, ponieważ jesteśmy wyuczeni znaczenia tego przedmiotu w rzeczywistości, szybko skojarzymy, że właśnie tam znajdują się usunięte przez nas rzeczy.

3. **Kontrola i swoboda użytkownika.**

   Zasada ta przypomina nam o tym, że użytkownicy często popełniają błędy. Powinno zostać zapewnione „wyjście awaryjne", czyli cofnięcie wykonanej akcji użytkownika. Perfekcyjnym przykładem na to jest funkcja cofania wysłanego maila, które dostępne jest w gmail.com/ przez pierwsze sekundy po wysłaniu wiadomości,

lub wyraźnie zaznaczony przycisk anulowania akcji.

4. **Spójność i standardy.**

Zasada mówi o fakcie, że użytkownicy spędzają większość swojego czasu, używając innych produktów niż naszego. Dlatego przysłowiowo „nie powinniśmy wymyślać koła na nowo", ponieważ użytkownicy są już wyuczeni pewnych wzorców, i rozwiązań więc powinniśmy wykorzystać ich potencjał i przyzwyczajenia użytkownika. Przykładem są „hamburger" menu, recepcja z przodu hotelu, menu restauracji dostępne przy ladzie.

5. **Zapobieganie błędom.**

W każdym momencie to my powinniśmy zabezpieczyć użytkownika przed popełnieniem błędu, ponieważ musimy założyć, że użytkownik nie zna tak dobrze naszego systemu i jego intencji jak my, jego twórcy. Przykładem będzie walidacja maila przy jego wprowadzaniu, napisy na drzwiach „Pchnij" „Ciągnij".

6. **Wybór zamiast zapamiętywania.**

Zasada ta przypomina, że użytkownik powinien mieć ciągły dostęp do pomocy w kontekście tego co robi. Ludzie mają ograniczoną pamięć krótkotrwałą więc, zamiast patrzeć w dokumentacje, instrukcję produktu, rozwiązania, lepiej dla użytkownika dostarczyć kontekst lub podpowiedzi. Przykładem jest przewaga interfejsu graficznego przeglądarki plików w komputerze nad wierszem poleceń. Użytkownik nie musi pamiętać skomplikowanych poleceń do terminala, tylko w łatwy sposób ma graficzny podgląd w to co robi.

7. **Zapewnić elastyczność i efektywność.**

Zasada ta przypomina, że z naszego systemu będą korzystać użytkownicy mniej i bardziej zaawansowani. Użytkownikom bardziej doświadczonym powinniśmy dostarczyć skróty i ułatwienia, które przyspieszą pracę, ale nie będą one wymagane do osiągnięcia rezultatu przez mniej doświadczonych użytkowników. Przykładem są na przykład skróty klawiszowe.

8. **Estetyka i minimalny design.**

Interfejs nie powinien zawierać informacji, które nie są tam konieczne lub

rzadko wykorzystywane. Każda z dodatkowych, niepotrzebnych opcji daje użytkownikowi szansę na pomyłkę. Opcje ekstra powinny domyślnie być schowane.

9. **Zapewnij obsługę błędów.**

   Nawet przy najlepszych naszych staraniach nie jesteśmy w stanie przewidzieć wszystkich możliwych błędów. Dlatego przy wystąpieniu błędu powinniśmy w zrozumiały użytkownika sposób dostarczyć informację, że błąd nastąpił, a następnie wskazać mu sposób na jego poprawę. Przykładem są komunikaty z błędem z pogrubionym, czerwonym tekstem, znaki drogowe, które wskazują na zły kierunek jazdy, strony błędów takich jak 404 które mają listę rzeczy, które można w przypadku błędu zrobić.

10. **Zapewnij pomoc i dokumentację.**

    Pomimo że idealistycznym rozwiązaniem byłby system, który nie wymaga dalszego tłumaczenia, powinniśmy dostarczyć dokumentację dla użytkownika. Dokumentacja powinna być łatwa do zrozumienia i powinna zawierać listę kroków, które użytkownik musi wykonać, by wykonać czynność, którą chce. Przykładem jest dokumentacja techniczna i punkt informacyjny w galeriach, lotniskach, muzeach.

3.2.3. Istota UX i UX w tym projekcie

Jednym z głównych założeń tego projektu jest prostota użytkowania. Intuicyjny i czytelny interfejs jest szczególnie istotny dla osób w różnym wieku i o różnym stopniu biegłości w użytkowaniu internetu i urządzeń mobilnych. Dobrze zaprojektowany interfejs skraca czas potrzebny na uzyskanie oczekiwanych rezultatów, co jest krytyczne w sytuacjach, w których mamy ograniczony czas. Serwis ten będzie pozwalać na minimalną ilość wykonanych błędów poprzez narzucenie użytkownikowi tylko wyborów z dostępnej listy.

3.3. HTML i CSS

Podstawą wykonania każdej strony jest wykorzystanie *HTML (HyperText Markup Language)*. Język ten definiuje strukturę strony internetowej i jest podstawą do wyświetlania

stron. Rozwój tego języka rozpoczął się już w 1990 roku przez Tima Bernesa-Lee, struktura języka oparta została na *SGML (Structured General Markup Language)*, który używa tagów do nadania znaczeniu elementów. Język ten przeszedł wiele zmian i został ustandaryzowany w 2008, a w 2014 roku powstał oficjalny standard o nazwie *HTML5*. [13]. Element języka HTML [40] został przedstawiony na Rys. 11.

```
<span> Zawartość </span>
```

Rys. 11. Przykładowy element HTML.

Gdzie:

`<span> </span>` - to tagi otwierające i zamykające. W tym przypadku tag „span" jest generycznym elementem stosowanym dla tekstowej zawartości, który sam w sobie nie zawiera dodatkowych funkcjonalności.

Zawartość pomiędzy tagami jest objęta efektem danego tagu.

Elementy, a raczej tagi otwierające mogą zawierać atrybuty takie jak `class`, który służy do nadania klasy elementowi. Elementy mogą również zawierać inne elementy, dzięki czemu jesteśmy w stanie uzyskać bardziej skomplikowane strony.

Do stylizowania, czyli nadaniu pożądanego wyglądu, strony napisanej w HTML używany jest język deklaratywny *CSS (ang. Cascading Style Sheets) Kaskadowe Arkusze Stylów*. Dzięki niemu możemy wyznaczyć zasady, które będą modyfikować wygląd elementów znajdujących się na stronie za pomocą tagów, klas, identyfikatorów poszczególnych elementów [14]. Odnosić się do elementów również możemy relatywnie, na przykład do dzieci danego elementu, jego rodzeństwa itp. Przykładowa zasada CSS przedstawiona jest na Rys. 12.

```
p {
     font-size: 16px
}
```

Rys. 12. Przykładowa zasada w języku CSS.

Zasada ta odnosi się do wszystkich tagów <p> na stronie i jej zadaniem jest zmiana wielkości czcionki na 16 pikseli. Pełny opis CSS wraz z przykładami możemy znaleźć w książce [41] *CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny.*

3.4. JavaScript

JavaScript jest interpretowanym język programowania wysokiego poziomu, stworzony w celu umożliwienia w roku 1995 roku i został wynaleziony dla celów przeglądarki Netscape. Jego głównym zadaniem, które w dużej mierze nie zmieniło się, jest manipulowanie zawartością strony internetowej poprzez *DOM (ang. Document Oriented Model) Obiektowy model dokumentu*, manipulowaniem danych poprzez *AJAX (ang. Asynchronous JavaScript And XML)*, interakcje przeglądarki z różnymi API i wiele innych. Pomimo że jego początkowym zadaniem było tylko działanie po stronie klienta, z czasem powstały również rozwiązania serwerowe takie jak *Node.JS* [15]. Obecnie język ten jest najczęściej wybieranym językiem na świecie [22].

Głównymi cechami tego języka jest fakt, że jest to język interpretowany, czyli nie wymaga kompilowania kodu, co znaczy, że wykonywany przez interpreter w przeglądarce. Kolejną cechą tego języka jest zorientowanie tego języka na zdarzenia (*ang. events*), dzięki którym możemy tworzyć natychmiastowe interakcje z użytkownikiem, na przykład wykonaniem zapytania do API po wciśnięciu przycisku na stronie lub różne efekty wizualne. Kolejną główną zaletą tego języka jest fakt, że potrafi wykonywać on zapytania asynchroniczne, czyli takie, które nie blokują całego kodu na czas wykonania i może wykonywać się takich zapytań równolegle wiele. Więcej zalet i szerszy opis języka JavaScript można znaleźć w książce [41].

Elastyczność, duże możliwości i łatwość nauki tego języka przyczyniły się do wzrostu popularności owego języka i gwałtownego rozwoju różnych bibliotek dla tego języka, które jeszcze bardziej zwiększają jego możliwości i łatwość użytkowania. Do najbardziej znanych należą:

- Dla tworzenia interfejsów użytkownika: React [43], Angular [44], Vue.js [36],
- Dla tworzenia rozwiązań serwerowych: Node.js [15], Express.js [45],
- Dla tworzenia aplikacji mobilnych: React Native [46],
- Dla tworzenia aplikacji komputerowych: Electron [47].

3.5. Vue.JS i Vuetify

Vue.JS [36] to otwartoźródłowy *framework* do JavaScript do tworzenia interfejsów użytkownika. Opublikowany został przez Evana You, byłego pracownika Google w 2014 roku [17]. Biblioteka ta została stworzona z myślą o łatwości tworzenia i utrzymywania *SPA,* czyli *jednostronicowej aplikacji internetowej (ang. single page application)* [16] oraz bardziej złożonych interfejsów użytkownika. Evan pracował dla Google z frameworkiem Angular, który rozwijany jest przez tę samą firmę. Jego motywację do stworzenia frameworka podsumował następująco [18]: „Pomyślałem, co by było, gdybym mógł po prostu wyodrębnić część, która naprawdę mi się podobała w Angularze i zbudować coś naprawdę lekkiego".

Cechą tej biblioteki jest jej reaktywność. Dzięki rozwiązaniom zawartym w tym frameworku DOM automatycznie jest aktualizowany. Dzięki takiemu rozwiązaniu deweloperzy nie muszą manualnie zarządzać aktualizacjami interfejsu i jego danymi.

Główną cechą, która wpłynęła na wzrost popularności tego języka, jest umożliwienie tworzenia komponentów, w których deweloper może nadawać funkcjonalność, wygląd i strukturę. Natomiast komponent ten może być ponownie używany w różnych częściach systemu, co czyni rozwój aplikacji dużo prostszy. Zaletą tego rozwiązania jest fakt, że możemy tworzyć niemalże nieskończone się zagnieżdżenia komponentów w sobie, gdzie cały widok strony internetowej może być jednym komponentem, który zawierać będzie inne dzieci-komponenty.

Na Rys. 13 przedstawiono przykładowy komponent VUE, którego zadaniem utworzenie czerwonego przycisku, który po wciśnięciu zwiększa swoją wartość, która się w nim znajduje.

```
<template>
  <button @click="incrementValue" class="sampleBtn">{{value}}</button>
</template>

<script>
export default {
  data() {
  return {
    value: 0,
  }},
```

```
  methods: {
    incrementValue() {
      this.value++;
    }
  }
}
</script>


<style>
.sampleBtn {
  background-color: #ff0000;
  padding: 15px;
}
</style>
```

Rys. 13. Kod dla przykładowego komponentu napisanego przy pomocy biblioteki VUE.JS.

Tak utworzony przycisk za pomocą kodu z Rys. 13. znajduje się na Rys. 14.

Rys. 14. Prosty przycisk na stronie —- stan przed wciśnięciem.

Stan przykładowego przycisku po jego jednokrotnym wciśnięciu znajduje się na Rys. 15.

Rys. 15. Prosty przycisk na stronie — stan po wciśnięciu.

Jak widać, dzięki vue nie jest konieczna potrzeba kodowania rozwiązania, które umożliwiłoby natychmiastową aktualizację wartości w przycisku. Implementacja tego rozwiązania natywnie w JavaScriptcie byłaby czasochłonna i wymagałaby wielokrotnego pisania funkcji, w przypadku użycia podobnej logiki w wielu miejscach.

Vuetify [20] jest biblioteką do wcześniej opisanego frameworka Vue. Biblioteka ta zawiera wiele gotowych, elementów, które jeszcze bardziej pomagają zwiększyć tempo

rozwoju aplikacji. Jak podaje oficjalna strona Vuetify [20] komponenty są spójnie ostylowane wedle specyfikacji material design od Google [19] lecz z zachowaniem możliwości ich przystosowania do własnych potrzeb. Komponenty te zawierają przygotowane już funkcjonalności, które w znaczący sposób pozwalają uniknąć kodowania reużywalnych komponentów od nowa.

Na Rys. 16. znajduje się kod wykonujący to samo zadanie jak kod z Rys. 13., lecz został w nim użyty element biblioteki Vuetify: `<VBtn>` .

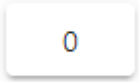
```
<template>
  <VBtn @click="incrementValue">{{value}}</VBtn>
</template>

<script>
  export default {
      data() {
          return {
              value: 0,
          }
      },
      methods: {
          incrementValue() {
              this.value++
          },
      },
  }
</script>
```

Rys. 16. Przykładowy kod napisany z użyciem biblioteki Vuetify.

Jak można zauważyć na Rys. 16., tym razem pominięte jest stylowanie komponentu, by pokazać natywny wygląd elementu z biblioteki Vuetify.

Na Rys. 17., Rys. 18. i Rys. 19. przedstawione są odpowiednio: wygląd przycisku, moment wciśnięcia przycisku, stan przycisku po naciśnięciu.



Rys. 17. Prosty przycisk stworzony za pomocą Vuetify — stan przed wciśnięciem.

Rys. 18. Prosty przycisk stworzony za pomocą Vuetify — stan w trakcie wciśnięcia.

Rys. 19. Prosty przycisk stworzony za pomocą Vuetify — stan po wciśnięciu.

Biblioteka oferuje zarówno spójny wygląd dla komponentów jak i pozytywne doświadczenia użytkownika płynące z pokazania akcji wciśnięcia przycisku.

Framework Vue i biblioteka Vuetify w tym projekcie została wybrana ze względu na szerokie możliwości tego połączenia, ich lekkość dla projektu i znajomość autora tej pracy przez codzienną pracę z nim.

3.6. Python i FastAPI

Język programowania Python stworzony w 1991 roku przez Guido van Rossuma. Język ten jest językiem wysokiego poziomu o ogólnym przeznaczeniu, charakteryzującym się czytelnością składni [21]. Język Python jest dynamicznie typowany, czyli znaczy to, że zmienne nie mają ściśle określonego typu. Wspiera on programowanie obiektowe, co oznacza, że programy definiuje się za pomocą obiektów, elementów łączących stan i zachowanie. Wspiera również proceduralny paradygmat programowania oraz funkcjonalnego. Łatwa składnia, czytelność i szeroki zakres możliwości przyczynił się do tego, że język ten jest 4 najczęściej używanym językiem na świecie, a przez dopiero uczących się ludzi jest 3 najpopularniejszym językiem [22].

W Pythonie można rozwijać w nim aplikacje dzięki frameworkom takimi jak Django [48], Flask [49], FastAPI [24]. Język ten jest najczęstszym językiem wykorzystywanym w analizie danych [23], ponieważ zawiera on bardzo dobrze rozwinięte biblioteki służące w tym celu takie jak Pandas [50], NumPy [51] i SciKit [52]. Język ten umożliwia również uczenie maszynowe, dzięki bibliotekom jak TensorFlow, Pytorch, które w ostatnich latach odnosi coraz większą popularność poprzez rozpowszechnienie się modeli językowych takich jak GPT.

FastAPI to framework do języka Python, który umożliwia budowanie API. Projekt ten bardzo szybko zdobył popularność wśród programistów dzięki głównym jego cechom [24]:

- Szybkość działania — jego bardzo szybkie działanie jest na równi z innymi popularnymi rozwiązaniami takimi jak Node.JS i GO
- Łatwy do zrozumienia – dzięki prostej, niewymagającej składni w bardzo szybki i intuicyjny sposób można utworzyć punkt dostępu przez API.
- Automatyczne tworzenie dokumentacji — dzięki OpenAPI framework ten automatycznie tworzy dokumentację dla API.

Przykładowy kod umożliwiający interakcję z API, przedstawiony na Rys. 20. Kod pochodzi z oficjalnej strony projektu [24].

```
from typing import Union

from fastapi import FastAPI

app = FastAPI()



@app.get("/")
def read_root():
    return {"Hello": "World"}
```

Rys. 20. Przykładowy kod umożliwiający interakcję z API [24].

Po wejściu na główny katalog “/” otrzymamy odpowiedź widoczną na Rys. 21.

```
{"Hello": "World"}
```

Rys. 21. Przykładowa odpowiedź po wejściu na główny katalog “/”.

Oczywiście przykład jest bardzo prosty i może być znacznie bardziej rozbudowywany. Obsługiwane są wszystkie niezbędne metody HTTP takie jak GET, POST, PUT, UPDATE, DELETE itd.

Dobór języka Python i Frameworka FastAPI pokierowany w tym projekcie jest prostotą w użyciu i wcześniejszymi doświadczeniami z tym języka autora tej pracy.

3.7. Docker

Docker [26] to platforma, która umożliwia konteneryzację programów, usług, aplikacji, które umożliwia ich rozwijanie w izolowanym środowisku. Docker oferuje zestaw produktów PaaS (Platform as a Service), który to zestaw usług skierowanych do programistów, które pozwala rozwijać oprogramowanie bez konieczności zajmowania się infrastrukturą, wirtualizacją, doborem zasobów. PaaS działa z wykorzystaniem IaaS (Infrastructure as a Service), z tym że sam automatycznie zarządza wszystkimi koniecznymi aspektami infrastruktury rozwoju aplikacji [25]. Zdecydowaną i największą zaletą korzystania z Dockera jest to, że aplikacje mogą być rozwijane bez potrzeby programisty zajmowania się wirtualizacją, a samo rozwiązanie jest przenośne, ponieważ kontener Dockera jest już odpowiednio skonfigurowany, dzięki czemu jesteśmy w stanie odtworzyć środowisko deweloperskie w wiele łatwiejszy sposób. Kolejną ważną zaletą Dockera jest jego małe zużycie zasobów komputera. Docker korzysta bezpośrednio z jądra systemu operacyjnego, na którym jest zainstalowany, aby uruchamiać aplikacje, rozwiązanie to eliminuje potrzebę wirtualizacji całego systemu operacyjnego, na którym aplikacja jest uruchomiona. [26]

W projekcie użycie Dockera zdecydowanie przyspieszyło rozwój tworzenia aplikacji potrzebnego na wykonanie niniejszej pracy. Dzięki gotowym obrazom z preinstalowanym oprogramowaniem, które są dostępne w „Docker Hub” [27] bez konieczności zamartwiania się instalacją środowiska z technologiami koniecznymi do wykonania projektu do tego projektu, udało się pozyskać obrazy dla Pythona [28] i MongoDB [29]. Po pobraniu i

uruchomieniu kontenera z bazą danych. natychmiastowo był dostęp do bazy poprzez MongoDb Compass [30] i kod w Pythonie do komunikacji z bazą danych.

3.8. MongoDB

MongoDB [53] to otwartoźródłowa, nierelacyjna baza danych, która w przeciwieństwie to baz relacyjnych, zamiast korzystać z tabel, korzysta z modelu opartego na dokumentach. MongoDB przechowuje dane w elastycznych dokumentach w formacie JSON (JavaScript Object Notation), który może zawierać różne rodzaje danych. Zdecydowaną zaletą tej bazy jest możliwość tworzenia zagnieżdżeń i tablic jako zawartość, pozwala to tworzyć bardzo elastyczne rozwiązania.

MongoDB nie wymaga żadnego schematu, co programiście daje ogromną swobodę w sposobie tworzenia i pozyskiwania dokumentów.

3.9. RESTful API

RESTful (Representational State Transfer) to architektoniczny standard projektowania interfejsów komunikacyjnych dla systemów informatycznych opartych na REST API (*ang. Application Programming Interface*) *interfejs programowania aplikacji*. API stanowi zestaw zdefiniowanych protokołów i narzędzi umożliwiających efektywną komunikację między różnymi elementami architektury systemów informatycznych, takimi jak serwery, klienty i bazy danych.

W roku 2008 Leonard Richardson zaproponował 3-warstwowy model dojrzałości API [33]. Jest to zbiór narzędzi służących do oceny architektury interfejsów API. Model ten prezentuje hierarchię dojrzałości API podzieloną na trzy poziomy (i poziom zerowy), gdzie każdy wyższy poziom, zawiera wszystkie cechy poziomów niższych [34]:

- Poziom 0: Swamp of POX — na tym poziomie komunikacja z API odbywa się przez HTTP i zawiera niezaadresowane zasoby. Oznacza to, że każda operacja, wymaga specjalnej instrukcji przesłanej w żądaniu. Na przykład: http://example.com

- Poziom 1: Resources — główny koncept architektury REST. Na tym poziomie, API zaczyna korzystać z wielu URI, każdy z zasobów dostaje własne URI. Na przykład: http://example.com/item/31
- Poziom 2: HTTP Verbs — na tym poziomie API zaczyna korzystać różnych metod HTTP takich jak: POST, GET, PUT, DELETE itp, aby manipulować zasobami. Dzięki tym metodom interakcje z API są łatwiejsze do zrozumienia i interpretacji.
- Poziom 3: Hypermedia Controls — ostatnia warstwa dojrzałości mówi o tym, że odpowiedzi z serwera nie tylko zawierają odpowiedzi na dane żądanie, ale również informacje o tym, jakie akcje są możliwe na danym zasobie.

W kontekście tej pracy API pełni kluczową rolę w zapewnieniu niezawodnego i efektywnego mechanizmu komunikacji między warstwą *frontendową* (*pol. interfejsu*) a *backendem* (*pol. zaplecza*). Dzięki temu rozwiązaniu przez interfejs można wysyłać asynchroniczne żądania do backendu, co umożliwia dynamiczne i responsywne doświadczenie użytkownika. Jest to kontrast w stosunku do architektury *SSR (ang. Server Side Rendering) renderowanie po stronie serwera*, gdzie cała strona, już z wygenerowanymi danymi, jest przesyłana od serwera do klienta, co może wpłynąć negatywnie na wydajność serwera przy wielu zapytaniach, a cała wygenerowana strona, ma znacznie większy rozmiar, co negatywnie wpływa na wydajność w warunkach z pogorszonym dostępem do internetu.

Do głównych zalet korzystania z architektury REST API można zaliczyć [34]:

- **Bezstanowość**: Każde żądanie od klienta do serwera jest samoobsługowe i zawiera wszystkie informacje niezbędne do jego przetworzenia. Oznacza to, że serwer nie przechowuje informacji między poszczególnymi żądaniami, co zwiększa odporność systemu na błędy i ułatwia zarówno skalowalność, jak i rozwój nowych funkcjonalności.
- **Uniwersalność i Szerokie Zastosowanie**: Dzięki wykorzystaniu standardowych protokołów HTTP, RESTful API można łatwo implementować w różnych kontekstach — od aplikacji internetowych, przez mobilne, aż po mikrousługi i złożone ekosystemy aplikacyjne.
- **Modularność i Rozszerzalność**: RESTful API ułatwia rozbicie systemu na mniejsze, niezależne moduły, co sprzyja lepszej organizacji kodu i ułatwia wprowadzanie zmian czy rozwijanie nowych funkcji.
- **Efektywność**: Użycie RESTful API pozwala na odciążenie serwera, skoro to klient jest odpowiedzialny za prezentację danych. To pozwala na zwiększenie efektywności i wydajności systemu.

Część praktyczna

4.1. Wymagania funkcjonalne

Głównym założeniem projektu jest ogólnopojęta prostota użytkowania niezależnie od wieku i biegłości w użytkowaniu urządzeń z dostępem do internetu. Wychodząc od tego głównego założenia powstały elementy i punkty styczności użytkownika z serwisem, które wymagają uwagi:

- System powinien umożliwić wyszukanie interakcji między lekami w minimalizujący popełnienie błędów sposób.
- System powinien gwarantować łatwy do zrozumienia interfejs użytkownika.
- System powinien ograniczać ilość możliwych błędów do popełnienia.
- System powinien posiadać możliwość utworzenia w nim konta.

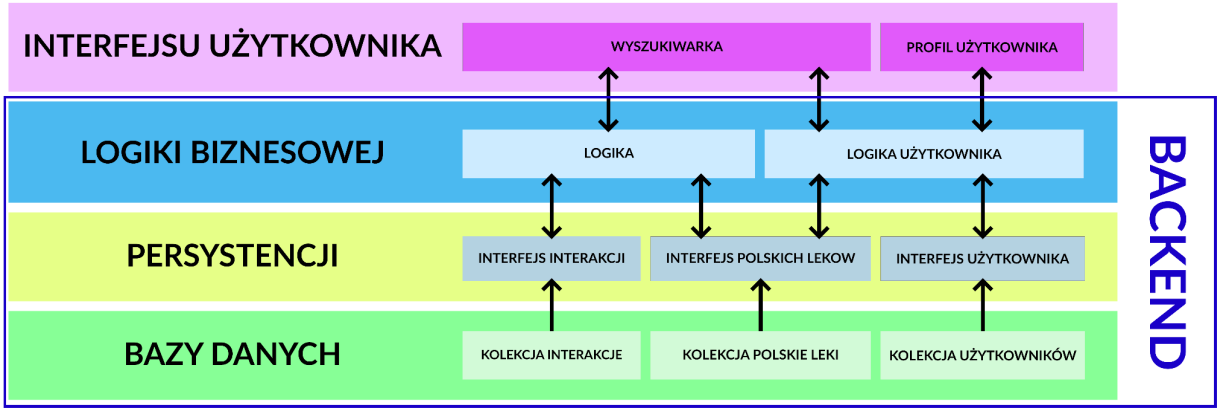
4.2. Wymagania niefunkcjonalne

- Strona musi być łatwa w obsłudze i przejrzysta. Wszystkie elementy na stronie muszą mieć jasno określone intencje, by nie wprowadzać w błąd użytkownika.
- System musi być oparty na najnowszych, wspieranych technologiach webowych by umożliwiających korzystanie na jak najszerszej puli urządzeń.
- Aplikacja ma być wieloplatformowa, wygoda użytku na komputerze i na urządzeniach mobilnych.
- Aplikacja ma mieć niskie wymagania sprzętowe, aby zapewnić szybkość działania w nowszych i starszych urządzeniach.
- Aplikacja musi być łatwa w rozbudowie, tak aby w przyszłości bez problemu można było dodawać nowe funkcjonalności bez przerabiania lub zmieniania już istniejących.
- Kod napisany czytelnie, według zasad „clean code” [32]. Czyli w sposób zorganizowany, sformatowany, łatwo zrozumiały (na przykład poprzez nazywanie zmiennych zgodnie z ich przeznaczeniem, nieużywanie skrótów).

4.3. Struktura projektu

Projekt do niezbędnego działania wykorzystuje 4 warstwy: interfejsu użytkownika, logiki biznesowej, persystencji i bazy danych. Każda z nich zawiera w sobie odpowiednie, niezależne od siebie komponenty, schemat struktury przedstawiony jest na Rys. 22. Dzięki warstwowej strukturze projektu, w przypadku gdyby nastąpiła zmiana w sposobie działania systemu, na przykład zmiana używanej bazy danych, wystarczy zmienić warstwę persystencji, by dopasować do obecnych wymagań, reszta warstw pozostaje niezmieniona.

Warstwy logiki biznesowej, persystencji i bazy danych wchodzą w skład grupy nazywanej backendem. Grupa ta odpowiada za funkcjonalność systemu, za przetwarzanie danych, za ich odbiór, przekazanie i zapis. Backend w przeciwieństwie do frontendu nie jest widoczny dla użytkownika.



Rys. 22. Schemat projektu i przepływu danych, opracowanie własne.

Pierwsza warstwa: Interfejsu użytkownika — W warstwie tej znajdują się komponenty, za pomocą których użytkownik końcowy będzie w stanie ingerować z całym systemem. Każdy komponent znajdujący się w niej komunikuje się bezpośrednio tylko z warstwą logiki biznesowej.

Możliwości działania komponentu “wyszukiwarka”:

- Przesłanie zapytania o listę leków zaczynających się od wpisanej treści, a następnie jej wyświetlenie.
- Wyświetlenie listy wybranych leków w tym leków zalogowanego użytkownika.
- Zapytanie, czy zalogowany użytkownik ma zapisane leki na swoim koncie
- Wysłanie żądania o przeszukanie interakcji pomiędzy wybranymi lekami.

- Wyświetlanie wyników interakcji.

Możliwości działania komponentu “profil użytkownika”:

- Wysłanie zapytania z danymi do utworzenia konta.
- Wysłanie zapytania z danymi do zalogowania użytkownika.
- Przesłanie zapytania o listę leków zaczynających się od wpisanej treści, a następnie jej wyświetlenie.
- Wyświetlenie listy wybranych leków.
- Wysłanie zapytania z danymi do zapisania leków przyjmowanych.
- Wysłanie zapytania z danymi o usunięcie konta.

Druga warstwa: Logika biznesowa — W tej warstwie znajdują się komponenty odpowiedzialne za funkcjonalność kluczową do działania systemu. Logika biznesowa stanowi warstwę odpowiadającą za przetwarzanie dostarczonych do niej danych oraz następnie przesyłanie ich dalej.

Możliwości działania komponentu “logiki”:

- Przyjęcie danych tekstowych z wyszukiwarki i wykonanie zapytania przez interfejs polskich leków w celu zwrócenia listy dostępnych leków, a następnie zwrócenie odpowiedzi ponownie do wyszukiwarki.
- Przyjęcie listy wybranych leków z wyszukiwarki, a następnie wysłanie zapytania przez interfejs polskich leków z prośbą o zwrócenie składników leków po angielsku, a następnie wysłanie składników do interfejsu interakcji z prośbą o zwrócenie interakcji pomiędzy nimi, a następnie zwrócenie wyników interakcji do wyszukiwarki.

Możliwości działania komponentu “logika użytkownika”:

- Przyjęcie danych użytkownika z komponentu profilu użytkownika w celu jego rejestracji, wysłanie tych danych, przez interfejs użytkownika, ewentualne zwrócenie błędu w przypadku błędu rejestracji do komponentu profilu użytkownika.
- Przyjęcie danych z komponentu profilu użytkownika w celu zalogowania, ewentualne zwrócenie błędu w przypadku błędu logowania do komponentu profilu użytkownika.
- Przyjęcie danych tekstowych z komponentu profilu użytkownika i wykonanie zapytania przez interfejs polskich leków w celu zwrócenia listy dostępnych

leków, a następnie zwrócenie odpowiedzi do komponentu profilu użytkownika.

- Wysłanie zapytania przez interfejs użytkownika w celu zapisu danych, na temat przyjmowanych leków, przyjętych z profilu użytkownika.
- Odbiór zapytania o zalogowanego użytkownika i przez zapytanie przez interfejs użytkownika uzyskanie i zwrócenie danych na temat przyjmowanych leków zalogowanego użytkownika.

Trzecia warstwa: Persystencji — W tej warstwie znajdują się komponenty, które odpowiadają za umożliwienie komunikacji z bazą danych. Dzięki separacji logiki interakcji z bazą danych zwiększa elastyczność systemu.

Możliwości działania "interfejs interakcji":

- Przyjęcie danych z logiki i wyszukanie w bazie danych w kolekcji interakcji, interakcji pomiędzy składnikami.

Możliwości działania "interfejs Polskich leków":

- Przyjęcie danych tekstowych z logiki lub logiki użytkownika i zwrócenie wyniku zapytania do kolekcji polskich leków, zwracając leki, które zaczynają się na podany ciąg znaków.
- Przyjęcie listy leków z logiki i wykonanie zapytania do kolekcji polskich leków w celu zwrócenia angielskich nazw składników leków przesłanych jako lista.

Możliwości działania "interfejs użytkownika":

- Przyjęcie danych z logiki użytkownika i stworzenie zapytania do kolekcji użytkowników w celu rejestracji użytkownika, w przypadku błędu lub sukcesu jest zwracana adekwatna wiadomość do logiki użytkownika
- Przyjęcie danych z logiki użytkownika i stworzenie zapytania do kolekcji użytkowników w celu logowania użytkownika, w przypadku błędu lub sukcesu jest zwracana adekwatna wiadomość do logiki użytkownika
- Przyjęcie danych z logiki użytkownika i wykonanie zapytania do kolekcji użytkowników w celu zapisu przyjmowanych leków do odpowiedniego użytkownika

Czwarta warstwa: Bazy danych — Warstwa ta stanowi fizyczne środowisko do przechowywania danych. Jej rola skupia się na gromadzeniu, organizacji i udostępnianiu danych. Warstwa ta ściśle powiązana jest z warstwą poprzednią.

4.4. Projekt interfejsu użytkownika

W kontekście tego projektu koncepcja Interfejsu Użytkownika (UI) jest skoncentrowana na trzech kluczowych parametrach: prostocie, intuicyjności oraz responsywności.

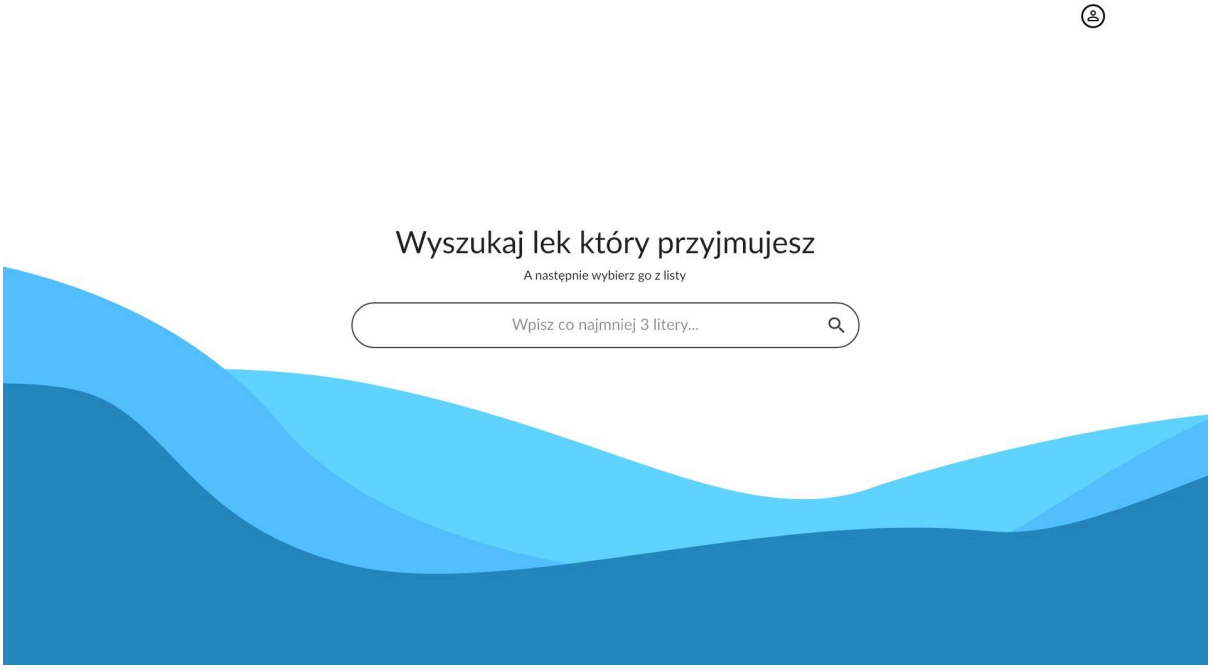
Pierwszym celem jest zaprojektowanie interfejsu, który jest prosty i niewymagający dla użytkownika. Prostota w tym przypadku odnosi się do zminimalizowania liczby kroków potrzebnych do wykonania zamierzonych działań w aplikacji, co czyni jej użycie bezproblemowym nawet dla osób nieobeznanych z nowoczesnymi technologiami.

Drugi aspekt, intuicyjność, zapewnia, że elementy nawigacyjne i funkcjonalne są łatwo zrozumiałe i przewidywalne. Celem jest eliminacja potrzeby czytania dokumentacji lub uzyskania pomocy z zewnątrz, użytkownik powinien być w stanie naturalnie i łatwo poruszać się po aplikacji.

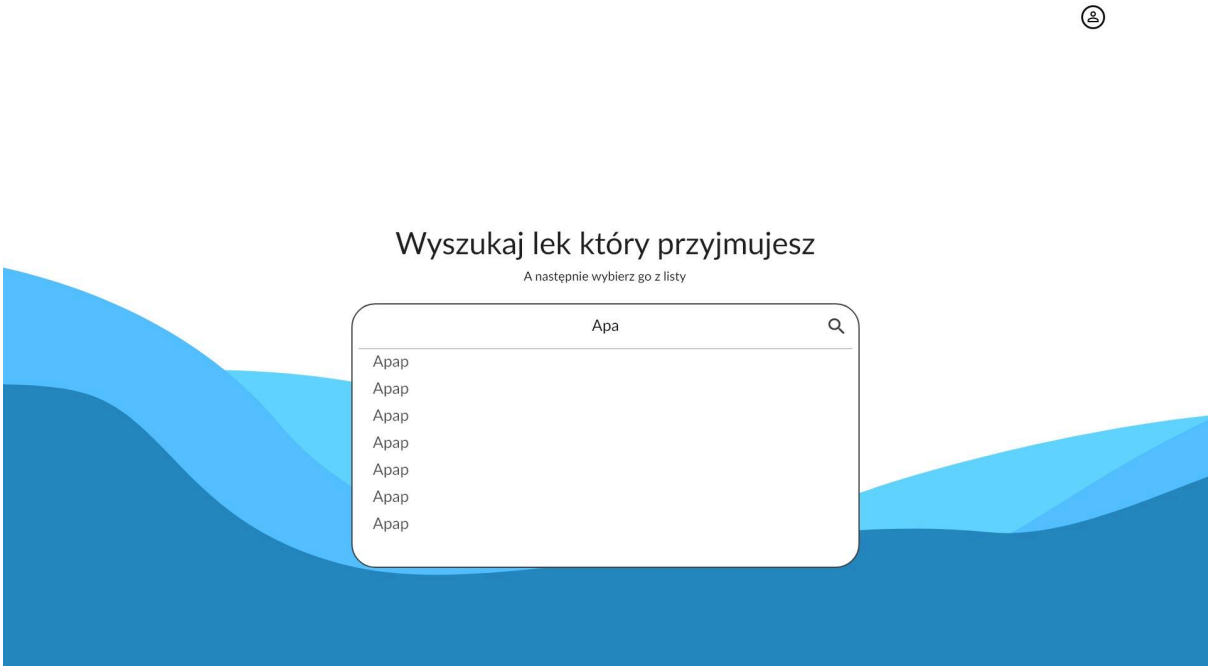
Trzeci element, responsywność, odnosi się do zdolności interfejsu do adaptacji do różnych rozdzielczości i wielkości ekranu. Dzięki zastosowaniu responsywnego designu aplikacja może być używana na wielu urządzeniach, od smartfonów i tabletów po komputery stacjonarne, bez utraty funkcjonalności czy estetyki.

Interfejs zostanie zrealizowany w technologii Vue.js [36] z wykorzystaniem biblioteki Vuetify [20] co pozwoli na uzyskanie jednolitego, estetycznego wyglądu. Wspomniana biblioteka nie tylko zapewni konsekwentność w wyglądzie, ale również przyspieszy tworzenie aplikacji.

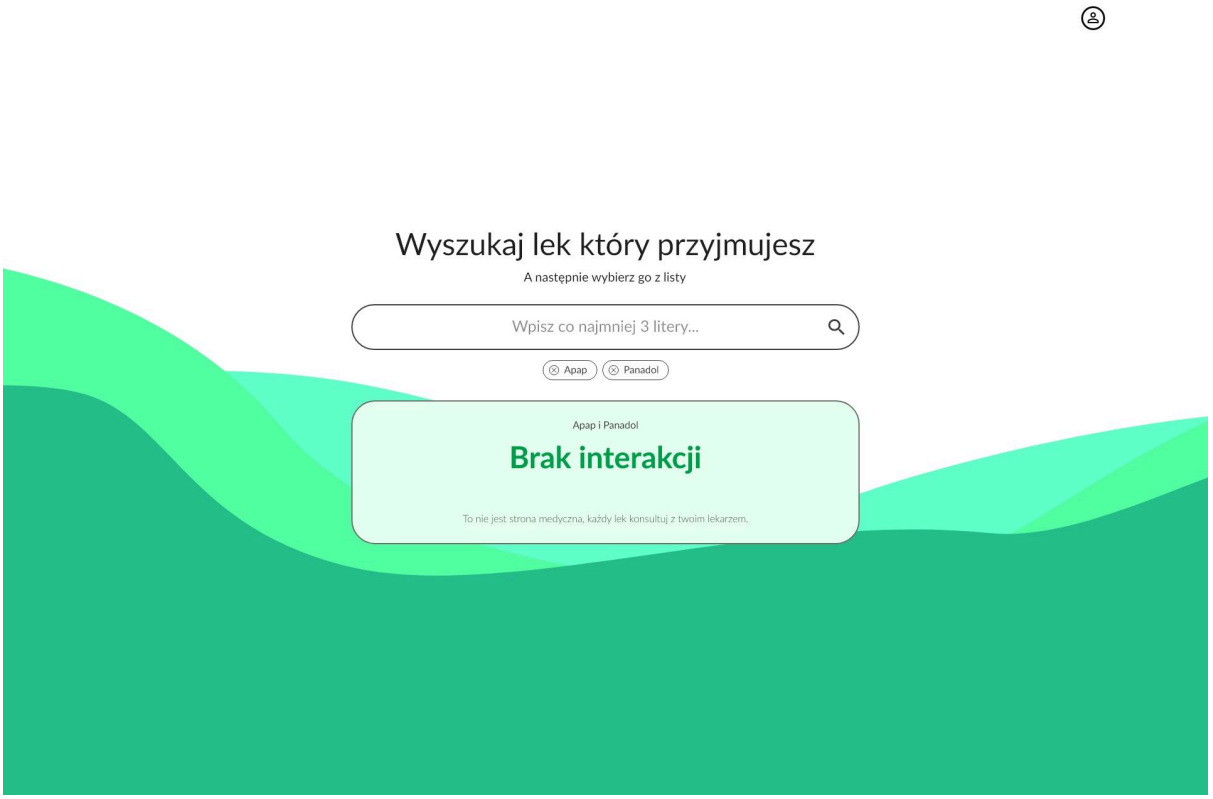
Na Rysunkach od 23 do 27 znajdują się projekty UI wykonane w Adobe Xd. Przedstawione są tylko widoki komputerowe dla zwiększenia czytelności tego opracowania. Kolejno są to: Strona startowa, Wybrane pole wpisywania z sugestiami leków, porównanie dwóch leków — brak interakcji, porównanie dwóch leków — interakcja, porównanie dwóch leków — poważna interakcja.



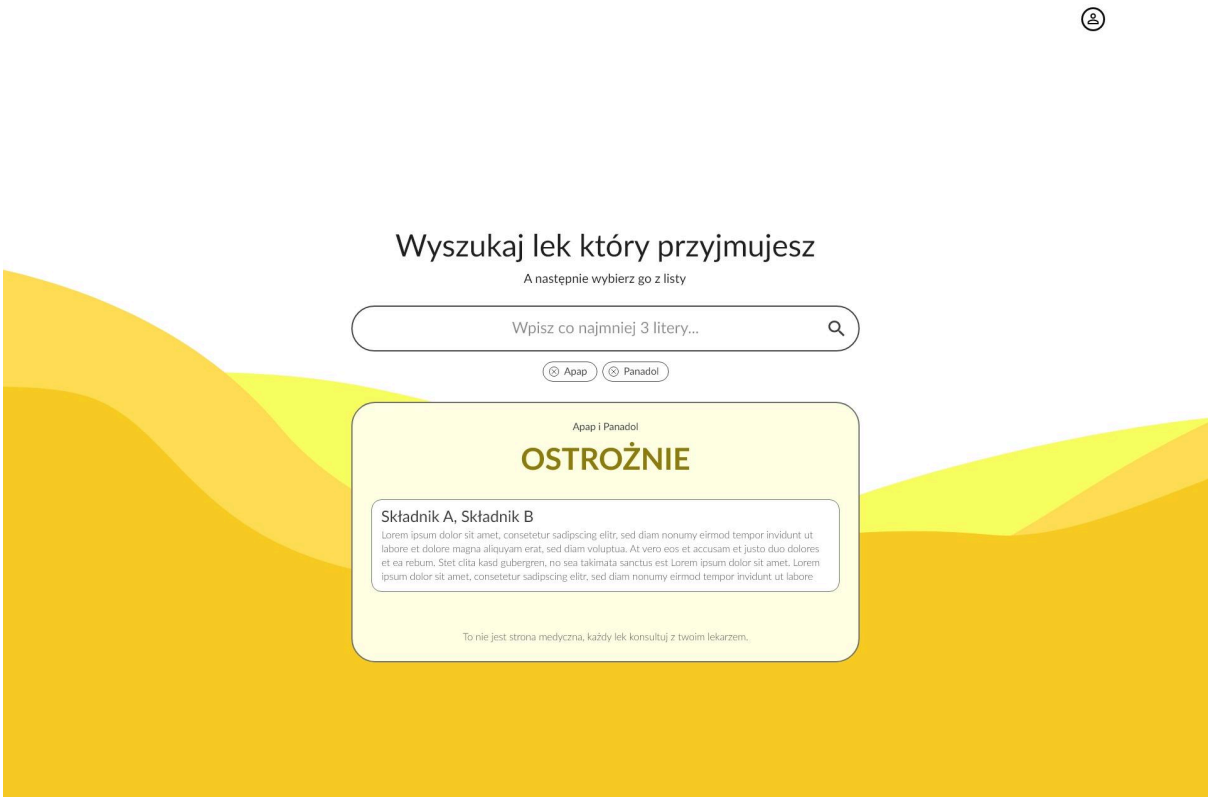
Rys. 23 Projekt strony głównej na komputery stacjonarne, opracowanie własne.



Rys. 24. Projekt strony głównej — wpisywanie leku, opracowanie własne.



Rys. 25. Porównanie dwóch leków — brak interakcji, opracowanie własne.



Rys. 26. Porównanie dwóch leków — interakcja, opracowanie własne.

Rys. 27. Porównanie dwóch leków — poważna interakcja, opracowanie własne.

4.5. Analiza zebranych danych

4.5.1. DrugBank

DrugBank [35] to zaawansowana platforma internetowa, która stanowi jedno z największych i najbardziej wszechstronnych źródeł informacji o składnikach farmaceutycznych. Oferowana baza danych jest nie tylko rozbudowana, ale również precyzyjnie skatalogowana i zawiera szeroki zakres informacji naukowych. Dzięki temu stanowi nieocenione narzędzie dla różnych grup profesjonalistów, w tym naukowców, lekarzy, farmaceutów i innych specjalistów w dziedzinie zdrowia.

W bazie można znaleźć różnorodne dane takie jak: nazwa chemiczna składnika, jego struktura molekularna, gatunek i kategoria, mechanizmy działania, a także informacje o potencjalnych interakcjach z innymi substancjami – nie tylko lekami, ale również żywnością i alkoholem. Co więcej, baza oferuje dostęp do powiązanych publikacji naukowych, co pozwala na głębsze zrozumienie kontekstu i implikacji związanych z danym składnikiem.

DrugBank wykazuje również dużą otwartość w kierunku środowiska akademickiego.

Oferują oni dostęp do swojego zasobu na preferencyjnych warunkach dla uczniów i studentów, co znacząco wspiera badania i rozwój w dziedzinie medycyny i farmakologii. Na chwilę obecną, baza zawiera 17,386 zaktualizowanych rekordów, co sprawia, że jest jednym z największych tego typu zasobów na świecie.

W kontekście niniejszej pracy baza danych DrugBank okazała się być nieocenionym źródłem informacji. Dostępność danych w formacie XML znacznie ułatwiła ich analizę, przetwarzanie i integrację z naszym systemem. Format ten jest nie tylko łatwy w obszarze, ale również umożliwia efektywną automatyzację procesów, co jest kluczowe dla skalowalności i funkcjonalności projektowanego systemu.

4.5.2. Polski zbiór leków

Kluczową rolę w procesie realizacji niniejszej pracy odgrywa baza danych o lekach dostępnych na rynku polskim. Z powodu braku udostępnionej powyższej bazy przez żadną stronę trzecią lub organ państwowy zaszła potrzeba pozyskania informacji za pomocą webscrapingu ze stron internetowych, które takie dane oferują.

Do tego celu wybrano stronę https://www.mp.pl/pacjent/leki/, która oferuje zbiór polskich leków w zorganizowanej liście wraz z opisem i jego składnikami. Do pozyskania danych z wybranej strony wykorzystano bibliotekę Selenium w programie napisanym w języku Python, który zamieszczono w dodatku A. Selenium pozwala na automatyzację sterowania przeglądarką internetową, dzięki której przy odpowiedniej wiedzy na temat struktury strony, w łatwy sposób można pozyskać dane, które nas interesują.

Problemem, który się pojawił jest brak jedności w nazwach składników leków pomiędzy bazą interakcji z Drugbank a strony mp.pl. Podczas gdy w bazie interakcji nazwy składników podane były po angielsku, analogicznie na Polskiej stronie podane one były po polsku. By uporać się z problemem, wykorzystano Google Cloud Translation API, które podczas testów manualnych wykazało rzetelność i poprawność w tłumaczeniu nazw z Polskiego na Angielski. API oferowane przez Google oferuje darmowe tłumaczenie do 500000 znaków miesięcznie [31] co jest zupełnie wystarczające na nasze potrzeby.

Algorytm pozyskania danych ze strony mp.pl

1. Zgodnie z alfabetem i cyframi przejście na strony z listą leków zaczynających się na daną literę.
2. Pozyskanie wszystkich linków z lekami, które znajdują się na stronie.
3. Przejście na następną podstronę danej litery
4. Po przejściu wszystkich stron zapisanie każdego leku w formacie JSON. Struktura tak zapisanego leku znajduje się na Rys. 28.

```
index: {
    "drug_name_pl": nazwa_leku_na_stronie,
    "url":link_do_leku_ze_strony,
    "ingredients_pl": {},
    "ingredients_en": {},
}
```

Rys. 28. Struktura w pliku JSON zapisanego leku ze strony mp.pl.

5. Po wykonaniu listy, odczytanie pierwszego elementu, przejście pod jego "url"
6. Po wejściu na stronie zlokalizowanie nazwy składniku (składników)
7. Odczytanie tekstu składniku (składników)
8. Ewentualne rozdzielenie wielu składników
9. Przetłumaczenie składniku (składników) na angielski
10. Zapisanie składniku (składników) do `ingredients_pl` i `ingredients_en`
11. Powtórzenie dla każdego elementu w liście kroków 6-10

4.6. Implementacja backendu

4.6.1. Przetworzenie danych do odpowiedniego formatu

Po pozyskaniu danych kluczowych służących do prawidłowego działania niniejszego projektu potrzebujemy przetworzyć dane do bazy danych. Najlepszy wyborem bazy danych do tego celu będzie MongoDB.

Ponieważ wyżej wymieniona baza do przechowywania danych używa formatu JSON, pierwszym krokiem będzie przetworzenie formatu XML i wyabstrahować jedynie interesujące nas dane, które są nam niezbędne do realizacji projektu.

Założenia i obserwacje przed parsowaniem: Plik XML ma przestrzeń nazw „ns0:" co oznacza, że podczas parsowania pliku będziemy musieli to uwzględnić. Każda encja składnika zawiera wiele ID, lecz jedyne, które będzie nam potrzebne to id z atrybutem „primary", czyli głównym. Struktura uzyskanych przez nas w ten sposób danych znajduje się na Rys. 29.

```
{
    id_leku: {
        "name": nazwa_leku,
        "drug_interactions": {
            id_interakcji: {
                "name": nazwa_leku_wchodzacego_w_interakcje,
                "description": opis_interakcji,
            }},
    },
}
```

Rys. 29. Format wyodrębnionych danych z pliku XML.

Do uzyskania rezultatu będziemy potrzebować dwóch bibliotek dostępnych w Pythonie, xml i json. Z biblioteki xml potrzebować będziemy modułu ElementTree, który importujemy pod skrótem ET. Zakładamy, że biblioteki są już zainstalowane na systemie. Program wykonujący to zadanie znajduje się na Rys. 30.

```
import xml.etree.ElementTree as ET
import json

def extract_drug_data_from_xml(xml_path):
    # Load the XML file
    tree = ET.parse(xml_path)
```

```
    root = tree.getroot()

    # Define namespace
    namespace = {'ns0': 'http://www.drugbank.ca'}

    # Extract data for all drugs
    all_drugs_data = {}
    for drug in root.findall('ns0:drug', namespace):

        # Extract drug ID
        drug_id = drug.find('ns0:drugbank-id[@primary="true"]',
namespace).text

        # Extract drug name
        drug_name = drug.find('ns0:name', namespace).text

        # Extract all drug interactions
        drug_interactions = {}
        for interaction in
drug.findall('ns0:drug-interactions/ns0:drug-interaction',
namespace):
            interaction_id = interaction.find('ns0:drugbank-id',
namespace).text
            interaction_name = interaction.find('ns0:name',
namespace).text
            interaction_description =
interaction.find('ns0:description', namespace).text
            drug_interactions[interaction_id] = {
                "name": interaction_name,
                "description": interaction_description
            }
```

```
        # Combine data
        all_drugs_data[drug_id] = {
            "name": drug_name,
            "drug-interactions": drug_interactions
        }

    return all_drugs_data

# Usage and save to json:
data = extract_drug_data_from_xml('full database.xml')
with open('output.json', 'w', encoding='utf-8') as json_file:
    json.dump(data, json_file, ensure_ascii=False, indent=4)
```

Rys. 30. Implementacja programu do wyodrębnienia danych z pliku XML i zapis do pliku JSON.

Przetworzone dane do formatu JSON przenosimy do bazy danych. Po odczytaniu pliku i połączeniu się z bazą dodajemy dane za pomocą kodu przedstawionego na Rys. 31.

```
documents = []
for drug_id, details in data.items():
    document = {
        "drug_id": drug_id,
        "drug_name": details["name"],
        "drug_interactions": [
            {
                "interaction_id": int_id,
                "interaction_name": int_details["name"],
                "interaction_description": int_details["description"],
            }
            for int_id, int_details in details["drug-interactions"].items()
        ],
    }
    documents.append(document)

drugs_collection.insert_many(documents)
```

Rys. 31. Kod i struktura dodawanych leków do bazy danych.

Następnie plik JSON pozyskanego z webscrapingu Polskiej bazy danych leków odczytujemy, sortujemy według kluczy i wgrywamy również do bazy.

```
with open("json_data.json", "r", encoding="utf-8") as f:
    data = json.load(f)

sorted_data = {k: data[k] for k in sorted(data.keys())}

polish_drugs_collection.insert_many(sorted_data.values())
```

Rys. 32. Kod służący dodaniu pliku JSON do bazy danych.

Z tak utworzoną bazą danych, możemy zająć się tworzeniem logiki obsługującej niniejszą bazę.

4.6.2. Utworzenie API,

W ramach procesu inżynieryjnego kolejnym etapem rozwoju aplikacji jest implementacja warstwy abstrakcji dla komunikacji z bazą danych. Ten komponent jest niezbędny do efektywnej interakcji z danymi, które będą używane w aplikacji, zwłaszcza w kontekście zapewnienia wsparcia dla API. W tym celu zostały zaprojektowane dwie klasy w języku Python, z wykorzystaniem biblioteki pymongo [54], która umożliwia interakcję z bazą danych MongoDB.

Na Rys. 33 znajduje się kod, który przedstawia inicjalizację klasy, służącej następnie do łączenia się z bazą danych i jedną metodę do pozyskania składnika leku przez jego nazwę za pomocą wyrażeń rozmytych.

```
import pymongo
import json
from fastapi.encoders import jsonable_encoder

class dbInterface:
    def __init__(self):
```

```
        self.client =
pymongo.MongoClient("mongodb://host.docker.internal:27017/")
        self.db = self.client["drug_database"]
        self.collection = self.db["drugs"]

    def getMedicineByName(self, name):
        query = {"drug_name": {"$regex": name, "$options": "i"}}
        result = self.collection.find_one(query)
        return json.dumps(result, default=str, ensure_ascii=True)
```

Rys. 33. Klasa interfejsu do komunikacji z kolekcją interakcji w bazie danych.

Ponieważ rozwiązanie backendowe i bazodanowe jest uruchomione w kontenerze Dockera, potrzebne było wykorzystanie specjalnego adresu, który Docker interpretuje jako adres hosta kontenera.

Analogicznie do powyższego rozwiązania zaimplementowano klasę, która dostarcza szereg metod do obsługi kolekcji z polskim zbiorem leków. Najważniejszą metodą, która umożliwia nam przekazanie tablicy nazw wielu polskich leków, a następnie otrzymania składników w języku angielskim potrzebnych do wyszukania interakcji pomiędzy nimi w bazie interakcji jest przedstawiona na Rys. 34.

```
    def fromManyGetEnIngredientsOnly(self, names):
        query = {"drug_name_pl": {"$in": names}}
        result = self.collection.find(query)
        returnedDict = {}
        for result in result:
            if "ingredients_en" not in result:
                continue
            returnedDict[result["drug_name_pl"]] = result["ingredients_en"]

         extracted_values = [value for item in returnedDict.values() for value
in item.values()]

        extracted_values = [x.split(" (")[0] for x in extracted_values]
        return extracted_values
```

Rys. 34. Metoda w klasie interfejsu do interakcji z kolekcją polskich leków.

Następnie dzięki uzyskanej tablicy leków, jesteśmy w stanie stworzyć wyszukanie interakcji pomiędzy lekami. Punkt dostępu utworzony w api zamieszczony jest na Rys. 35.

```
@app.get("/get_pl_drugs_names_to_interactions/{drug_names}")
def get_pl_drugs_names_to_interactions(drug_names: str):
    drug_names = drug_names.split(",")
    result = polish_drug_db.fromManyGetEnIngredientsOnly(drug_names)

    interactions = []
    for drug_name in drug_names:

interactions.append(interaction_db.getManyInteractionsFromOneName(drug_name,
result))

    return {"data": interactions}
```

Rys. 35. Utworzony punkt dostępu w API dla wyszukania interakcji pomiędzy składnikami.

Natomiast funkcja w interfejsie pobierająca dane z bazy danych, przedstawiona jest na Rys. 36.

```
def getManyInteractionsFromOneName(self, name, namesArray):
        query = {"drug_name": {"$regex": name, "$options": "i"},
"drug_interactions.interaction_name": {"$in": namesArray}}
        projection = {"drug_interactions": {"$elemMatch": {"interaction_name":
{"$in": namesArray}}}}

        results =  self.collection.find(query, projection)
        returnedDict = {}
        for result in results:
            print(result)
            returnedDict["interaction_id"] =
result["drug_interactions"][0]["interaction_id"]
            returnedDict["interaction_name"] =
result["drug_interactions"][0]["interaction_name"]
            returnedDict["interaction_description"] =
result["drug_interactions"][0]["interaction_description"]

        return returnedDict
```

Rys. 36. Pobrane danych o interakcjach za pomocą zapytania do bazy danych.

Punkt dostępowy na Rys. 34 został zoptymalizowany pod względem szybkości działania. W formie pierwotnej powyższego rozwiązania podczas testów zauważono długi (powyżej 10 sekund) czas odpowiedzi na zapytanie. Pierwotna wersja zapytania, znajdująca się na Rys. 37, wykorzystywała podwójną pętlę, ponieważ w interfejsie utworzona była tylko funkcja, która obsługiwała jedynie porównanie dwóch składników jednocześnie.

```
def get_pl_drugs_names_to_interactions(drug_names: str):
    drug_names = drug_names.split(",")
    result = polish_drug_db.fromManyGetEnIngredientsOnly(drug_names)

    interactions = {}
    for i in range(len(result)):
        for j in range(i + 1, len(result)):
            interaction = interaction_db.getInteractionBetweenNames(result[i],
result[j])
            if interaction:
                interactions[result[i]] = interaction



    return {"data": interactions}
```

Rys. 37. Pierwotna wersja zapytania do API interakcji lekowych.

Jak widać, dzięki rozłożeniu odpowiedzialności pomiędzy różne komponenty otrzymujemy czytelny kod, który w bardzo łatwy sposób interpretować i wykorzystać przy działaniu z nim.

Dodatkowo by użytkownik nie miał problemów z wyborem leku, dodano funkcję, która na bieżąco podpowiada użytkownikowi wyniki wyszukiwania, które zaczynają się od wpisanej przez niego frazy.

```
    def getMedicinesWithNameStartingWith(self, name):
        query = {"drug_name_pl": {"$regex": "^" + name, "$options": "i"}}
        queryResult = self.collection.find(query).limit(8)
        returnedDict = {}

        for index, result in enumerate(queryResult):
            returnedDict[index] = result["drug_name_pl"]
```

```
        return returnedDict
```

Rys. 38. Pobranie danych z nazwami polskich leków zaczynających się na podany fragment tekstu.

Mając wszystkie powyższe metody bezproblemowo jesteśmy w stanie wykonać warstwę frontendu, która będzie odpowiedzialna za komunikację z użytkownikiem.

4.7. Implementacja frontendu

Po wykonaniu warstwy abstrakcji interakcji z bazą danych bezproblemowo możemy spełnić wszelkie założenia niezbędne do realizacji frontendu wedle ustalonego wcześniej planu. Ponieważ do wykonania niniejszej pracy wykorzystujemy bibliotekę vueitify, możemy korzystać z gotowych już komponentów, które spełniają wcześniej wypisane wymagania.

Pierwszym i głównym z nich, który będzie głównym elementem wykorzystywanym przez użytkownika, jest wyszukiwarka. Wykonana została na bazie komponentu <VAutocomplete>, który dzięki dostarczonej mu liście umożliwia na przeszukanie jej, a następnie wyboru wyników, które interesują użytkownika.

```
    <v-autocomplete
        ref="autocomplete"
        v-model="selected"
        v-model:search="search"
        :loading="loading"
        :items="items"
        no-data-text="Brak wyników"
        label="Wyszukaj"
        density="default"
        multiple
        clearable
        variant="outlined"
        auto-select-first
        :counter="5"
        :counter-value="selectedCount"
        persistent-clear
        persistent-counter
        persistent-placeholder
        placeholder="Wyszukaj"
        return-object
        chips
        closable-chips
        rounded
```

```
            @click:clear="clearResults"
            @input="getItems(search)"
          >
          </v-autocomplete>
```
Rys. 39. Element w komponencie wyszukiwarki.

Jak widać na Rys. 39, komponent przyjmuje wiele atrybutów, których nazwy są bardzo jasno zrozumiałe. Jednak na szczególną uwagę zasługuje parę z nich:

- v-model=”selected” — jak wspomniane w dziale 3.5 vue.js umożliwia natychmiastową reaktywność ze zmiennymi. V-model wskazuje na to, że wartość tego pola będzie używać modelu (zmiennej). W tym przypadku wybrane wartości z listy zapisują się do zmiennej „selected”,
- v-model:search=”search” — zmienna, która odpowiada za wpisany tekst podczas wyszukiwania w polu. Po wpisaniu co najmniej 3 znaków wysyłana jest również do backendu przez API, by następnie zwrócić listę dostępnych leków, które zaczynają się od wyszukania,
- :items=”items” — jest to atrybut, który przyjmuje listę, która następnie jest możliwa do przeszukiwania i wybierania z niej wartości. To właśnie tutaj przychodzi odpowiedź z backendu z punktu powyżej,
- @input=”getItems(search)” — jest to atrybut, który nasłuchuje na event “input” emitowany przez komponent. Na podstawie tego eventu jesteśmy w stanie wykonać metodę (funkcję). W tym przypadku jest to wysłanie wartości “Search” do backendu by zwróciła listę dostępnych leków,
- multiple, chips — oznacza to, że lista wyszukiwania jest wielokrotnego wyboru a wyniki wybrane w zmiennej “selected” pojawią się w polu wyszukiwania jako chips — komponent, który zdefiniowany jest w material design.

Kolejną ważną częścią tego projektu jest utworzenie *magazynu (ang. store)*. Ponieważ do zmiennych istniejących w komponentach, możemy jedynie odwoływać się lokalnie w danym komponencie, potrzebne jest miejsce, do którego bez problemu można zapisywać i odczytywać zmienne do wykorzystania pomiędzy wieloma komponentami. Wykorzystamy do tego bibliotekę pinia [55], która przychodzi wraz z instalacją i tworzeniem projektu Vue.JS. Przykładowa struktura takiego magazynu prezentowana jest na Rys. 40.

```
import { defineStore } from "pinia";

export const useAppStore = defineStore("app", {
 state() {
   return {
       variable1: 0,
   };
 },
 actions: {
   testFunction(value) {
       this.variable1 = value;
       console.log("Hello World!");
   }
 },
});
```

Rys. 40. Implementacja przykładowego magazynu w bibliotece pinia.

Nasz magazyn będzie nazywać się ”app” stąd też w pierwszej linijce kodu po zaimportowaniu definiujemy i eksportujemy zmienną, która wykorzystuje “defineStore” z biblioteki pinia. Magazyn wykorzystywany w naszym projekcie składa się z dwóch głównych elementów składowych: state oraz actions. Magazyn “state” działa analogicznie do “data”, które zaprezentowane było w dziale 3.6, natomiast actions, analogicznie do “methods”. Następnie, aby użyć takiego store wymagać będzie, abyśmy go użyli poprzez mapowanie jego zmiennych, akcji w miejscu, gdzie będziemy go używać. Użycie magazynu zdefiniowanego na Rys. 40 w komponencie zaprezentowane jest na Rys. 41.

```
import { mapState, mapActions } from "pinia";

export default {
 name: "test",
 computed: {
   ...mapState(useAppStore, {
     usedVariable: "variable1",
   }),
 },
 methods: {
   ...mapActions(useAppStore, [
       "testFunction",
   ])
 }
};
```

Rys. 41. Użycie przykładowego magazynu z biblioteki pinia.

Dzięki mapowaniu (j.w.) magazynu jesteśmy w stanie korzystać ze zmiennych i funkcji ze magazynu, tak jakby były one zadeklarowane w danym komponencie. Dodatkowo podczas mapowania zmiennych z magazynu jesteśmy w stanie przypisać własną nazwę, która będzie używana.

Dzięki wyżej wymienionej funkcjonalności zostały stworzone funkcje, które umożliwiają interakcję wyszukiwarki z bazą danych.

```
async getItems(name) {
      if (name.length > 3) {
        await this.getAvailableDrugs(name);
      }
    },
```

Rys. 42. Wykonanie przez store zapytania do bazy danych.

Na Rys. 42 zaprezentowana jest funkcja w komponencie, która po wpisaniu co najmniej 3 znaków w pole wyszukiwania, robi zapytanie do bazy danych, a następnie przez magazyn dodaje je do listy wyboru w wyszukiwarce. Funkcja getAvailableDrugs użyta na Rys. 42, znajduje się na Rys. 43.

```
async getAvailableDrugs(name) {
      this.loading = true;
      const url = `http://localhost:8000/drug_name_starting_with/${name}`;
      const response = await axios.get(url);
      const { data } = response.data;
      const extracted = Object.values(data);
      this.drugsAvaliable = extracted;
      this.loading = false;
    },
```

Rys. 43. Funkcja w magazynie wykonująca zapytanie do bazy danych poprzez API.

Funkcja w magazynie która wykonuje zapytanie do API w celu wyszukania interakcji między lekami, znajduje się na Rys. 44.

```
async getInteractions() {
      this.loading = true;
      this.clearResults();
      const drug_names = this.selectedDrugs.join(",");
```

```
const url =
`http://localhost:8000/get_pl_drugs_names_to_interactions/${drug_names}`;
      const response = await axios.get(url);
      const { data } = response.data;
      this.sereachResults = data;
      this.loading = false;
    },
```
Rys. 44. Funkcja w magazynie wykonująca zapytanie do bazy danych poprzez API.

Jak widać, dzięki utworzeniu API cały proces pozyskiwania jest uproszczony do maksimum. A dzięki wykorzystaniu magazynu, który dzieli zmienne pomiędzy komponentami, wynik zapisany do "searchResults" jesteśmy w bardzo łatwy sposób wykorzystać w komponencie, który odpowiada do wyświetlenia wyników. Dzięki możliwości utworzenia pętli na komponencie odbywa się to w bardzo uproszczony sposób, zaprezentowane zostało to na Rys. 45.

```
<SereachResult
        v-for="(result, key) in results"
        :key="key"
        :keyName="key"
        :item="result"
      />
```
Rys. 45. Element w komponencie służący do wyświetlenia wyników.

```
  <VCard
    :title="keyName"
    :subtitle="item.interaction_name"
    variant="outlined"
    class="mb-6 result-card mx-auto rounded-xl pa-3"
    width="500"
  >
    <v-card-text class="text-left mt-3p">
      {{ item.interaction_description }}
    </v-card-text>
  </VCard>
```
Rys. 46. Zawartość komponentu SereachResult.

4.8. Testy

Sposobem, który został wybrany do wykonania testów, jest pomiar szybkości odpowiedzi na zapytania wykonanych do API aplikacji. Przez fakt, że renderowanie wyników po stronie interfejsu użytkownika w dużej mierze jest zależne od szybkości działania urządzenia klienta, przeprowadzono pomiar przez kod napisany w języku Python który znajduje się w Dodatku B. Do testów wykorzystano dwa punkty API, do sprawdzenia interakcji między wieloma lekami i zwrócenia nazw leków zaczynających się na losowy ciąg znaków.

Pierwszy punkt dostępu został wybrany ze względu na złożoność zapytania, ponieważ wymaga on interakcji przez dwa interfejsy do komunikacji z bazą. Ponieważ obiekt zwracany z zapytania przez bibliotekę requests [56], zawiera czas pomiędzy wysłanym zapytaniem a otrzymanym we właściwości „elapsed", jesteśmy w łatwy sposób zmierzyć czasy. Test polegał na wykonaniu 100 zapytań z losowo wygenerowanymi lekami i zwróceniu średniego czasu zapytania, maksymalnego czasu i minimalnego. Kod do testów umieszczony został w dodatku B, natomiast wyniki pierwszego testu, podane w sekundach, umieszczone zostały na Rys. 47.

```
Average time of 100 requests to interactions : 0.1691366
Max time of 100 requests to interactions     : 0.55311
Min time of 100 requests to interactions     : 0.004545
Standard deviation                           : 0.12799113333333334
```

Rys. 47. Wyniki testów pomiaru szybkości zapytania.

Drugi punkt dostępu został wybrany przez częstotliwość użytkownika, ponieważ gdy użytkownik wpisze co najmniej 3 litery, każda następna litera wykona następne zapytanie do bazy danych. Test polegał na wykonaniu 1000 zapytań z losowo wygenerowanymi czterema literami i zwróceniu czasu średniego, maksymalnego i minimalnego. Kod do testów umieszczony jest w dodatku B, natomiast wyniki testu podane w sekundach umieszczone zostały na Rys. 48.

```
Average time of 1000 requests to name search : 0.016622639999999977
Max time of 1000 requests to name search     : 0.023973
Min time of 1000 requests to name search     : 0.014915
Standard deviation                           : 0.0024501200000000083
```

Rys. 48. Wyniki testów pomiaru szybkości zapytania.

Dodatkowo wykonano podobne testy dla pierwotnej wersji zapytania o interakcje między lekami, zaprezentowanych na Rys. 35 w na stronie 51, ze względu na nieoptymalne zapytanie, zmniejszono ilość zapytań do 10. Kod do testów zaprezentowany jest w dodatku B, natomiast wyniki testu zaprezentowane są na Rys. 49.

```
Average time of 10 requests to old interactions : 29.2036429
Max time of 10 requests to old interactions     : 69.173147
Min time of 10 requests to old interactions     : 8.122134
Standard deviation                              : 13.323168033333333
```

Rys. 49. Wyniki testów pomiaru szybkości zapytania.

Na podstawie przedstawionych powyżej wyników, można stwierdzić, że szybkość z jaką operuje API jest akceptowalna dla większości zapytań. Widoczne są różnice w wydajności pomiędzy różnymi typami zapytań. Najprawdopodobniej spowodowane jest to, przez fakt, że zapytanie o interakcje między lekami wymaga przetworzenia danych przez logikę a czas ten nie tylko jest zróżnicowany pomiędzy różnymi sprzętami na których jest uruchomiona aplikacja. Natomiast zapytanie dotyczące nazw leków są szybko i efektywnie obsługiwane gdyż nie wymagają one przetworzenia po otrzymaniu odpowiedzi z bazy danych.

4.9. Problemy, przyszłość i ograniczenia projektu

Wykonanie projektu okazało się wymagającym zadaniem. Zaprojektowanie i stworzenie od podstaw całego serwisu informatycznego włącznie ze strukturą bazy danych, interfejsu API, frontendu aż po możliwe interakcje użytkownika z systemem wymaga dokładnego planowania każdego elementu struktury systemu.

Projekt umożliwia dalszą rozbudowę o nowe funkcjonalności, może to być udostępnienie API publicznie, wyszukiwarka informacji o konkretnym, jednym leku, lub nawiązanie współpracy z placówkami medycznymi w Polsce.

Niestety projekt ma też swoje ograniczenia. Baza udostępniona przez DrugBank [35] pozwalają jedynie na akademickie użycie, w tym do stworzenia projektu dla pracy inżynierskiej, tak więc praca bez oficjalnego pozwolenia dystrybutora nie może zostać udostępniona publicznie. Natomiast jeżeli udałoby się taką współpracę otrzymać, to

utrzymanie niniejszego projektu wymagałoby ciągłe aktualizacje baz danych interakcji i polskiego zbioru leków.

5. Podsumowanie

Po analizie wyników pomiarów szybkości działania projektu można stwierdzić, że główne cele zostały osiągnięte. Projekt spełnił wszystkie założone wcześniej wymagania, funkcjonalne i niefunkcjonalne.

Zaprojektowany interfejs użytkownika, w dużej mierze przyczynia się do funkcjonalności i użyteczności systemu. Zastosowanie zasad projektowania interfejsu wedle 10 Heurystyk Nielsena [12], pozwoliło na stworzenie platformy, która minimalizuje ryzyko popełnienia błędów przez użytkownika pozytywne doświadczenia użytkownika.

Wykorzystane technologii i szczegółowe zaplanowanie architektury pozwoliło na płynne i responsywne działanie strony, jak i również jego łatwą skalowalność i możliwość dalszego rozwoju.

6. Bibliografia

[1] Masnoon N., Shakib S, Kalisch-Ellett L., Caughey G.E., *What is polypharmacy? A systematic review of definitions*. BMC Geriatr. 2017 Oct 10;17(1):230.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5635569/
Data dostępu: 27.08.2023.

[2] Lu Gao and others, *Medication usage change in older people (65+) in England over 20 years: findings from CFAS I and CFAS II, Age and Ageing,* Volume 47, Issue 2, March 2018, Pages 220–225.
https://doi.org/10.1093/ageing/afx158
Data dostępu: 27.08.2023.

[3] Kantor E.D., Rehm C.D., Haas J.S., Chan A.T., Giovannucci E.L., *Trends in Prescription Drug Use Among Adults in the United States From 1999-2012*, JAMA. 2015 Nov 3;314(17):1818-31.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4752169/
Data dostępu: 27.07.2023.

[4] Taghy N., Cambon L., Cohen J.M., Dussart C., *Failure to Reach a Consensus in Polypharmacy Definition: An Obstacle to Measuring Risks and Impacts-Results of a Literature Review*, Ther Clin Risk Manag. 2020 Feb 11;16:57-73.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7023902/
Data dostępu: 27.08.2023.

[5] NIHR Evidence, *Multiple long-term conditions (multimorbidity): making sense of the evidence,* NIHR.
https://doi.org/10.3310/collection_45881
Data dostępu: 27.08.2023.

[6] Khezrian M., McNeil C.J., Murray A.D., Myint P.K., *An overview of prevalence, determinants and health outcomes of polypharmacy.* Ther Adv Drug Saf. 2020 Jun 12;11:2042098620933741.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7294476/
Data dostępu: 27.08.2023.

[7] Scottish Government Polypharmacy Model of Care Group, *Polypharmacy Guidance, Realistic Prescribing 3rd Edition,* 2018. Scottish Government.
https://www.therapeutics.scot.nhs.uk/wp-content/uploads/2018/04/Polypharmacy-Guidance-2018.pdf
Data dostępu: 27.08.2023.

[8] Błeszyńska-Marunowska E., Jagiełło K., Grodzicki T., and others, *Polypharmacy among elderly patients in Poland: prevalence, predisposing factors, and management strategies*, Pol Arch Intern Med. 2022; 132: 16247.
https://www.mp.pl/paim/issue/article/16347/
Data dostępu: 27.08.2023.

[9] Główny Urząd Statystyczny, Urząd Statystyczny w Szczecinie, *Społeczeństwo informacyjne w Polsce w 2022 r.*, Warszawa, Szczecin 2022.
https://stat.gov.pl/obszary-tematyczne/nauka-i-technika-spoleczenstwo-informacyjne/spoleczenstwo-informacyjne/spoleczenstwo-informacyjne-w-polsce-w-2022-roku,1,16.html
Data dostępu: 27.08.2023.

[10] Nielsen J., *A 100-Year View of User Experience.*
https://www.nngroup.com/articles/100-years-ux/
Data dostępu: 27.08.2023.

[11] Norman D., The Design Of Everyday Things

[12] Nielsen J., *10 Usability Heuristics for User Interface Design*, on Apr. 24, 1994; Updated Nov. 15, 2020.
https://www.nngroup.com/articles/ten-usability-heuristics/
Data dostępu: 27.08.2023.

[13] MDN Web Docs; *HTML.*
https://developer.mozilla.org/en-US/docs/Glossary/HTML
Data dostępu: 27.08.2023.

[14]MDN Web Docs; *CSS.*
https://developer.mozilla.org/en-US/docs/Glossary/CSS
Data dostępu: 27.08.2023.

[15]MDN Web Docs; *JavaScript.*
https://developer.mozilla.org/en-US/docs/Glossary/JavaScript
Data dostępu: 27.08.2023.

[16] Flanagan D., *JavaScript - The Definitive Guide*, 5th ed., *O'Reilly, Sebastopol, CA, 2006*, p.497.
Data dostępu: 27.08.2023.

[17] You E., *First week of launching vue.js.*
https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/
Data dostępu: 27.08.2023.

[18] Cromwell V., *Evan You,* Between the Wires.
https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/
Data dostępu: 27.08.2023.

[19] Google, Oficjalna strona material design.
https://material.io/
Data dostępu: 27.08.2023.

[20] Oficjalna strona projektu Vuetify.
https://vuetifyjs.com/en/introduction/why-vuetify/
Data dostępu: 27.08.2023.

[21] Van Rossum G., Drake L. F.; *Python 3 Reference Manual,* CreateSpace Independent Publishing Platform.
Data dostępu: 27.08.2023.

[22] Stack overflow, *2022 Developer Survey.*
https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022#most-popular-technologies-language
Data dostępu: 27.08.2023.

[23] Canales Luna J., *Top programming languages for data scientists in 2023*, Updated Mar 2023
https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022
Data dostępu: 27.08.2023.

[24] Oficjalna dokumentacja FastAPI.
https://fastapi.tiangolo.com/
Data dostępu: 27.08.2023.

[25] SK Singh; *Cloud Computing: Cloud Computing Fundamentals*
Data dostępu: 29.08.2023.

[26] Oficjalna strona dokumentacji dockera.
https://docs.docker.com/
Data dostępu: 11.09.2023.

[27] Docker hub.
https://hub.docker.com
Data dostępu: 11.09.2023.

[28] Obraz pythona w docker Hub.
https://hub.docker.com/_/python
Data dostępu: 11.09.2023.

[29] Obraz MongoDB w dockerHub.
https://hub.docker.com/_/mongo
Data dostępu: 11.09.2023.

[30] MongoDB compass.
https://www.mongodb.com/products/compass
Data dostępu: 11.09.2023.

[31] Google Cloud Translation API pricing.
https://cloud.google.com/translate/pricing
Data dostępu: 11.09.2023.

[32] Martin R. C., *Czysty kod. Podręcznik dobrego programisty,* Wydawnictwo Helion.
Data dostępu 11.09.2023

[33] Richardson L., *Justice Will Take Us Millions Of Intricate Moves, Act Three: The Maturity Heuristic.*

https://www.crummy.com/writing/speaking/2008-QCon/act3.html
Data dostępu: 07.09.2023.

[34] Koschel A., Astrova I., Blankschyn M., Schöner D., Schulze K., *Evaluating the RESTfulness of "APIs from the Rough"*, Journal: Hochschule Hannover.
https://core.ac.uk/display/286438891?utm_source=pdf&utm_medium=banner&utm_campaign=pdf-decoration-v1
Data dostępu: 11.09.2023.

[35] Oficjalna strona Drugbank.
Wishart D.S., Feunang Y.D., Guo A.C., Lo E.J., Marcu A., Grant J.R., Sajed T., Johnson D., L.i C., Sayeeda Z., Assempour N., Iynkkaran I., Liu Y., Maciejewski A., Gale N., Wilson A., Chin L., Cummings R., Le D., Pon A., Knox C., Wilson M.. DrugBank 5.0: a major update to the DrugBank database for 2018. Nucleic Acids Res. 2017 Nov 8. doi: 10.1093/nar/gkx1037.
https://go.drugbank.com/
Data dostępu: 11.09.2023.

[36] Oficlana strona Vue.js.
https://vuejs.org/
Data dostępu: 11.09.2023.

[37] Walter D., "Computational Complexity Theory", The Stanford Encyclopedia of Philosophy (Fall 2021 Edition).
https://plato.stanford.edu/archives/fall2021/entries/computational-complexity/
Data dostępu: 12.09.2023.

[38] mp.pl, *Leki od a do Z.*
https://www.mp.pl/pacjent/leki/
Data dostępu: 12.09.2023.

[39] Nielsen J., *Usability 101: Introduction to Usability.*
https://www.nngroup.com/articles/usability-101-introduction-to-usability/
Data dostępu: 12.09.2023.

[40] MDN Web Docs, Span Element.
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span
Data dostępu: 12.09.2023.

[41] Meyer E. A., Weyl E., *CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny,* wydawnictwo Helion.

[42] Duckett J., *JavaScript i jQuery. Interaktywne strony WWW dla każdego. Podręcznik Front-End Developera;* Wydawnictwo Helion.

[43] Oficjalna strona projektu React.js.
https://react.dev/
Data dostępu: 12.09.2023.

[44] Oficjalna strona projektu Angular.js.
https://angular.io/

Data dostępu: 12.09.2023.

[45] Oficjalna strona projektu Express.js.
https://expressjs.com/
Data dostępu: 12.09.2023.

[46] Oficjalna strona projektu React Native.
https://reactnative.dev/
Data dostępu: 12.09.2023.

[47] Oficjalna strona projektu Electron.js.
https://www.electronjs.org/
Data dostępu: 12.09.2023.

[48] Oficjalna strona projektu Django.
https://www.djangoproject.com/
Data dostępu: 12.09.2023.

[49] Oficjalna strona projektu Flask.
https://flask.palletsprojects.com
Data dostępu: 12.09.2023.

[50] Oficjalna strona projektu Pandas.
https://pandas.pydata.org/
Data dostępu: 12.09.2023.

[51] Oficjalna strona projektu NumPy.
https://numpy.org/
Data dostępu: 12.09.2023.

[52] Oficjalna strona projektu SciKit.
https://scikit-learn.org/
Data dostępu: 12.09.2023.

[53] Oficjalna strona projektu MongoDb.
https://www.mongodb.com/
Data dostępu: 12.09.2023.

[54] Oficjalna dokumentacja biblioteki PyMongo.
https://pymongo.readthedocs.io/en/stable/
Data dostępu: 12.09.2023.

[55] Oficjalna strona projektu Pinia.
https://pinia.vuejs.org/
Data dostępu: 12.09.2023.

[56] Oficjalna strona biblioteki requests.
https://pypi.org/project/requests/
Data dostępu: 12.09.2023.

7. Spis Rysunków

8. Dodatki

A. Kod napisany w pythonie do pozyskania listy leków ze strony mp.pl

```
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.common.by import By
```

```
import json
from itertools import islice
import concurrent.futures
import os

# global variables
os.environ[
    "GOOGLE_APPLICATION_CREDENTIALS"
] = "D:/INZYNIERKA/selenium/inzynierka-387317-b6c75eaf8bb3.json"
complete_dict = {}


def create_new_browser():
    """Creates new browser to be used in thread."""
    options = Options()
    options.add_argument("-headless")
    driver = webdriver.Firefox(options=options)
    return driver


def translate_text(text):
    """Translate the given text from Polish to English using Google Cloud
Translate."""
    from google.cloud import translate_v2 as translate

    translate_client = translate.Client()

    if isinstance(text, bytes):
        text = text.decode("utf-8")
    result = translate_client.translate(
        text, target_language="en", source_language="pl"
    )
    return result["translatedText"]


def get_ingredients(part_dict):
    """Fetch ingredients for a given set of drugs using multi-threading."""
```

```
    # create new headless browser
    driver = create_new_browser()
    browser_pid = driver.service.process.pid
    print("## New browser with pid: {}".format(browser_pid))

    # from dict passed, go to URL and get ingredients
    for key in part_dict:
        print(browser_pid, "Drug: ", part_dict[key]["drug_name_pl"], key)

        # go to url
        driver.get(part_dict[key]["url"])

        # find ingredients

        try:
            drug_used = driver.find_element(By.CSS_SELECTOR, "a.color")
        except:
            print("No ingredients found for: ",
part_dict[key]["drug_name_pl"])
            continue

        drug_name = drug_used.text

        # if there are more than one ingredient, split it
        if "+" in drug_name:
            parts = drug_name.split(" + ")
            part_dict[key]["ingredients_en"]["0"] = translate_text(parts[0])
            part_dict[key]["ingredients_en"]["1"] = translate_text(parts[1])
            part_dict[key]["ingredients_pl"]["0"] = parts[0]
            part_dict[key]["ingredients_pl"]["1"] = parts[1]
        elif drug_name in ["preparat złożony", "preparat ziołowy"]:
            continue
        else:
            part_dict[key]["ingredients_en"]["0"] = translate_text(drug_name)
            part_dict[key]["ingredients_pl"]["0"] = drug_name

    with open("tmp/lists/list{}.json".format(browser_pid), "w") as outfile:
        outfile.write(json.dumps(part_dict, indent=4))
```

```
    # free memory
    del part_dict
    print("## End of browser with pid {}. Quitting..".format(browser_pid))
    driver.quit()


def chunks(data, size=100):
    """Yield successive n-sized chunks from l."""
    it = iter(data)
    for i in range(0, len(data), size):
        yield {k: data[k] for k in islice(it, size)}


def get_drugs_list(letter_array):
    """Get list of drugs from mp.pl."""

    # create browser
    driver = create_new_browser()

    index = 0
    # iter through letters and get drug list
    for el in letter_array:
        print("-- Letter: {}".format(el))

driver.get("https://www.mp.pl/pacjent/leki/items.html?letter={}".format(el))
        elements = driver.find_elements(By.CSS_SELECTOR, ".drug-list li a")
        for element in elements:
            result = {
                index: {
                    "drug_name_pl": element.text,
                    "url": element.get_attribute("href"),
                    "ingredients_pl": {},
                    "ingredients_en": {},
                }
            }
            complete_dict.update(result)
            index += 1
```

```
        with open("list.json", "w") as outfile:
            outfile.write(json.dumps(complete_dict, indent=4,
ensure_ascii=False))
    driver.quit()


def make_json_list():
    """Generate or update the list.json file."""
    letter_array = [
        2,
        4,
        5,
        "A",
        "B",
        "C",
        "D",
        "E",
        "F",
        "G",
        "H",
        "I",
        "J",
        "K",
        "L",
        "M",
        "N",
        "O",
        "P",
        "Q",
        "R",
        "S",
        "Ś",
        "T",
        "U",
        "V",
        "W",
        "X",
        "Y",
```

```
        "Z",
        "Ż",
    ]

    # get list of drugs
    get_drugs_list(letter_array)


def main():
    """Main function."""

    # generate list of drugs
    make_json_list()

    # load already prepared json list to complete_dict
    file_path = "list.json"
    with open(file_path, "r") as file:
        data = json.load(file)

    # create array of dicts that are equally split
    dicts_array = []
    for item in chunks(data, 400):
        dicts_array.append(item)

    # delete data for memory saving
    del data

    # Parallel execute get_ingredients for dictionaries
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = [
            executor.submit(get_ingredients, dictonary) for dictonary in
dicts_array
        ]
        del dicts_array
        concurrent.futures.wait(futures)

    # read lists prepared from browsers
    lists_dir = "tmp/lists"
```

```
    complete_list = {}
    for filename in os.scandir(lists_dir):
        if filename.is_file():
            with open(filename, "r") as file:
                data = json.load(file)
            os.remove(filename)
            complete_list.update(data)

    # convert dictionary to json
    with open("json_data.json", "w", encoding="utf-8") as json_save:
        json.dump(
            complete_list, json_save, ensure_ascii=False, indent=4,
sort_keys=True
        )
    print("Saved json to file. Quitting.")


if __name__ == "__main__":
    main()
```

B. Kod napisany w pythonie do wykonania pomiaru szybkości odpowiedzi na zapytania

```
import requests
import random
import csv
import numpy as np




items = ["Ibupar%20forte%20(tabletki%20powlekane)",
"Aspirin%20C%20(tabletki%20musuj%C4%85ce)",
"Panadol%20Sprint%20(tabletki%20powlekane)","Dexaven%20(roztw%C3%B3r%20do%20wstrzyk
iwa%C5%84)","Magne%20B6%20(tabletki%20powlekane)",
        "Zio%C5%82a%20mnicha%20na%20cholesterol%20(zio%C5%82a)",
        "Fervex%20(gran.%20do%20sporz.%20roztworu%20doustnego)",
        "Nurofen%20(tabletki%20powlekane)",
```

```
"Fervex%20o%20smaku%20malinowym%20(gran.%20do%20sporz.%20roztworu%20doustnego)",
        "Kaletra%20(roztw%C3%B3r%20doustny)",

"Nurofen%20Junior%20Fort%20(granulat%20do%20sporz%C4%85dzania%20roztworu%20doustneg
o)",
        "Kaletra%20(tabletki%20powlekane)"]


url = "http://localhost:8000/get_pl_drugs_names_to_interactions/"


response = requests.request("GET", url)


resultList = []


for i in range(100):
   randomCount = random.randint(2, 6)
   itemsSeparatedWithComma = ""
   for j in range(randomCount):
       randomIndex = random.randint(0, len(items) - 1)
       itemsSeparatedWithComma = ",".join(items[:randomIndex])
   randomUrl = "http://localhost:8000/get_pl_drugs_names_to_interactions/" +
itemsSeparatedWithComma
   response = requests.request("GET", randomUrl)
   resultList.append(response.elapsed.total_seconds())

mean = sum(resultList) / len(resultList)
sd = np.std(resultList)

print("Average time of 100 requests to interactions : " + str(mean))
print("Max time of 100 requests to interactions     : " + str(max(resultList)))
print("Min time of 100 requests to interactions     : " + str(min(resultList)))
print("Standard deviation                           : " + str(sd))




print("\n-------------------------------------------------\n")
```

```
url2 = "http://localhost:8000/drug_name_starting_with/"
resultList2 = []


for i in range(1000):
    randomLetters = ""
    for j in range(4):
        randomLetters += chr(random.randint(97, 122))
    randomUrl = url2 + randomLetters
    response = requests.request("GET", randomUrl)
    resultList2.append(response.elapsed.total_seconds())


mean = sum(resultList2) / len(resultList2)
sd = np.std(resultList2)

print("Average time of 1000 requests to name search : " + str(mean))
print("Max time of 1000 requests to name search     : " + str(max(resultList2)))
print("Min time of 1000 requests to name search     : " + str(min(resultList2)))
print("Standard deviation                           : " + str(sd))







print("\n-------------------------------------------------\n")


url = "http://localhost:8000/OLD_get_pl_drugs_names_to_interactions/"


response = requests.request("GET", url)


resultList = []


for i in range(10):
    print("Request number: " + str(i))
    randomCount = random.randint(2, 6)
    itemsSeparatedWithComma = ""
    for j in range(randomCount):
        randomIndex = random.randint(1, len(items) - 1)
```

```
        itemsSeparatedWithComma = ",".join(items[:randomIndex])
    randomUrl = "http://localhost:8000/OLD_get_pl_drugs_names_to_interactions/" +
itemsSeparatedWithComma
    response = requests.request("GET", randomUrl)
    resultList.append(response.elapsed.total_seconds())

mean = sum(resultList) / len(resultList)
sd = np.std(resultList)

print("Average time of 10 requests to old interactions : " + str(mean))
print("Max time of 10 requests to old interactions     : " + str(max(resultList)))
print("Min time of 10 requests to old interactions     : " + str(min(resultList)))
print("Standard deviation                              : " + str(sd))
```