Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **ITE**
Specjalność: **INS**

PRACA DYPLOMOWA MAGISTERSKA

Wielokryterialna analiza frameworków frontendowych z użyciem wnioskowania rozmytego

Dominik Tłokiński

Opiekun pracy
Prof. dr hab. inż. Jan Magott

Słowa kluczowe: analiza porównawcza, wnioskowanie rozmyte, *Vue*, *React*, *Angular*

WROCŁAW 2024

Streszczenie

W niniejszej pracy przedstawiono proces budowy aplikacji internetowych w trzech popularnych frameworkach frontendowych: *Angular*, *React* i *Vue* oraz przeprowadzono szczegółową analizę porównawczą. Analiza dotyczyła kryteriów wydajności, rozmiaru paczek produkcyjnych, elastyczności, popularności, bezpieczeństwa oraz implementacji DOM. Przeprowadzone analizy mają na celu ułatwienie programistom wyboru odpowiedniego narzędzia do tworzenia aplikacji webowych.

Implementacja aplikacji obejmowała część frontendową i backendową, realizowaną za pomocą *ASP.NET Core* oraz bazy danych *PostgreSQL*. Dodatkowo, w projekcie wykorzystano bazę danych *Redis* do przechowywania danych wyświetlanych na stronie szczegółów ofert. W projekcie wykorzystane zostały narzędzia konteneryzacyjne *Docker* i *Docker Compose* w celu ułatwienia zarządzania środowiskiem aplikacji.

Wyniki analizy porównawczej, przedstawione w pracy, obejmują nie tylko mierzalne parametry techniczne, ale również subiektywne opinie czy kwestie bezpieczeństwa bądź elastyczności. Dzięki zastosowaniu wnioskowania rozmytego możliwe było stworzenie modelu, który uwzględniając wszystkie czynniki, pozwolił na zaprogramowanie systemu wspomagającego decyzję o wyborze odpowiedniego frameworka frontendowego.

Abstract

This thesis presents the process of building web applications using three popular frontend frameworks: *Angular*, *React*, and *Vue*, and conducts a detailed comparative analysis. The analysis covers criteria such as performance, production package size, flexibility, popularity, security, and DOM implementation. The conducted analyses aim to assist programmers in choosing the appropriate tool for creating web applications.

The implementation of the applications included both frontend and backend parts, realized using *ASP.NET Core* and the *PostgreSQL* database. Additionally, the project utilized the *Redis* database for storing data displayed on the offer details page. The project also employed containerization tools *Docker* and *Docker Compose* to facilitate the management of the application environment.

The comparative analysis results presented in this thesis encompass not only measurable technical parameters but also subjective opinions, security issues, and flexibility. By applying fuzzy logic, it was possible to create a model that, taking all factors into account, allowed for the programming of a system to support the decision-making process in choosing the appropriate frontend framework.

Spis treści

WSTĘP

Podczas pracy jako programista aplikacji internetowych z użyciem frameworku[1] *Angular* zaobserwowano duże zróżnicowanie zdań społeczności na temat tego, jakie narzędzie do tworzenia aplikacji jest najlepsze pod względem wydajności, sposobu tworzenia skomplikowanych komponentów w prosty sposób, testowania czy kompletności dokumentacji. Niejednokrotnie spotykano się ze stwierdzeniem, że któreś narzędzie jest złe, trudne, wymaga użycia zbyt dużej ilości kodu szablonowego (ang. „*boilerplate code*") lub jest przestarzałe. W Internecie zaczęło krążyć pytanie „Które narzędzie do pisania aplikacji internetowych jest najlepsze?". Nie udało się znaleźć artykułów bądź prac korzystających z metody wnioskowania rozmytego, które udzieliłyby odpowiedzi na to pytanie. W związku z tym rozpoczęto pracę nad stworzeniem aplikacji w trzech narzędziach: *Angular, React* oraz *Vue*. Będą one podstawą do analizy bibliotek oraz stworzenia systemu wnioskowania rozmytego opartego na zbiorach i regułach rozmytych. W niniejszej pracy przedstawiono proces budowania aplikacji, porównywania narzędzi w wielu aspektach oraz tworzenia systemu wnioskowania rozmytego, który będzie miał na celu uproszczenie wyboru odpowiedniego frameworka.

CEL I ZAKRES PRACY

Celem pracy jest zbudowanie aplikacji internetowych używając trzech frameworków. Tematyka aplikacji to serwis ogłoszeniowy sprzedaży samochodów. Te aplikacje będą służyć do analizy i porównania tych narzędzi z uwzględnieniem kryteriów:

- Składni oraz struktury,
- Wydajności,
- Rozmiarów paczek,
- Elastyczności,
- Popularności,
- Bezpieczeństwa,
- Implementacji DOM (Document Object Model).

Po zebraniu informacji na temat tych kryteriów zostanie przygotowany system wnioskowania rozmytego, który będzie ułatwiał wybór odpowiedniego narzędzia. Będzie to wymagało zdefiniowania funkcji przynależności dla zbiorów rozmytych, a także opartych na nich reguł.

Zakres pracy składa się ze zbudowania aplikacji internetowej za pomocą języków *C#* i *Typescript* oraz frameworków *Angular, React, Vue* i ASP.NET Core. W celu komunikacji *backendu* i *frontendu* wystawione zostanie RESTowe API po stronie serwera, który będzie łączył z bazą danych *PostgreSQL*. W ramach tego projektu zostanie też postawiona baza *NoSQL – Redis*. Z tą bazą łączyć się będzie dodatkowy mniejszy serwer używający biblioteki *express*, do którego aplikacje frontendowe będą wysyłać zapytania REST.

Po zbudowaniu aplikacje zostaną porównane pod względem składni i struktury za pomocą metryk *Chidambera Kemerera*. Wydajność zostanie przetestowana narzędziami *WebPageTest* i *Sonarqube*. Porównaniu ulegną również rozmiary paczek dla pustego, skończonego projektu. Rozmiar zmieni się też dla paczki produkcyjnej. Porównane zostaną popularność, bezpieczeństwo i sposób implementacji DOM.

[1] „Frameworki określają strukturę aplikacji, jej mechanizm działania oraz dostarczają biblioteki i komponenty przydatne podczas tworzenia programów." [1]

System wnioskowania rozmytego będzie zbudowany za pomocą języka *Python* i biblioteki *SciKit-Fuzzy*. W ramach tego systemu na podstawie wcześniejszej analizy aspektów porównawczych zdefiniowane zostaną zbiory rozmyte oraz reguły wnioskowania.

1 WPROWADZENIE

Obecnie rozwój w zakresie tworzenia aplikacji webowych jest nieodłącznym elementem zmieniającego się świata technologii informacyjnych. Programiści aplikacji internetowych muszą zmierzyć się z wyzwaniem wyboru odpowiedniego frameworka frontendowego. Narzędzie powinno nie tylko spełnić wymagania funkcjonalne programisty, ale także być wydajne, skalowalne i łatwe w utrzymaniu. Na rynku dostępnych jest wiele frameworków. To sprawia, że decyzja wyboru odpowiedniej technologii nie jest prosta, szczególnie, jeśli uwzględnione zostaną różnorodne kryteria techniczne i biznesowe.

Celem niniejszej pracy magisterskiej jest przeprowadzenie wielokryterialnej analizy frameworków frontendowych z użyciem wnioskowania rozmytego. Wnioskowanie rozmyte jest oparte na teorii zbiorów rozmytych. Pozwala to na analizę informacji, które są niedokładne i niepewne. Takie analizy są nieodłącznym elementem procesu decyzyjnego w warunkach rzeczywistych. Wnioskowanie rozmyte pozwoli na porównanie różnych frameworków, uwzględniając nie tylko mierzalne parametry techniczne, ale także subiektywne opinie programistów i użytkowników.

W pracy zostaną omówione trzy popularne frameworki frontendowe: *Angular, React* i *Vue.js*. Zdobyły one uznanie wśród programistów i firm technologicznych na całym świecie. Analiza obejmie wiele kryteriów, takich jak wydajność, łatwość nauki, rozmiar paczek produkcyjnych, strukturę aplikacji, implementację DOM oraz wbudowane bezpieczeństwo. Wykorzystanie wnioskowania rozmytego pozwoli na prosty wybór odpowiedniego frameworku dla programisty.

Większe oczekiwania użytkowników końcowych względem aplikacji webowych spowodowały dynamiczny rozwój narzędzi wspierających ich tworzenie. Przeprowadzenie rzetelnej analizy porównawczej frameworków frontendowych jest kluczowe dla sukcesu wielu projektów IT. Niewłaściwie wybrane narzędzie może spowodować opóźnienia, wzrost kosztów i inne problemy związane z aplikacją i jej utrzymaniem.

Mimo istniejących badań i analiz porównujących frameworki frontendowe, brakuje prac wykorzystujących wnioskowanie rozmyte do oceny tych narzędzi. Przeprowadzenie wielokryterialnej analizy z użyciem wnioskowania rozmytego umożliwi stworzenie modelu decyzyjnego. Będzie on wykorzystany przez deweloperów i menedżerów projektów przy wyborze najbardziej odpowiedniego narzędzia uwzględniając potrzeby i warunki projektu.

Wyniki analiz zostaną przedstawione w formie przejrzystych tabel i wykresów, pozwoli to na łatwe porównanie i zrozumienie mocnych i słabych stron każdego z frameworków.

1.1 LUKI W BADANIACH

Wiele artykułów porusza temat najlepszego frameworka frontendowego. Przykładowo artykuł „*Evaluating the Performance of Web Rendering Technologies Based on Javascript: Angular, React, and Vue*" [25] przedstawia testy wydajności wskazujące na przewagę *Reacta* i *Vue* nad *Angularem* w kontekście szybkości renderowania i czasu odpowiedzi. Artykuł ten niestety nie zbadał kryteriów takich jak bezpieczeństwo czy elastyczność danych frameworków. Nie rozwiązał też problemu programistów, którzy nie wiedzą, którą technologię powinni wybrać. Kryteria takie jak łatwość nauki, popularność czy elastyczność są istotnie ważne przy wyborze frameworka. W artykule „*DOM Benchmark Comparison of the Front-End Javascript Frameworks React, Angular, Vue, and Svelte*" przeprowadzona została analiza popularności i wsparcia społeczności frameworków. Jednak przeciętny użytkownik, mimo wiedzy o tych aspektach wciąż może mieć problem z wyborem technologii. Nie będzie wiedział, który z frameworków jest najbardziej wydajny. Brak wiedzy spowoduje, że użytkownik nie będzie potrafił zdecydować się na któreś z narzędzi.

Artykuł *„Evaluating the Performance of Web Rendering Technologies Based on Javascript: Angular, React, and Vue*" [25] dostarcza wiele informacji na temat wydajności frameworków w różnych scenariuszach renderowania. Jednak ten artykuł koncentruje się głównie na technicznych aspektach wydajności, pomijając inne ważne kryteria wspomniane wcześniej. Zastosowanie teorii zbiorów rozmytych pozwoli na uwzględnienie wszystkich czynników – w tym tych subiektywnych. Zapewni to bardziej kompleksową i wszechstronną analizę. Stworzenie modelu będzie stanowiło pomoc w wyborze odpowiedniego frameworka frontendowego dla użytkowników.

2 FAZA PROJEKTOWA

Początkowym etapem budowania systemu informatycznego jest faza projektowa. W jej ramach należy ustalić cele systemu oraz określić założenia. Etap ten powinien zostać przeprowadzony przed rozpoczęciem programowania i implementacji rozwiązań. W tym celu wyszczególniono wymagania funkcjonalne oraz niefunkcjonalne. Przygotowany został też diagram związków encji, a także przypadków użycia wraz z opisem scenariuszy.

2.1 WYMAGANIA FUNKCJONALNE

Wymaganiem funkcjonalnym aplikacji jest przede wszystkim umożliwienie użytkownikowi przeglądania ofert sprzedaży samochodów i motocykli. Aplikacja powinna udostępniać możliwość przeglądania szczegółów konkretnej oferty. Te szczegóły obejmują większą ilość zdjęć, wyposażenie pojazdu, opis i cenę. Powinna istnieć strona, która pozwala na stworzenie nowej oferty W trakcie tworzenia ogłoszenia możliwe jest dodanie zdjęć oferty poprzez przeciągnięcie i upuszczenie pliku o odpowiednim rozszerzeniu w sekcji *„Zdjęcia”*. Na ekranie głównym aplikacji wyświetlone zostają: formularz umożliwiający precyzyjne wyszukiwanie ofert, a także sekcja z wyróżnionymi ofertami.

2.2 WYMAGANIA NIEFUNKCJONALNE

Serwis ogłoszeniowy wymaga przechowywania dużej ilości zdjęć, a trzymanie obrazów w bazie danych nie jest optymalne. Z tego powodu w *PostgreSQL* zapisane zostaną tylko ścieżki do zdjęć znajdujących się na dysku. Bazy danych oraz serwer powinny być skonteneryzowane za pomocą narzędzia *Docker* i *Docker Compose*. Celem pogodzenia konteneryzacji serwera z zapisywaniem zdjęć na dysku powinien zostać stworzony wspólny wolumin, z którego kontener będzie pobierał dane z komputera. Projekt korzysta z bazy *Redis,* by zapisywać w niej dane wyświetlane na stronie szczegółów oferty, którą przeglądał użytkownik. Czas przechowywania danych konkretnej oferty powinien wynosić pięć minut. Aplikacja powinna wymagać wypełnienia wszystkich danych na stronie tworzenia oferty, by móc stworzyć ogłoszenie.

2.3 DIAGRAM ER

Diagram związków encji (Rysunek 2.1) składa się z trzynastu tabel. Sześć z nich (*Body_Type, Car_Status, Fuel_Type, Transmission_Type, Drive_Type, Vehicle_Type*) posiada tylko jeden atrybut typu tekstowego – zupełnie jak *enum*. Przyjęto takie rozwiązanie ze względu na częste problemy z mapowaniem typu wyliczeniowego w bazie danych do tego po stronie serwera.

Najważniejszą encją w projekcie serwisu ogłoszeniowego jest tabela *Offer*. Ta tabela jest w relacji z trzema innymi tabelami. Do jednej oferty może być przypisany tylko jeden pojazd, stąd relacja *jeden-do-jednego* z tabelą *Vehicle*. Ogłoszenie powinno zawierać zdjęcia sprzedawanego przedmiotu, dlatego tabela *Vehicle_Image* łączy się z encją *Offer* w relacji *wiele-do-jednego*. Oferta powinna mieć przypisanego użytkownika, który ją wystawił. W przypadku tego systemu każdy użytkownik będzie nazwany *Dealerem*. Z tego powodu *Dealer* jest w relacji *jeden-do-wielu* z tabelą *Offer*.

Tabela *Vehicle_Image* przechowuje informacje na temat zdjęć konkretnego ogłoszenia. W samej tabeli zapisane są ścieżki do pliku obrazu znajdującego się na maszynie, na której uruchomiony jest serwer. Zapisywanie obrazów w bazie danych jest anty-wzorcem. W przypadku aplikacji działających na rynku takie obrazy powinny być przechowywane w oddzielnym serwisie takim jak *Nuxeo*. Wtedy zamiast ścieżki do pliku wystarczyłoby zapisać w tabeli id zdjęcia przechowywanego na tej platformie. Następnie takie zdjęcia można pobrać zapytaniem z serwera, bez konieczności przechowywania plików lokalnie na maszynie, która utrzymuje *backend*.

Najbardziej wyróżniającą się jest tabela *Vehicle*. Zawiera ona atrybuty dokładnie opisujące cechy sprzedawanego pojazdu. Jest ona w relacji *wiele-do-jednego* z sześcioma tabelami wymienionymi powyżej.

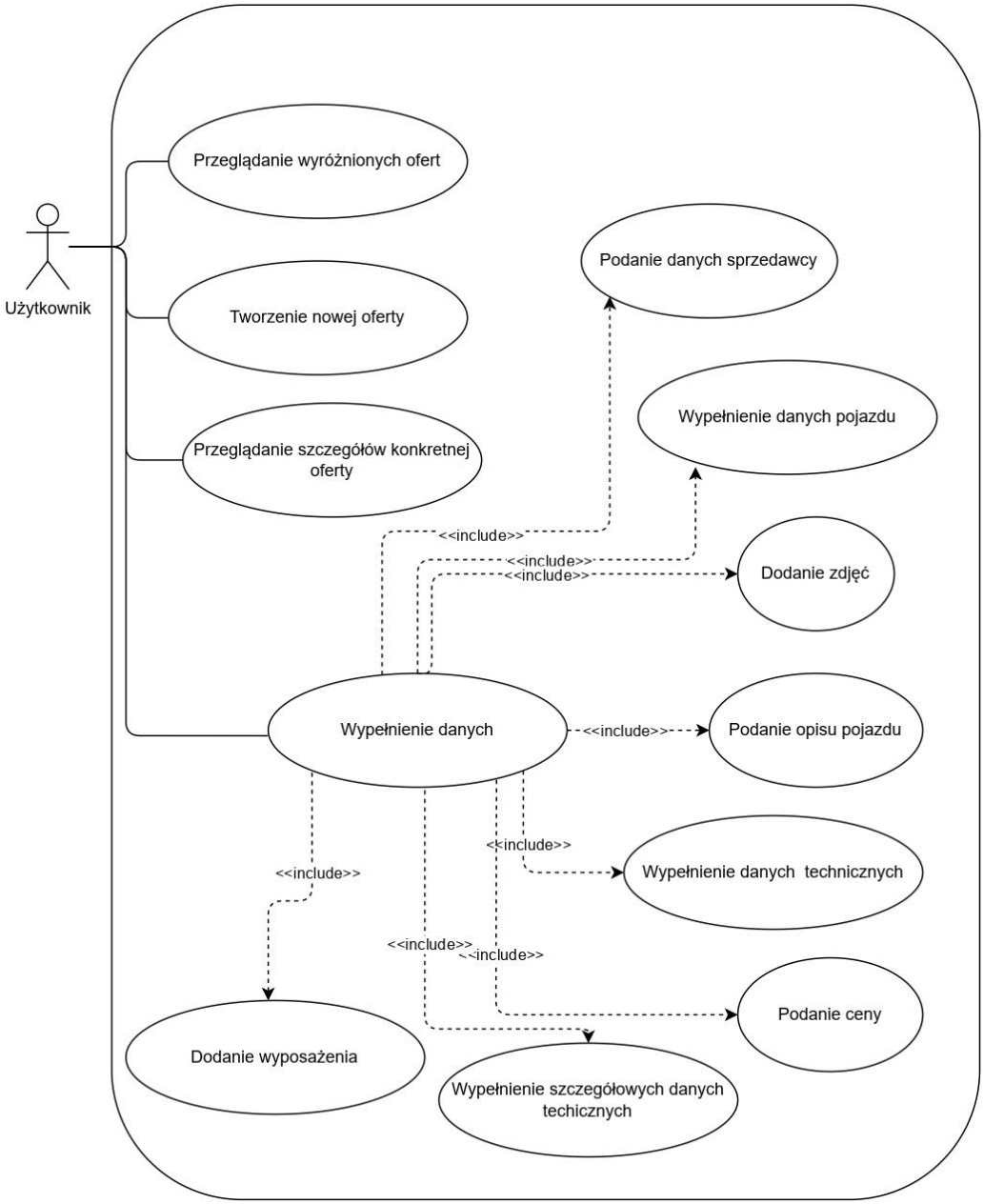
Każdy pojazd ma wyposażenie. W przypadku tego systemu tabela *Vehicle* jest w relacji *jeden-do-wielu* z tabelą krzyżową *Vehicle_Equipment* posiadającą atrybuty łączące id tabeli pojazdu oraz tabeli wyposażenia. Tabela krzyżowa jest znanym sposobem omijania relacji *wiele-do-wielu*, która normalnie występowałaby pomiędzy tymi zbiorami danych. Encja *Equipment* jest w relacji *wiele-do-jednego* z tabelą *Equipment_Type*. Jest to spowodowane podziałem wyposażenia na kategorie takie jak: audio i multimedia, komfort i dodatki, czy *„Dla samochodów elektrycznych”*. Każde wyposażenie należy do któregoś z rodzajów wyposażenia.

Rysunek 2.1: Diagram związków encji

2.4 DIAGRAM PRZYPADKÓW UŻYCIA

Przypadek użycia to opis interakcji aktora z systemem przy pomocy języka naturalnego. Może przyjmować różną formę. Najprostszą z nich to pojedyncze zdanie określające cel danego przypadku. Zachowanie może być opisane za pomocą informacji tj. [2]:

- nazwa przypadku użycia,
- aktor biorący udział w interakcji,
- scenariusz.



Rysunek 2.2: Diagram przypadków użycia

2.5 OPIS I DEFINICJA SCENARIUSZY PRZYPADKÓW UŻYCIA

Diagram przypadków użycia (Rysunek 2.2) przedstawia jedynego aktora w systemie - *Użytkownika*. Istnieją cztery główne przypadki: „Przeglądanie wyróżnionych ofert”,

„Tworzenie nowej oferty", „Przeglądanie szczegółów konkretnej oferty" oraz „Wypełnianie danych" odnoszące się do czynności, które należy wykonać w trakcie zakładania ogłoszenia. Ostatni wymieniony przypadek wymaga od użytkownika podania danych sprzedawcy, pojazdu, opisu pojazdu, szczegółowych danych technicznych oraz ceny. Użytkownik musi pamiętać o dodaniu zdjęć do ogłoszenia, a także uzupełnieniu informacji na temat wyposażenia pojazdu.

Scenariusze przypadków użycia zostały zdefiniowane poniżej:

Opis przypadku użycia: Przeglądanie wyróżnionych ofert
Cel: Przedstawienie użytkownikowi godnych uwagi ofert.
Warunki wstępne: Użytkownik znajduje się na ekranie głównym aplikacji.
Warunki końcowe: Wyróżnione oferty zostają przedstawione użytkownikowi.
Przebieg:

1. Użytkownik otwiera stronę główną aplikacji.
2. Zostaje wyświetlona lista wyróżnionych ofert.
3. Użytkownik w zależności od tego, czy zainteresowała go dana oferta, może ją otworzyć wywołując **PU Przeglądanie szczegółów konkretnej oferty**.

Opis przypadku użycia: Tworzenie nowej oferty
Cel: Umożliwienie użytkownikowi stworzenia nowej oferty.
Warunki wstępne: Użytkownik znajduje się na ekranie głównym aplikacji.
Warunki końcowe: Użytkownik pomyślnie stworzył ofertę.
Przebieg:

1. Użytkownik otwiera stronę tworzenia oferty klikając przycisk „Wystaw ogłoszenie".
2. Użytkownik wykonuje **PU Wypełnienie danych**.
3. Użytkownik tworzy ofertę poprzez naciśnięcie przycisku „Dodaj ogłoszenie".

Opis przypadku użycia: Przeglądanie szczegółów konkretnej oferty.
Cel: Umożliwienie użytkownikowi poznania szczegółów oferty.
Warunki wstępne: Użytkownik znajduje się na ekranie wyświetlającym oferty w miniaturze.
Warunki końcowe: Użytkownik przegląda szczegóły oferty.
Przebieg:

1. Użytkownik naciska miniaturę interesującej go oferty. Skutkuje to przeniesieniem na stronę przedstawiającą szczegóły oferty.
2. Wyświetlone zostają galeria zdjęć, wyposażenie, opis pojazdu oraz cena.

Opis przypadku użycia: Wypełnienie danych
Cel: Umożliwienie użytkownikowi stworzenie nowej oferty.
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w każdej sekcji formularza tworzącego ofertę.
Przebieg:

1. Użytkownik wykonuje **PU Podanie danych sprzedawcy**, **PU Wypełnianie danych pojazdu**, **PU Dodanie zdjęć**, **PU Podanie opisu pojazdu**, **PU Wypełnienie danych technicznych**, **PU Podanie ceny**, **PU Wypełnienie szczegółowych danych technicznych**, **PU Dodanie wyposażenia**.

Opis przypadku użycia: Podanie danych sprzedawcy
Cel: Wypełnienie sekcji „Dane sprzedającego".

Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w sekcji „Dane sprzedającego” w formularzu tworzącym ofertę.
Przebieg:

1. Użytkownik wypełnia pole „Twoje imię”,
2. Użytkownik wypełnia pole „Adres”,
3. Użytkownik wypełnia pole „Numer telefonu”.

Opis przypadku użycia: Wypełnienie danych pojazdu
Cel: Wypełnienie sekcji „Cechy główne” oraz „Informacje podstawowe”.
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w sekcji „Cechy główne” oraz „Informacje podstawowe” w formularzu tworzącym ofertę.
Przebieg:

1. Użytkownik zaznacza odpowiedni przycisk przy polach „Uszkodzony”, „Importowany”.
2. Użytkownik wypełnia pole „VIN”,
3. Użytkownik wypełnia pole „Przebieg”.

Opis przypadku użycia: Dodanie zdjęć
Cel: Dodanie zdjęć pojazdu w sekcji „Zdjęcia”
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik pomyślnie dodał zdjęcia do oferty.
Przebieg:

1. Użytkownik poprzez przeciągnięcie zdjęcia na komponent lub poprzez naciśnięcie przycisku „Dodaj zdęcie” może załączyć zdjęcia do formularza tworzenia oferty.
2. Po dodaniu obrazów zostają one wyświetlone w sekcji „Zdjęcia”

Opis przypadku użycia: Podanie opisu pojazdu
Cel: Wypełnienie sekcji „Cechy główne” oraz „Informacje podstawowe”.
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w sekcji „Opis pojazdu”.
Przebieg:

1. Użytkownik wypełnia pole „Tytuł”,
2. Użytkownik wypełnia pole „Opis”.

Opis przypadku użycia: Wypełnienie danych technicznych
Cel: Wypełnienie danych sekcji „Wyświetl dodatkowe szczegóły”
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w podsekcjach „Dane techniczne” oraz „Wyposażenie”.
Przebieg:

1. Użytkownik wypełnia pola w podsekcji „Dane techniczne”
2. Użytkownik zaznacza odpowiednie wyposażenie w sekcji „wyposażenie”

Opis przypadku użycia: Podanie ceny
Cel: Wypełnienie danych sekcji „Cena”
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w sekcji „Cena”.

Przebieg:

1. Użytkownik wypełnia pole „Cena”,
2. Użytkownik wypełnia pole „Waluta”
3. Użytkownik zaznacza przyciskiem, czy cena jest netto, czy brutto.

Opis przypadku użycia: Wypełnienie szczegółowych danych technicznych
Cel: Wypełnienie danych sekcji „Cena”
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik wypełnił dane w sekcji „Wyświetl dodatkowe szczegóły”, zakładce „Dane techniczne”.
Przebieg:

1. Użytkownik rozwija zakładkę „Dane techniczne” w sekcji „Wyświetl dodatkowe szczegóły.
2. Użytkownik wypełnia pole „Napęd”, „Emisja CO2”, „Rodzaj koloru”, „Liczba miejsc”
3. Użytkownik zaznacza przyciskiem, czy kierownica znajduje się po prawej stronie.

Opis przypadku użycia: Dodanie wyposażenia
Cel: Wskazanie wyposażenia pojazdu, którego dotyczy nowa oferta.
Warunki wstępne: Użytkownik znajduje się na ekranie tworzenia oferty
Warunki końcowe: Użytkownik zaznaczył odpowiednie kratki w sekcji „Wyświetl dodatkowe szczegóły”, zakładce „Wyposażenie”.
Przebieg:

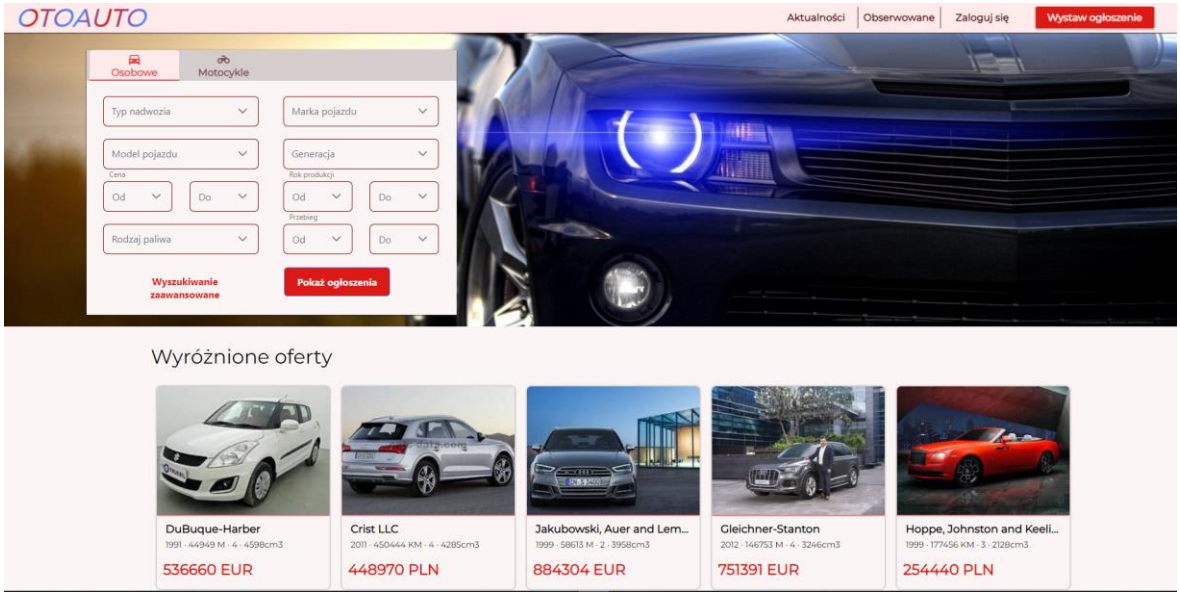
1. Użytkownik rozwija zakładkę „Wyposażenie” w sekcji „Wyświetl dodatkowe szczegóły.
2. Użytkownik rozwija zakładki znajdujące się w zakładce „Wyposażenie”.
3. Użytkownik zaznacza wyposażenie swojego pojazdu.

3 INTERFEJS UŻYTKOWNIKA

Design interfejsu użytkownika ma wpływ na wysiłek, który użytkownik musi zużyć, by uzupełnić odpowiednie dane, a także zinterpretować odpowiedź systemu. Na jego działanie wpływa także to, jak długo zajmuje mu nauczenie się korzystania z aplikacji. Pośród wielu cech, które ma interfejs użytkownika, można wymienić m.in: użyteczność – stopień, do którego design konkretnego interfejsu bierze pod uwagę czynniki takie jak: psychologia i fizjologia użytkowników, a następnie sprawia, że proces korzystania z systemu jest efektywny, satysfakcjonujący i wydajny [3]. Aplikacja musi być intuicyjna w obsłudze, ponieważ osoby w podeszłym wieku, które z niej korzystają mogą mieć problem, by się po niej sprawnie poruszać. Skomplikowany i nieczytelny interfejs może powodować frustrację. By jej uniknąć aplikacja została stworzona ze zrozumiałych, dużych i podpisanych komponentów.

3.1 WIDOK STRONY GŁÓWNEJ

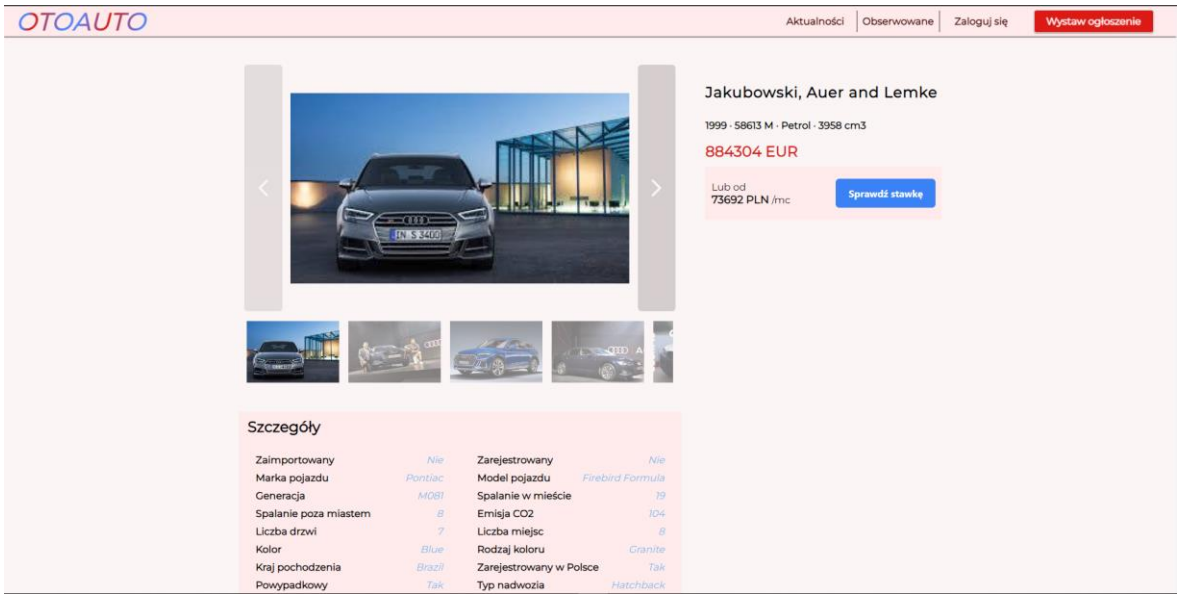
Po wejściu na stronę główną użytkownik ma do wyboru wyszukanie interesujących go marek samochodu używając wyszukiwarki znajdującej się po lewej stronie ekranu. Może on też przeglądać wyróżnione oferty znajdujące się w dolnej części strony. Kliknięcie w którąś z ofert spowoduje przeniesienie użytkownika na ekran szczegółów tej oferty. W prawym górnym rogu znajduje się przycisk wystaw ogłoszenie, który otworzy stronę zakładania oferty.



Rysunek 3.1: Widok strony głównej

3.2 WIDOK SZCZEGÓŁÓW OFERTY

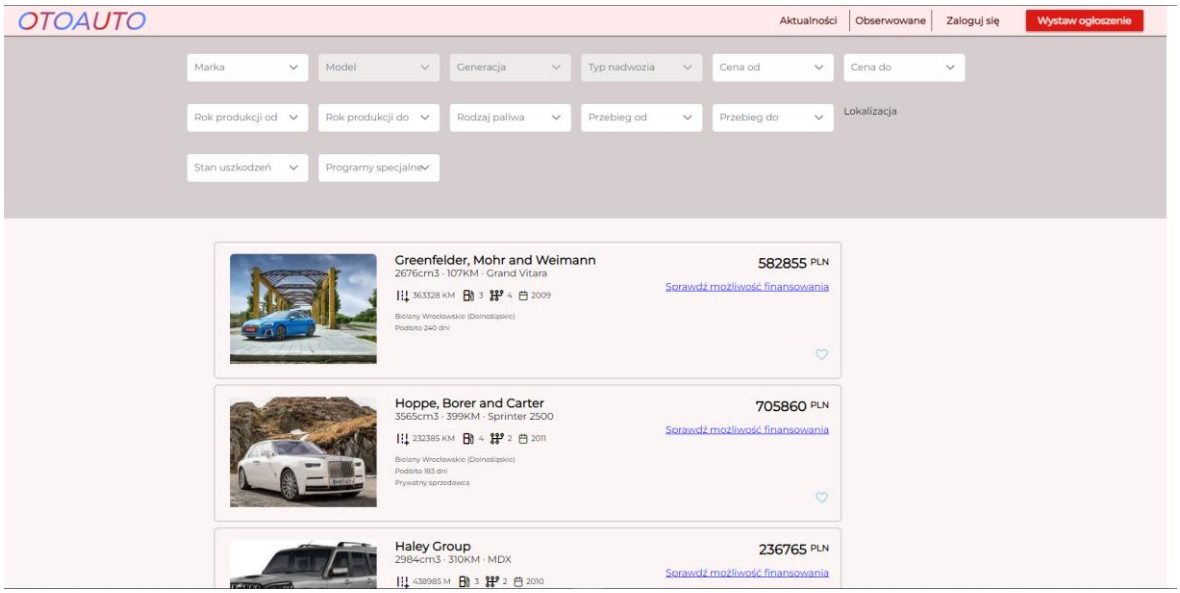
Głównym elementem ekranu szczegółów oferty jest galeria zdjęć, która pozwala na zapoznanie się z wyglądem i stanem pojazdu. Pod zdjęciami znajduje się komponent wyświetlający szczegóły danego pojazdu. Informacje zawarte w tym miejscu dotyczą marki, modelu, mocy czy roku produkcji sprzedawanej maszyny. Z prawej strony galerii znajduje się komponent wyświetlający tytuł i najważniejsze informacje dotyczące pojazdu. Pod nimi umieszczony został komponent wyświetlający cenę pojazdu.



Rysunek 3.2: Widok strony szczegółów oferty

3.3 WIDOK PRZEGLĄDARKI OFERT

Po kliknięciu przycisku „Pokaż ogłoszenia" znajdującego się na ekranie głównym (Rysunek 3.1) użytkownik zostaje przekierowany na stronę przeglądania ofert. Na tym ekranie oferty wyświetlone są w większym komponencie, zawierają więcej informacji, a kliknięcie na nie spowoduje przejście na ekran szczegółów oferty (Rysunek 3.2).

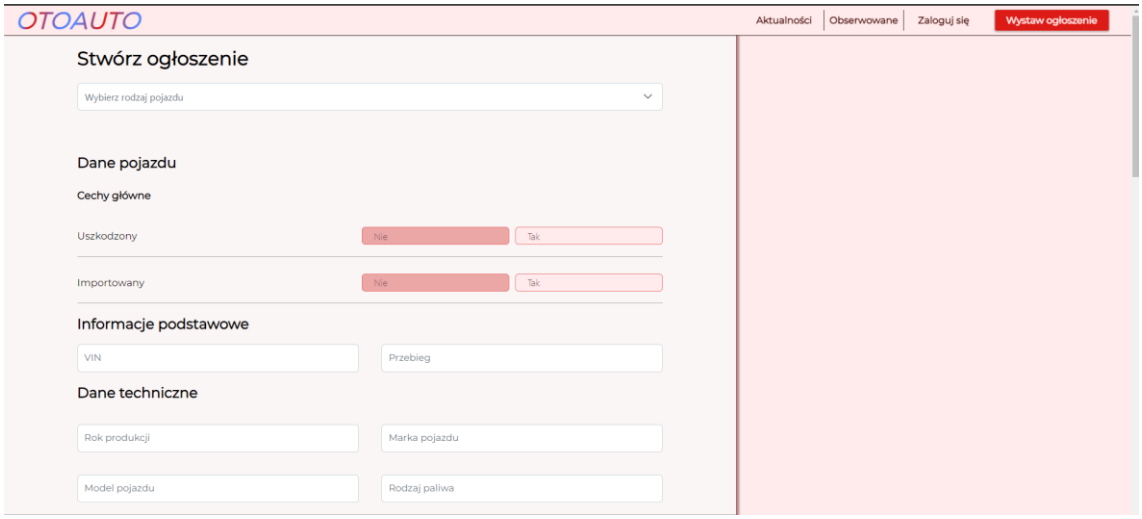


Rysunek 3.3: Widok strony przeglądania ofert

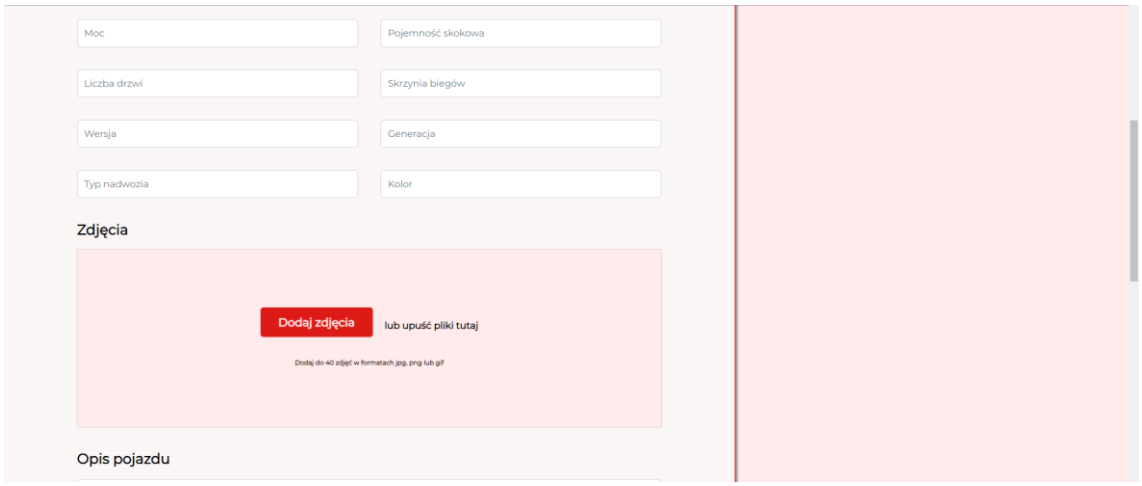
3.4 WIDOKI EKRANU TWORZENIA OGŁOSZENIA

Podstrona służąca do tworzenia ogłoszeń wyświetla użytkownikowi duży formularz, który użytkownik musi wypełnić, by założyć ofertę. Ekran udostępnia możliwość wypełnienia pól opisujących sprzedawany pojazd. Najciekawszymi sekcjami w tym widoku są sekcje:

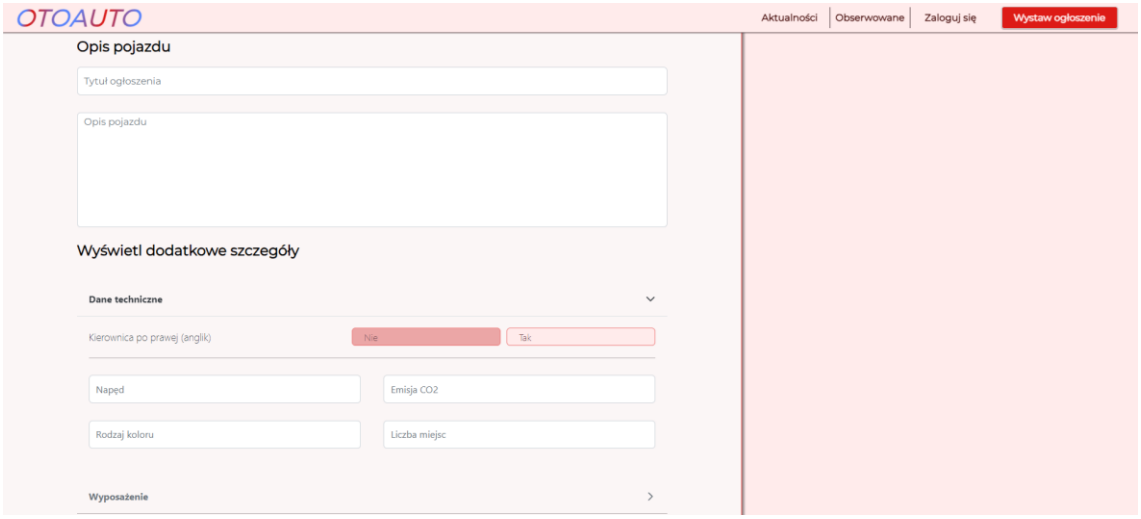
- „Zdjęcia” – umożliwiające dodawanie zdjęć do formularza poprzez przeciągnięcie pliku o formacie .jpg lub .png i upuszczenie go w tej sekcji,
- „Wyświetl dodatkowe szczegóły”, podsekcja „Wyposażenie” – dynamicznie uzupełniana sekcja, która pozwala na wybranie wyposażenia, które ma sprzedawany pojazd.



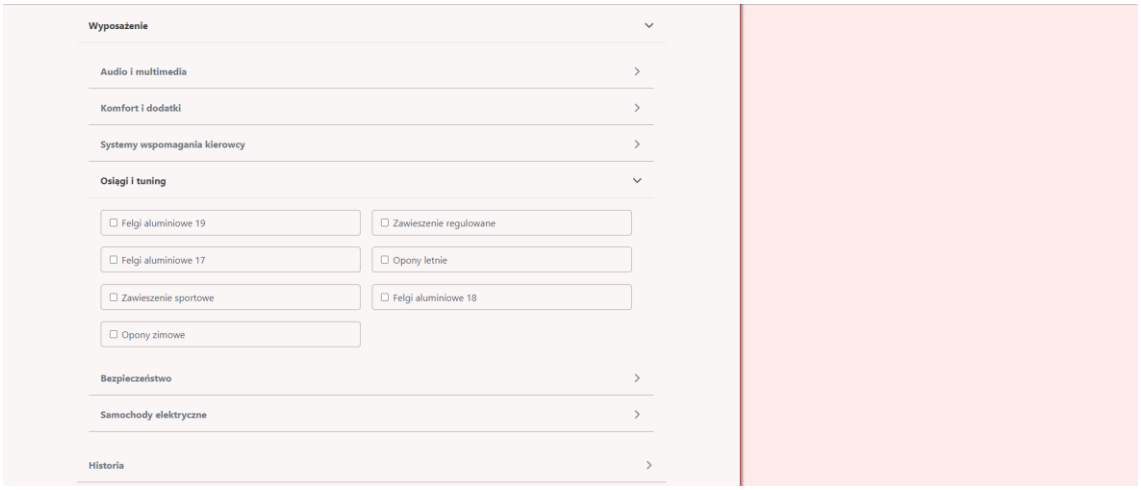
Rysunek 3.4: Ekran tworzenia ogłoszenia cz.1



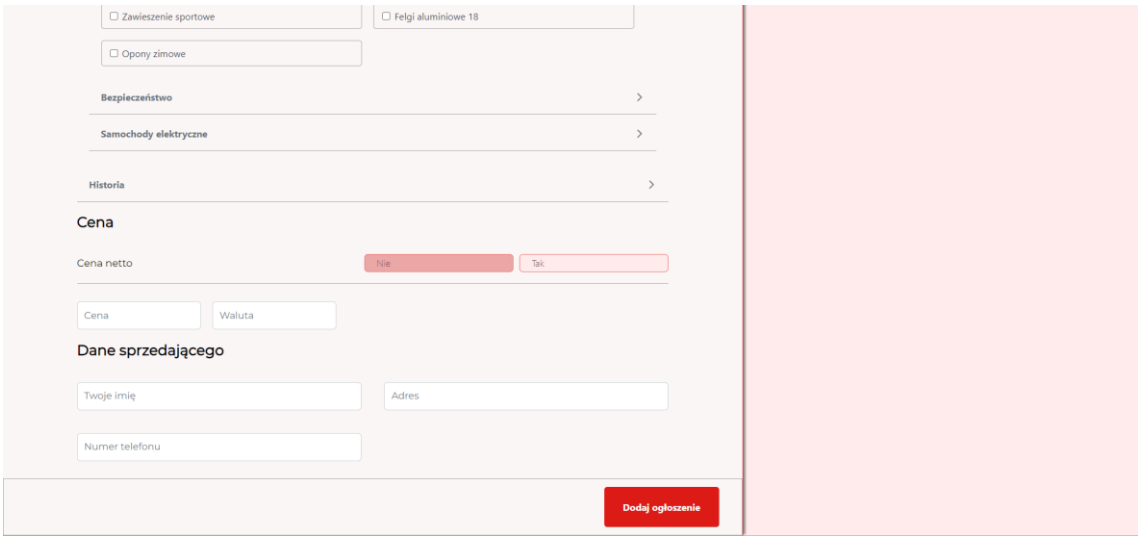
Rysunek 3.5: Ekran tworzenia ogłoszenia cz.2



Rysunek 3.6: Ekran tworzenia ogłoszenia cz.3



Rysunek 3.7: Ekran tworzenia ogłoszenia cz.4



Rysunek 3.8: Ekran tworzenia ogłoszenia cz.5

4 WYKORZYSTANE TECHNOLOGIE

Projekt informatyczny wymaga złożonego procesu wyboru technologii, który zależy od wielu czynników, takich jak znajomość rozwiązań czy użyteczność. Pojęcie technologii zawiera w sobie szeroki zakres narzędzi i metod dostępnych dla programisty. Do tego pojęcia zalicza się język programowania, biblioteki, frameworki, narzędzia do obsługi baz danych, testowania, konteneryzowania czy wdrażania. Z perspektywy budowy aplikacji, technologie są istotne w kwestii przepływu informacji i doskonalenia procesów. Umożliwiają użytkownikowi uzyskanie łatwego dostępu do danych odpowiedniego typu, co bezpośrednio wpływa na relacje z klientem oraz odbiór aplikacji przez użytkownika. Obecne technologie funkcjonujące w branży IT znacząco pomagają w pracy programistów i pozwalają na szybkie wdrażanie opracowanych koncepcji.

4.1 WYBRANE JĘZYKI

W projekcie wykorzystano dwa języki programowania: *C#* oraz *Typescript*, a także język zapytań SQL.

C# jest prostym, współczesnym, obiektowym językiem programowania zapewniającym bezpieczeństwo typów. Korzenie *C#* sięgają do rodziny języków C i z tego powodu jest podobny do języków *C, C++*, czy *Java*. Zapewnia wsparcie dla programowania opartego o komponenty. Współczesne oprogramowanie jest oparte na komponentach wformie samowystarczalnych i samo opisujących się paczek z funkcjonalnościami. *C#* wyposażony jest w właściwości pomagające mu tworzyć solidne i wytrzymałe aplikacje. Zbieranie śmieci (ang. *garbage collection*) automatycznie zwalnia pamięć zajętą przez nieużywane obiekty. Obsługa wyjątków pozwala na ustrukturyzowane podejście do detekcji błędów. Bezpieczeństwo typów sprawia, że błędy polegające na odczytywaniu wartości z niezainicjonowanych zmiennych są niemożliwe do wystąpienia. Wszystkie typy w *C#* (z typami prymitywnymi, takimi jak *int* i *double* włącznie) dziedziczą z jednego typu *object*. Powoduje to, że wszystkie typy dzielą ten sam zbiór operacji. Dodatkowo język ten zezwala na definiowanie przez użytkownika typów referencyjnych oraz wartości. [4]

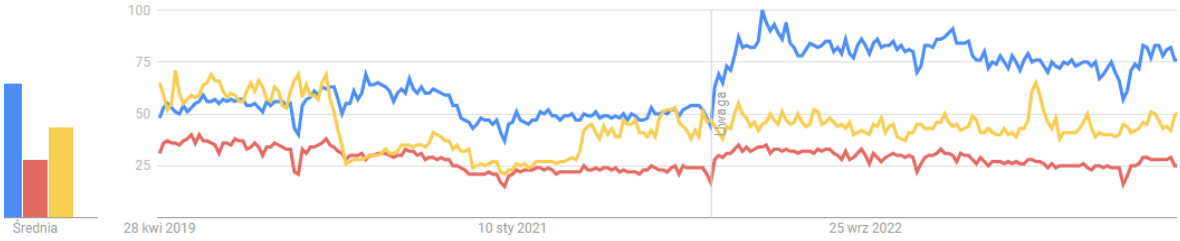
Typescript jest nakładką na język programowania *Javascript*. Powoduje to, że każdy poprawny program *Javascript* będzie też poprawny w *Typescript* (przynajmniej w większości przypadków). Dostarcza on mechanizmy strukturyzujące kod. Dodaje obiekty oparte o klasy, interfejsy i moduły. Te cechy pozwolą na strukturyzowanie kodu w o wiele lepszy sposób. *Typescript* sprawia, że kod staje się łatwiejszy w utrzymaniu oraz bardziej skalowalny poprzez trzymanie się najlepszych zorientowanych obiektowo zasad i praktyk. [5]

Javascript to język programowania używany głównie w sieci. Większość stron internetowych korzysta z tego języka, a wszystkie współczesne przeglądarki posiadają wbudowany interpreter *Javascript*. W ciągu ostatniej dekady, dzięki *Node.js, Javascript* zaczął być stosowany także poza przeglądarkami. Ogromny sukces *Node.js* sprawił, że *Javascript* stał się najczęściej używanym językiem programowania wśród wielu programistów. Język ten jest wysokiego poziomu, dynamiczny i interpretowany. Doskonale nadaje się do programowania obiektowego, jak i funkcyjnego. Jego składnia jest wzorowana na *Javie*, ale w przeciwieństwie do niej, *Javascript* jest językiem nietypowanym. Poza podobną nazwą, te dwa języki nie mają ze sobą wiele wspólnego. [6]

4.2 NARZĘDZIA STRONY INTERFEJSU UŻYTKOWNIKA

Rynek przepełniony jest bibliotekami do budowania aplikacji webowych. Najpopularniejszymi, bez wątpienia, są *Angular*, *React* i *Vue*. Dane przedstawione na stronie *trends.google.pl* (Rysunek 4.1) pokazują, że *React* i *Vue* na przestrzeni ostatnich lat cieszą

się większą popularnością niż *Angular*. Ze względu na styczność z tym frameworkiem w pracy zawodowej zostanie on też poddany analizie w tej pracy.



Rysunek 4.1: Porównanie zainteresowania *Angularem* (linia czerwona), *Reactem* (linia niebieska) oraz *Vue* (linia żółta)

4.2.1 *REACT*JS

React to popularna biblioteka służąca do tworzenia interfejsów graficznych użytkownika. Została stworzona przez firmę Facebook, która musiała zmierzyć się z problemami dotyczącymi wysoko skalowalnych aplikacji opartych na dużych ilościach danych. *React* został wydany w 2013 roku.

Biblioteka nie dostarcza wielu narzędzi, które zwykle są dostępne w tradycyjnych frameworkach *Javascript*. Daje to deweloperom swobodę w wyborze i importowaniu dodatkowych narzędzi oraz bibliotek według własnych potrzeb. [8]

4.2.2 *ANGULAR*

Angular to platforma programistyczna napisana w języku *Typescript*. Składa się z mniejszych podsystemów, interfejsu wiersza poleceń oraz dużego zestawu własnych bibliotek. Umożliwia tworzenie skalowalnych aplikacji webowych.

Został stworzony przez zespół Google'a. Pierwsza wersja nosząca nazwę *AngularJS* została wypuszczona w 2012 roku. *Javascript* był językiem, z którego korzystała ta wersja. W 2016 roku zespół *Angulara* połączył siły z zespołem *Typescriptu* Microsoftu. Ten krok wprowadził do frameworku *AngularJS* język *Typescript*.

Ten framework opiera się na najnowocześniejszych standardach webowych, a także wspiera wszystkie najpopularniejsze przeglądarki. [7]

4.2.3 *VUE*

Vue to progresywny framework służący do budowania interfejsów użytkownika. Progresywny oznacza, że ma architektoniczne zalety frameworku, ale także szybkość i modularność biblioteki, jako że funkcjonalności mogą być implementowane narastająco. W praktyce oznacza to, że to narzędzie określa pewne modele do budowania aplikacji, ale w tym samym czasie, zawsze pozwala na rozpoczęcie prac powoli i rozbudowanie się w miarę potrzeb. [9]

4.2.4 BIBLIOTEKI DO BUDOWANIA KOMPONENTÓW

Każda z aplikacji korzystała z dodatkowej biblioteki służącej poprawie wizualnej sekcji wyświetlanych na stronie. Biblioteka jest autorstwa tej samej firmy, jednak dla każdego frameworku ma inną nazwę. Dla *Angulara* biblioteka nazywa się *PrimeNg*, dla *Reacta* – *PrimeReact*, a dla *Vue* - *PrimeVue*. Te biblioteki są ogromnym zbiorem natywnych komponentów o otwartym kodzie źródłowym. W projekcie skorzystano z komponentów takich jak:

- *InputText*,
- *Button*,
- *Toast*,

- *Message,*
- *SelectButton,*
- *Accordion,*
- *AccordionTab,*

Instalacja bibliotek *prime* jest bardzo prosta. Wystarczy w otwartym terminalu w folderze z projektem wpisać komendę: *npm install <<nazwa biblioteki odpowiedniej do frameworku>>*.

4.3 NARZĘDZIA STRONY SERWEROWEJ

Do zbudowania strony serwerowej wybrany został język *C#* z frameworkiem *ASP.NET Core*. Wybór tego narzędzia był spowodowany chęcią poznania nowej metody alternatywnej do frameworku *Springboot*, którym posługuje się język Java.

Aplikacja serwerowa, by móc poprawnie wykonywać swoje zadania, korzystała z modułów *Web API* oraz *MVC Controller*.

4.3.1 ASP.NET CORE

ASP.NET Core to multi-platformowy framework, który można użyć do zbudowania aplikacji webowych. Używając *ASP.NET Core* można napisać aplikacje renderowane na serwerze lub serwer *backendowy*. To narzędzie dostarcza strukturę, pomocnicze funkcje a także framework do budowania aplikacji, co znacząco oszczędza czas pisania kodu. [10]

4.3.2 MODEL-VIEW-CONTROLLER (MVC)

MVC to elegancki sposób podziału odpowiedzialności w aplikacji i idealnie sprawdza się w aplikacjach webowych. MVC oznacza:

- **(M) Modele**: to są klasy reprezentujące model domenowy. Większość z nich odpowiada danym przechowywanym w bazie danych,
- **(V) Widoki**: to dynamicznie wygenerowany kod strony HTML,
- **(C) Kontroler**: to klasa zarządzająca interakcją pomiędzy widokiem i modelem.

4.3.3 WEB API

ASP.NET Web API może być użyte do zbudowania serwisów opartych o HTTP. Z tego powodu może być łatwo wykorzystane przez każdą z technologii frontendowych. Swój debiut narzędzie zaliczyło w 2012 roku, gdzie posiadało najbardziej podstawowe potrzeby dotyczące serwisów opartych o HTTP. [11]

4.4 NARZĘDZIA BAZY DANYCH

Baza danych to zorganizowany zbiór danych, które są zapisywane i odczytywane elektronicznie. Projektowanie baz danych obejmuje zarówno techniki formalne, jak i praktyczne aspekty, takie jak modelowanie danych, efektywne przedstawianie i przechowywanie danych, języki zapytań, a także bezpieczeństwo i prywatność poufnych informacji.

4.4.1 *POSTGRESQL*

Istotą każdej aplikacji, która wchodzi w interakcje z użytkownikiem, jest wyświetlanie, pobieranie, usuwanie i aktualizowanie odpowiednich informacji w interfejsie użytkownika (UI). Nie byłoby to możliwe bez narzędzia do przechowywania dużych ilości danych.

PostgreSQL to system zarządzania relacyjnymi bazami danych (ang. *object-relational-database management system, ORDBMS*), oparty na *Postgres*, opracowany na Uniwersytecie Kalifornijskim w Berkeley na wydziale informatyki. *PostgreSQL* jest

narzędziem o otwartym kodzie źródłowym. Wspiera wiele standardów SQL, oferując jednocześnie liczne dodatkowe funkcje, takie jak:

- Rozbudowane zapytania (query),
- Klucze obce,
- Wyzwalacze,
- Aktualizowalne widoki.

Dodatkowo, *PostgreSQL* można rozszerzać o nowe elementy, w tym:

- Typy danych,
- Funkcje,
- Operatory,
- Metody indeksowe,
- Języki proceduralne. [12]

4.4.2 *REDIS*

Bazy danych *no-sql* są idealnym rozwiązaniem problemu *cachowania* danych w aplikacji. W przypadku tego projektu zapamiętane zostają informacje na temat niedawno wyświetlonych szczegółów ogłoszeń.

Redis jest bazą danych w pamięci oferującą wysoką wydajność, replikacje i unikalny model danych. Wszystko po to, by stworzyć idealną platformę do rozwiązywania problemów. *Redis* wspiera pięć różnych typów danych. Wiele problemów może zostać rozwiązane z użyciem tego narzędzia. *Redis* pozwala na rozwiązywanie problemów bez konieczności nadmiernego wysiłku umysłowego, tak jak przy innych bazach danych. [13]

4.5 NARZĘDZIA KONTENERYZACJI

W obecnych czasach, by jak najmocniej zmniejszyć zużywanie zasobów przez uruchomione aplikacje używa się narzędzi konteneryzujących, takich jak *Docker* i *Docker-compose*. W tej aplikacji zostały one użyte w celu skonteneryzowania baz danych oraz serwera.

4.5.1 *DOCKER*

Nie każdy użytkownik aplikacji będzie miał zainstalowane narzędzia *PostgreSQL*, czy *Redis*. Aby zapewnić stałe połączenie aplikacji z bazą danych użyto narzędzia *Docker*.

Docker to technologia wirtualizacji kontenerów, czyli rodzaj maszyny wirtualnej, która zajmuje bardzo mało przestrzeni dyskowej. *Docker* powstał w celu rozwiązania problemu dużej ilości pamięci zajmowanej przez tradycyjne maszyny wirtualne. Tradycyjna maszyna wirtualna obejmuje cały system operacyjny, na którym działają aplikacje, co powoduje problemy z szybkością i wydajnością. *Docker* tworzy lekkie i elastyczne środowisko, w którym kontenery uruchamiają się szybko i można uruchomić ich wiele jednocześnie, co zapewnia dużą skalowalność.

Jednym z przypadków, w których deweloperzy chcą używać *Dockera*, jest ciągła integracja i ciągłe wdrażanie (CI/CD). Dzięki lekkości *Docker* umożliwia programistom tworzenie wielu kontenerów na ich maszynach, które mogą być kopiami środowisk produkcyjnych. [14]

4.5.2 *DOCKER COMPOSE*

Docker Compose to narzędzie definiowania i uruchamiania aplikacji wielokontenerowych. Konfiguracja następuje za pomocą plików YAML oraz przez konsole. Komendy, które oferuje to narzędzie, pozwalają na wykonywanie operacji na kontenerach uruchomionych *Docker Composem*. To oprogramowanie pozwala między innymi na

uruchomienie kompleksowych wielokontenerowych aplikacji na jednej maszynie, pozwala na zachowanie danych przy zmianie aplikacji, pozwala na zaktualizowanie wersji aplikacji, umożliwia ponowne użycie konfiguracji, czy rozlokowanie aplikacji na środowiskach produkcyjnych. [15]

Docker Compose w tym projekcie został użyty ze względu na ułatwienie pracy nad projektem na wielu komputerach. Uruchamianie dwóch baz danych i serwera byłoby mocno obciążające dla komputera, a ponieważ zmiany na serwerze były wprowadzane tylko w momencie implementacji pierwszej aplikacji webowej (aplikacji pisanej w frameworku *Angular*), to nie było sensu uruchamiać wymagającego dużej ilości zasobów IDE w celu uruchomienia serwera. Prosta konteneryzacja baz danych oraz serwera pozwoliła na bardziej komfortowe programowanie aplikacji ze względu na zwolnienie zasobów komputera.

5 IMPLEMENTACJA APLIKACJI

Implementacja i testy rozwiązania są końcowymi etapami projektu informatycznego. Implementacja polega na przekształceniu założeń projektowych i dokumentacji w działający kod, który swoim prawidłowym funkcjonowaniem potwierdza zakończenie prac nad projektem. Testy natomiast weryfikują, czy rozwiązanie spełnia wszystkie wymagania i działa zgodnie z oczekiwaniami, zapewniając jego wysoką jakość i niezawodność przed wdrożeniem.

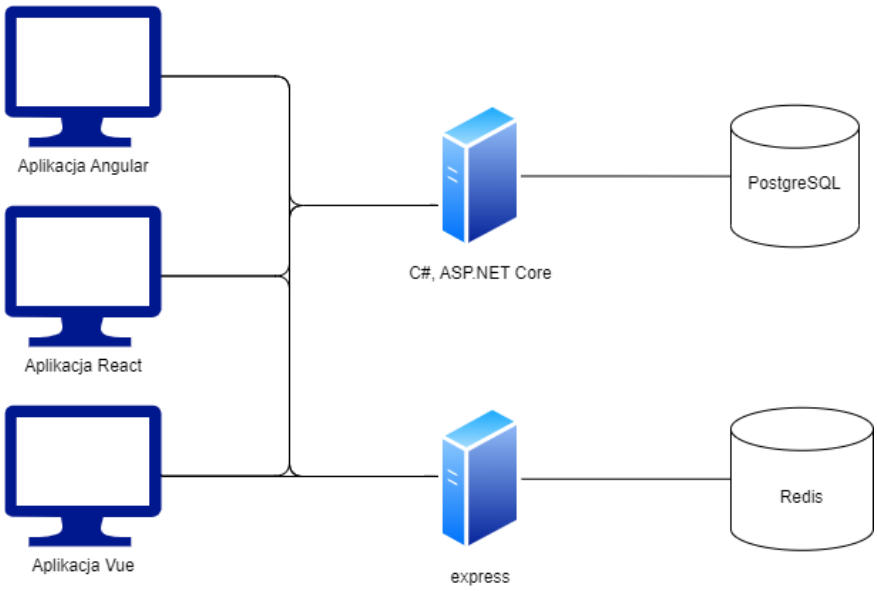
5.1 ŚRODOWISKA I NARZĘDZIA PROGRAMISTYCZNE

Aplikacja została napisana w dwóch środowiskach programistycznych. *JetBrains Rider* w wersji 2023.3.3 oraz *Visual Studio Code*. Ten pierwszy używany jest dla języka *C#,* a dzięki licencji Politechniki Wrocławskiej został zainstalowany w wersji *Ultimate*. Środowisko to pozwala na zarządzanie paczkami w prosty sposób dzięki narzędziu *NuGet*. Przewidywanie, jakie zmienne lub funkcje chce użyć programista, a także możliwość dodania wielu pomocnych pluginów, stanowi znaczącą zaletę tego narzędzia. W *Visual Studio Code* napisane zostały aplikacje w frameworkach *Angular*, *React* oraz *Vue*. Zaletą tego narzędzia jest jego mały rozmiar, co sprzyja pracy przy kilku otwartych oknach jednocześnie. *VS Code* można spersonalizować na mnóstwo sposobów. Umożliwia instalacje wielu rozszerzeń i jest obecnie najpopularniejszym, bezpłatnym środowiskiem programistycznym.

Oprócz narzędzi, w których możliwe było pisanie kodu, zostały użyte również *Git, GitHub* oraz *Docker Desktop*. Pierwsze dwa pozwoliły na proste zarządzanie kodem i jego synchronizacje na urządzeniach, na których wykonywane były prace nad projektem. *Docker Desktop* to aplikacja okienkowa stworzona dla narzędzia *Docker*. Ułatwia ona pracę z kontenerami, a także udostępnia prosty sposób wykonywania poleceń w konsoli danego kontenera.

5.2 ARCHITEKTURA APLIKACJI

Architektura aplikacji otoauto przedstawiona została na rysunku 5.1. W projekcie są trzy aplikacje webowe zbudowane za pomocą frameworków *Angular*, *React* i *Vue*. Dla tych aplikacji powstał serwer w języku *C#*, komunikujący się z bazą danych *Postgresql* oraz mniejszy serwer express służący do komunikacji aplikacji z bazą cachującą *Redis*.



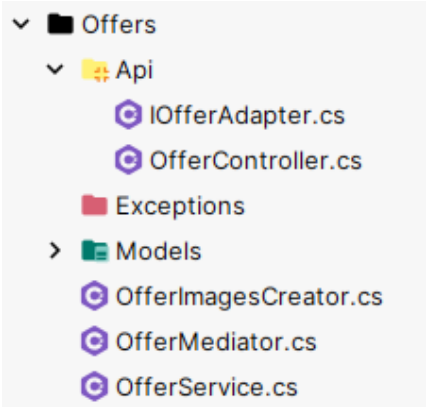
Rysunek 5.1: Architektura aplikacji otoauto

5.2.1 BACKEND

Struktura plików serwera została podzielona na foldery:

- *Properties* – pakiet przechowujący konfigurację uruchomieniową,
- *DbContexts* – pakiet, w którym znajduje się klasa tworząca konteksty dla tabel w bazie danych,
- *Entities* – pakiet, w którym znajdują się klasy odzwierciedlające tabele w bazie danych,
- *Logic* – najważniejszy pakiet strony serwerowej, znajdują się w nim pakiety przechowujące logikę każdego aspektu aplikacji:
  - *Dealers* – służące do obsługi logiki sprzedawców,
  - *Offers* – zajmujące się logiką tworzenia, aktualizowania, usuwania ofert,
  - *Vehicles* – służące do zapisywania danych o pojazdach,
  - *Others* – pakiet przechowujący logikę przypisywania wyposażenia do pojazdów, a także pobierania danych z tabel, które są za małe, by miało sens tworzenie dla nich osobnego folderu oraz klas,
- *Profiles* – pakiet, w którym znajduje się klasa definiujące *mappery* pomiędzy konkretnymi klasami,
- *Repository* – pakiet przechowujący repozytoria dla każdej tabeli w bazie danych. Repozytorium to klasa, która dzięki kontekstowi może pobrać lub zapisać dane w bazie danych,
- *Utils* – pakiet przechowujący klasy, najczęściej z metodami statycznymi, które nie należą do żadnej klasy w pakiecie *Logic*, a mogą być użyte w dowolnym miejscu w aplikacji.

W każdym pakiecie w folderze *Logic* znajdują się foldery *Api*, *Exceptions* oraz *Models*. Oprócz nich znajduje się także klasa z sufiksem *Mediator*. Na przykładzie pakietu *Offers* (Rysunek 5.2) można zauważyć, że oprócz tej klasy, jest w nim również klasa *Service* oraz *ImageCreator*.



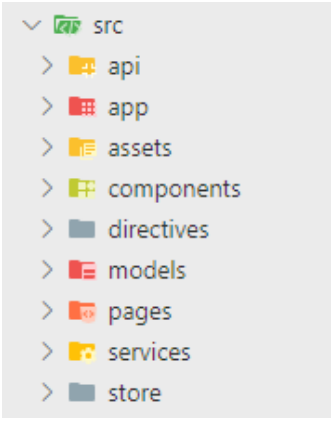
Rysunek 5.2: Pakiet Offers

W folderze *Api* znajduje się interfejs *Adapter* definiujący metody klasy *Mediator* oraz klasa *Controller*, która wystawia *endpointy*, z którymi komunikuje się aplikacja webowa.

5.2.2 FRONTEND

Struktura plików aplikacji webowej zostanie omówiona na przykładzie frameworku *Angular*. Pozostałe frameworki mają prawie identyczną strukturę z drobnymi różnicami w nazewnictwie niektórych folderów. W katalogu *src* znajdują się foldery:

- *Api* – w tym folderze znajdują się dwa katalogi: pierwszy zawiera klasy, które używając klasy *HttpClient* wysyłają zapytania do serwera, a drugi interfejsy, których obiekty są przekazywane do backendu,
- *App* – folder zawierający pliki komponentu–rodzica, plik głównego modułu aplikacji, a także plik określający ścieżki (ang. „*route*") podstron,
- *Assets* – folder zawierający obrazy i grafiki (pliki z rozszerzeniem .jpg, .png, .svg),
- *Components* – folder zawierający komponenty budujące widoki aplikacji,
- *Directives* – folder zawierający dyrektywy używane w komponentach,
- *Models* – folder zawierający interfejsy definiujące obiekty używane w aplikacji,
- *Pages* – folder zawierający komponenty-kontenery. Z nich zbudowane są podstrony aplikacji,
- *Services* – folder zawierający klasy pomocnicze z metodami wykonującymi logikę biznesową,
- *Store* – folder, w którym znajdują się klasy przechowujące stan (ang. „*state*") aplikacji.



Rysunek 5.3: Podział frontendu na foldery

5.3 IMPLEMENTACJA I KONTENERYZACJA BAZY DANYCH

Baza danych, by nie obciążać hosta instalacją wszystkich potrzebnych narzędzi, została postawiona za pomocą *Dockera*. By ułatwić pracę zainstalowana została aplikacja *Docker Desktop*. W projekcie został utworzony folder *Deployment/Database*, a w nim stworzony plik *docker-compose.yaml*. W tym pliku skonfigurowano kontener bazy danych *PostgreSQL* oraz *Redis*.

W tagu *services* zostały stworzone kontenery, w których ustawianych jest kilka zmiennych:

- *image* – definiuje oficjalny *Dockerowy* obraz, którego odpowiednia wersja zostanie pobrana ze strony *https://hub.docker.com/*. *Docker Image* to plik zawierający kod, którego wykonanie zbuduje kontener bazy danych,
- *environment* – w tej sekcji ustawiane zostają zmienne środowiskowe,
- *volumes* – aby zapewnić trwałość danych w kontenerze, należy użyć wolumenów. Wolumeny są tworzone i zarządzane przez *Docker*a, a każdy nowy wolumen jest zapisywany na dysku hosta. Oznacza to, że w przypadku usunięcia kontenera, dane przechowywane w wolumenie pozostaną nienaruszone. Wolumeny zapewniają niezawodność i bezpieczeństwo danych, umożliwiając ich trwałe przechowywanie niezależnie od stanu kontenera,

- *ports* – pierwszy adres oznacza, na którym porcie dostępna będzie baza danych na maszynie lokalnej. Drugi oznacza port po stronie kontenera,
- *networks* – odnosi się do sieci utworzonej w sekcji networks, umożliwiając konfiguracje połączeń sieciowych między kontenerami,
- *restart* – określa sposób ponownego uruchomienia kontenera w przypadku jego awarii lub zakończenia działania, zapewniając ciągłość działania aplikacji.

Poniżej przedstawiona została zawartość pliku *docker-compose.yaml* (Listing 5.1) definiująca kontenery baz danych.

```
postgres:
   image: postgres:15.4
   restart: always
   environment:
     POSTGRES_USER: dominik
     POSTGRES_PASSWORD: 12345
     PGDATA: /data/postgres
     POSTGRES_DB: oto_auto
   volumes:
     - postgres:/data/postgres
     -  D:\Studia\Magisterka\Aplikacja\Deployment\Database\SQL_Scripts
        :/var/SQL_Scripts
     - D:\Studia\Magisterka\Aplikacja\Deployment\Database\initdb.sh
        :/var/SQL_Scripts/initdb.sh
   ports:
     - "8090:5432"

redis:
   image: redis:7.2.4
   restart: always
   command: redis-server --save 20 1 --loglevel warning --requirepass
12345
   volumes:
     - redis:/data/redis
   ports:
     - "6379:6379"
```

Listing 5.1: Definicja kontenerów baz danych

W sekcji *volumes* głównej bazy danych dodane zostały dwa wiersze, które udostępniają kontenerowi folder z plikami zawierającymi skrypty, tworzące tabele i uzupełniające je danymi. Udostępniony jest także plik o rozszerzeniu *.sh* zawierający skrypt wykonujący komendy znajdujące się w plikach w folderze *SQL_Scripts* w kontenerze *Postgres*.

Listing 5.2 przedstawia fragment skryptu *V1__CreateTables.sql*. Przedstawione na nim zostały skrypty tworzące tabelę *Offer* oraz *Vehicle_Image.*

```
CREATE TABLE Offer (
  id              SERIAL NOT NULL,
  name            varchar(50) NOT NULL,
  creation_date   date NOT NULL,
  expiration_date date NOT NULL,
  price           varchar(20) NOT NULL,
  currency        varchar(5) NOT NULL,
  description     varchar(300),
  Dealer_id       int4 NOT NULL,
  Vehicle_id      int4 NOT NULL,
  PRIMARY KEY (id));
CREATE TABLE Vehicle_Image (
  id SERIAL NOT NULL,
  path_to_image varchar(300) NOT NULL,
  is_main_image boolean NOT NULL,
  offer_id int4 NOT NULL,
PRIMARY KEY (id));
```

Listing 5.2: Fragment skryptu tworzącego tabele.

```
#!/bin/bash
SCRIPTS=(
  "V1__CreateTables.sql"
  "V2__BodyType.sql"
  "V3__VehicleType.sql"
  "V4__TransmissionType.sql"
  "V5__FuelType.sql"
  "V6__DriveType.sql"
  "V7__CarStatus.sql"
  "V8__EquipmentType.sql"
  "V9__Equipment.sql"
  "V10__Dealer.sql"
  "V11__Vehicle.sql"
  "V12__Offer.sql"
  "V13__VehicleImages.sql"
  "V14__Vehicle_Equipment.sql"
)

POSTGRES_USER="dominik"
POSTGRES_DB="oto_auto"

for SCRIPT in "${SCRIPTS[@]}"; do
  psql -U $POSTGRES_USER -d $POSTGRES_DB -a -f ./$SCRIPT
done

tail -f /dev/null
```

Listing 5.3: Skrypt wykonujący komendy SQL w plikach znajdujących się w folderze SQL_Scripts.

Listing 5.3 przedstawia skrypt wykonujący komendy w plikach w folderze *SQL_Scripts*. Jest to prosty program, w którym zdefiniowana jest lista z nazwami plików oraz zmiennymi wymaganymi do połączenia się z odpowiednią bazą danych w kontenerze *Postgres*. Pętla w tym programie iteruje po nazwach plików po kolei wykonując skrypty w nich zawarte, przy użyciu komendy *psql*.

5.4 KONTENERYZACJA SERWERA

Podobnie, jak bazy danych, konteneryzacja serwera wymagała zdefiniowania kontenera serwera. W tym celu wymagane jest stworzenie pliku *Dockerfile* (Listing 5.4), w którym zdefiniowany został obraz *Dockerowy backendu*. Składa się z dwóch etapów:

- etapu budującego projekt, którego wynikiem jest plik *oto-auto-c-sharp-server.dll*,
- etapu uruchamiającego ten plik.

Po stworzeniu pliku *Dockerfile* wykonane zostały komendy *Docker build*, *Docker tag*, *Docker push*. Zbudowały one obraz, odpowiednio go otagowały, a następnie udostępniły go na repozytorium na stronie *hub.Docker.com*.

```
FROM mcr.microsoft.com/dotnet/sdk:7.0-jammy AS build-stage

WORKDIR /source

COPY oto-auto-c-sharp-server.csproj .
RUN dotnet restore

COPY . .
RUN dotnet publish -c Release -o /app --self-contained false

FROM mcr.microsoft.com/dotnet/nightly/sdk:7.0-jammy
WORKDIR /app

COPY --from=build-stage /app .
EXPOSE 5000
ENTRYPOINT [ "dotnet", "oto-auto-c-sharp-server.dll"]
```

Listing 5.4: *Docker*file aplikacji serwerowej.

W pliku *docker-compose.yaml* dodana została sekcja *server* (Listing 5.5). Obraz pobrany został z repozytorium na *dockerhubie*. Jednym z zadań serwera jest zapisywanie zdjęć nowych ofert. Ponieważ zdjęcia te powinny być przechowywane na dysku maszyny, a nie w kontenerze, stworzono wolumin, który łączy folder na dysku z folderem, do którego zapisywane są obrazy.

```
  server:
    image: tloku/oto-auto-server:1.0
    restart: always
    volumes:
      - D:\Studia\Magisterka\Dodatki\car-images-
dataset\Cars_Dataset:/var/cars_dataset
    ports:
      - "5252:5000"
```

Listing 5.5: Definicja kontenera serwera

5.5 IMPLEMENTACJA SERWERA

Klasa *OfferController* (Listing 5.6, Rysunek 5.2), by poprawnie działała, musi dziedziczyć po *ControllerBase* – klasie z pakietu *Microsoft.AspNetCore.Mvc*, a także posiadać adnotacje *ApiController* oraz *Route*, która ustawia ścieżkę do endpointów w tej klasie.

Metody kontrolerów muszą posiadać adnotacje mówiącą o tym, jaki rodzaj zapytania obsługują, np. metody, które zwracają dane, powinny mieć adnotację *HttpGet*, a te które aktualizują dane – *HttpPost*.

Folder *Exceptions* w domyśle zawiera klasy, które dziedziczą po klasie wbudowanej w *C# - Exception*. Własne klasy *Exception* mogą przekazywać więcej informacji odnośnie występującego błędu. Sama nazwa takiej klasy może szybciej nakierować programistę na źródło błędu i zmniejszyć czas potrzebny na rozwiązanie problemu.

```
[ApiController]
[Route("api/offer")]
public class OfferController: ControllerBase
{
    private readonly IOfferAdapter _offerAdapter;
    public OfferController(IOfferAdapter offerAdapter) {
        _offerAdapter = offerAdapter;
    }
    [HttpGet()]
    public async Task<ActionResult<IEnumerable<Offer>>> GetAllOffers() {
        var offers = await _offerAdapter.GetAllOffers();
        return Ok(offers);
    }
// dalsza część klasy
```

Listing 5.6: Fragment klasy OfferController

Folder *Models* zawiera klasy typu *dto*[2], które ułatwiają separację warstw aplikacji, ułatwiając przekazywanie danych między nimi. Poprawiają też wydajność w komunikacji, ponieważ przesyłane obiekty nie zawierają zbędnych informacji.

Klasy *Mediator* implementują metody zadeklarowane w interfejsach *Adapter*. Korzystają też z metod zadeklarowanych w repozytoriach, czyli *de facto* pośrednio wywołują operacje na bazie danych. Z założenia te klasy powinny korzystać z serwisów, *mapperów* i repozytoriów, by wykonać założoną operację, ale same nie powinny zawierać dodatkowych metod prywatnych.

Obiekty zapisywane w bazie danych to klasy, które są zmapowane obiektowo-relacyjnie z tabelami z bazy danych. Klasa *Offer* (Listing 5.7) zawiera wszystkie pola z odpowiadającej jej encji. Nad jej definicją wymagana jest adnotacja *Table*, która wskazuje, do której tabeli ta klasa jest mapowana. Wymagane jest oznaczenie klucza głównego adnotacją *Key*. Jednocześnie ta sama zmienna, jeśli zamierzamy tworzyć nowe obiekty i zapisywać je w bazie, powinna mieć adnotację *DatabaseGenerated*, oznaczającą sposób generowania wartości klucza głównego. Każda ze zmiennych, wykluczając te oznaczone jako wirtualne, powinna mieć adnotacje *Column*, by serwer poprawnie mapował wartości tych zmiennych klasy z tymi w tabeli. Encja *Offer* zawiera w sobie trzy klucze obce. Zmienne z oznaczeniem *virtual* wskazują na obiekt, który może być pobrany z bazy danych przy pobieraniu klasy *Offer*. Jest to możliwe, gdy w klasie *OfferRepository* przy pobieraniu tej encji z kontekstu dodamy metodę *Include* (Listing 5.8), w której wskażemy na obiekt klasy Offer, który również chcemy pobrać.

W przypadku metody *GetOfferWithVehicleByOfferId* pobrana zostanie oferta o id przekazanym w parametrze tej funkcji. Oferta ta będzie miała uzupełnione pola *Vehicle* oraz *VehicleImages*.

[2] Dto – (z ang. „Data Transfer Object”) obiekt transferu danych

Poprawne działanie repozytoriów wymaga zadeklarowania ich klas w głównym pliku uruchamiającym aplikację – Program.cs (Listing 5.9). Do serwisów aplikacji zostają dodane metodą *AddScoped* interfejs repozytorium oraz dziedzicząca po nim klasa.

```
[Table("offer")]
public class Offer
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    [Column("id")]
    public int Id { get; set; }
    [Column("name")]
    public string Name { get; set; }
    [Column("creation_date")]
    public DateTime CreationDate { get; set; }
    [Column("expiration_date")]
    public DateTime ExpirationDate { get; set; }
    [Column("price")]
    public string Price { get; set; }
    [Column("currency")]
    public string Currency { get; set; }
    [Column("description")]
    public string Description { get; set; }
    [Column("vehicle_id")]
    public int VehicleId { get; set; }
    public virtual Vehicle Vehicle { get; set; }
    [Column("dealer_id")]
    public int DealerId { get; set; }
    public virtual Dealer Dealer { get; set; }
    public virtual ICollection<VehicleImage> VehicleImages { get; } = new
List<VehicleImage>();
    public Offer() {}
//Pozostały kod klasy Offer
```

Listing 5.7 Klasa Offer

```
class OfferRepository: IOfferRepository
{
        //reszta klasy
public async Task<Offer?> GetOfferWithVehicleByOfferId(int offerId)
{
    return await _context.Offer
        .Include(o => o.Vehicle)
        .Include(o => o.VehicleImages)
        .Where(o => o.Id == offerId)
        .FirstOrDefaultAsync();
}
        //reszta klasy OfferRepository
}
```

Listing 5.8: Metoda z repozytorium oferty pobierająca ofertę z pojazdem po id

Wcześniej wspomniany kontekst bazy danych, z którego korzystają repozytoria, ustawiany jest w pliku *OtoAutoContext.cs* (Listing 5.10). Zawiera on klasę generyczną *OtoAutoContext* dziedziczącą po klasie *DbContext* z pakietu

Microsoft.EntityFrameworkCore. Jej pola to zmienne typu *DbSet*, które odpowiadają każdej z tabel w bazie danych.

```
builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseNpgsql(builder.Configuration.GetConnectionString("Connection"))
);

builder.Services.AddScoped<IVehicleRepository, VehicleRepository>();
builder.Services.AddScoped<IOfferRepository, OfferRepository>();
builder.Services.AddScoped<IBodyTypeRepository, BodyTypeRepository>();
builder.Services.AddScoped<ITransmissionTypeRepository,
TransmissionTypeRepository>();
builder.Services.AddScoped<IVehicleTypeRepository,
VehicleTypeRepository>();
builder.Services.AddScoped<IEquipmentRepository, EquipmentRepository>();
//reszta konfiguracji
```

Listing 5.9: Fragment kodu pliku Program.cs

W tym samym pliku znajduje się definicja klasy *ApplicationContext*, która dziedziczy po klasie omówionej powyżej. Ta klasa jest wstrzykiwana do repozytoriów i z niej możliwe jest pobieranie i zapisywanie danych konkretnych tabel. Jednak, by kontekst miał informacje na temat bazy, do której ma się połączyć, musi on być dodany do serwisów aplikacji metodą *AddDbContext* (Listing 5.9).

```
public class OtoAutoContext<T> : DbContext where T : DbContext
{
    public OtoAutoContext(DbContextOptions<T> options) : base(options)
    { }
    public DbSet<Offer> Offer { get; set; } = null!;
    public DbSet<Vehicle> Vehicle { get; set; } = null!;
    public DbSet<BodyType> BodyType { get; set; } = null!;
    public DbSet<FuelType> FuelType { get; set; } = null!;
    public DbSet<TransmissionType> TransmissionType { get; set; } = null!;
    public DbSet<Dealer?> Dealer { get; set; } = null!;
    public DbSet<EquipmentType> EquipmentType { get; set; } = null!;
    public DbSet<Equipment> Equipment { get; set; } = null!;
    public DbSet<CarStatus> CarStatus { get; set; } = null!;
    public DbSet<DriveType> DriveType { get; set; } = null!;
    public DbSet<VehicleType> VehicleType { get; set; } = null!;
    public DbSet<VehicleImage> VehicleImage { get; set; } = null!;
    public DbSet<VehicleEquipment> VehicleEquipment { get; set; } = null!;
}


public class ApplicationContext : OtoAutoContext<ApplicationContext>
{
    public ApplicationContext(DbContextOptions<ApplicationContext> options)
: base(options) { }
}
```

Listing 5.10: Zawartość pliku OtoAutoContext

W środku tej metody wywoływana jest metoda *UseNpgsql*, która za parametr przyjmuje ciąg znaków definiujących połączenie do bazy danych. Ciąg zawiera dane takie jak ip hosta, port, login, hasło i nazwa bazy. Definicja zmiennej *Connection* znajduje się w pliku konfiguracyjnym *appsettings.json*.

```
"ConnectionStrings":{
    "Connection":
"Host=postgres;Port=5432;Database=oto_auto;Username=dominik;Password=12345;
"}
```

Listing 5.11: Definicja połączenia z bazą danych w pliku appsettings.json

5.6 IMPLEMENTACJA INTERFEJSU UŻYTKOWNIKA

Podział folderów w projekcie strony internetowej został opisany w punkcie **Błąd! Nie można odnaleźć źródła odwołania.**. Na przykładzie folderu *pages/offer-view* opisana zostanie implementacja logiki tworzącej aplikację internetową dla każdego z frameworków. Dokładniejsza analiza tych narzędzi znajduje się w rozdziale 6.

W większości przypadków *Angular* do stworzenia komponentu wymaga utworzenia trzech plików (Rysunek 5.4). Jeden z nich definiuje logikę, drugi szablon, a trzeci styl komponentu.

Rysunek 5.4: Komponent offer-view (*Angular*)

Komponent *OfferView* jest rodzicem komponentów budujących widok szczegółów oferty. Jego logika znajduje się w metodzie *ngOnInit()*. Jest to metoda wbudowana w ten framework, która jest wywoływana w momencie inicjalizacji komponentu. Aplikacje webowe powinny dzielić komponenty na „mądre", czyli te, które wykonują skomplikowaną logikę oraz „głupie", które wyświetlają otrzymane dane. Ze względu na fakt, że *OfferView* będzie wyświetlać informacje na temat oferty, zostanie on zakwalifikowany jako „głupi" komponent, wywołujący jedynie metodę *dispatch*. Efektem tej metody będzie wywołanie funkcji pobierającej szczegóły otwartej oferty.

```
@Component({
    selector: 'offer-view-component',
    templateUrl: './offer-view.component.html',
    styleUrls: ['./offer-view.component.css'],
})
export class OfferViewComponent implements OnInit {
    constructor(
        private _store: Store,
        private _activatedRoute: ActivatedRoute
    ) {}
    ngOnInit(): void {
        const offerId: string | null =
this._activatedRoute.snapshot.paramMap.get('offerId');
        this._store.dispatch(new GetOfferById(+offerId!!))
    }
}
```

Listing 5.12: Komponent OfferView (*Angular*)

Store to globalny manager stanu biblioteki NGXS, który wysyła akcje, a także dostarcza sposób na pobranie poszczególnych danych z globalnego stanu aplikacji. [16] Poprawne działanie tego managera stanu wymaga stworzenia:

- klasy, która będzie odpowiadała akcji (klasa *GetOfferById* z listingu 5.12 i listingu 5.13),
- klasy, która obsługuje akcje,
- interfejsu, który definiuje stan przechowywany przez klasę obsługującą akcje.

Biblioteka NGXS wymaga, by akcja była opisana, stąd w każdej klasie akcji zdefiniowana została statyczna zmienna typu *string*. Jeśli do akcji musi zostać przekazany parametr, wymaganym jest stworzenie konstruktora, który będzie przyjmował ten publiczny atrybut.

```
export class GetOfferById {
    static readonly type: string = "[Get Offer By Id] Get Offer by id"
    constructor(public offerId: number) {}
}
```

Listing 5.13: Klasa GetOfferById zdefiniowana w pliku src/store/actions/offer-actions.ts

Stan oferty przechowuje dane dotyczące oferty wyświetlanej na stronie szczegółów oferty oraz jej zdjęcia, ofert wyświetlonych w kartach na stronie głównej oraz informacje czy baza danych *Redis* bądź serwis, który się z nią komunikuje, odpowiada.

W klasie *OfferState* oznaczonej adnotacjami *State<T>* oraz *Injectable* zdefiniowane są akcje ofert. Działanie akcji *GetOfferById* (Listing 5.14) polega na pobraniu szczegółów oferty z jednego z dwóch miejsc – serwera połączonego z bazą *PostgreSQL* lub serwera połączonego z bazą *Redis*, służącej jako pamięć podręczna. Korzystanie z drugiej bazy pozwala na kilkukrotne skrócenie czasu potrzebnego do pobrania szczegółów oferty.

Działanie metody *getOfferById* można opisać następująco:

a) Funkcja sprawdza, czy baza *Redis* odpowiada i jeśli nie, pobiera ofertę z serwera komunikującego się z bazą *PostgreSQL*.
b) Funkcja wywołuje metodę *get* z serwisu *RedisCacheService*. Ta metoda powinna pobrać ofertę zapisaną w pamięci podręcznej lub zwrócić wartość *undefined*.
c) Jeśli metoda *get* zwróciła wynik, to wywoływana jest akcja *GetCachedOfferByIdSuccess*, która zapisuje dane w stanie.
d) Jeśli metoda *get* zwróciła wartość *undefined*, to wywoływana jest metoda pobierająca ofertę z serwera.

Jeśli podczas wykonywania metoda *getOfferById* zwróci błąd, to zostanie on złapany w metodzie *catchError*, który wyświetli komunikat użytkownikowi.

Metoda *getOfferIdFromServer* używa serwisu *OfferRestService* w celu pobrania szczegółów oferty. W momencie zwrócenia wyniku jest on zapisywany w stanie po wywołaniu akcji *GetOfferByIdSuccess*. Jeśli w trakcie pobierania danych wystąpi błąd to wywołana zostanie metoda *catchError*, która wywoła akcję *GetOfferByIdFailure*.

```
@Action(GetOfferById)
getOfferById(ctx: StateContext<OfferCardComponentStateModel>, action:
GetOfferById) {
  const offerId: number = action.offerId;
  const state = ctx.getState();

  if (state.redisNotResponding) {
     return this.getOfferByIdFromServer(ctx, offerId);
  }

  return this._redisCacheService
     .get<OfferActivityComponentModel>(RedisCacheKeys.OFFER + offerId)
       .pipe(
         map((redisResponse: OfferActivityComponentModel | undefined) => {
          if (redisResponse) {
          return ctx.dispatch(new GetCachedOfferByIdSuccess(redisResponse))
          }
          return this.getOfferByIdFromServer(ctx, offerId);
          }),
         catchError(error => {
           this._messageService.add({key: 'toast', severity: 'error',
summary: 'Błąd podczas wyświetlania oferty'})
            throw error;
          })
       )
   }

private getOfferByIdFromServer(ctx:
StateContext<OfferCardComponentStateModel>, offerId: number) {
  return this._restService.getOfferById(offerId)
     .pipe(
       map((offer: OfferActivityComponentModel) => ctx.dispatch(new
GetOfferByIdSuccess(offer))),
       catchError(error => ctx.dispatch(new GetOfferByIdFailure(error)))
       )
    }
```

Listing 5.14: Akcja GetOfferById zdefiniowana w klasie OfferState

Implementacja komponentu *OfferView* w frameworku *React* (Listing 5.15) jest podobna do tej z *Angulara*. Różnicą jest to, że zamiast metody *ngOnInit* wywołującej pobranie danych oferty użyty jest w tym celu *hook*[3] *useEffect*. W tym przypadku wywołany zostanie on tylko raz podczas ładowania strony szczegółów oferty, ponieważ zmienił się parametr *params*, na który ten *hook* reaguje. Wewnątrz wywoływana jest akcja *getOfferById* za pomocą funkcji *dispatch*. Stan przetrzymywany jest w aplikacji dzięki bibliotece *Redux Toolkit*.

Redux jest biblioteką służącą do tworzenia przewidywalnego i łatwego w utrzymaniu globalnego zarządzania stanem aplikacji. Cały stan trzymany jest w obiekcie znajdującym się w jednym *store*. By móc zmienić stan tego obiektu należy stworzyć „akcję”, czyli obiekt opisujący to, co się wydarzyło, a następnie „wysłać” (ang. „*dispatch*”) tę akcję do *store*.

[3] „Hooki są to funkcje, które pozwalają „zahaczyć się” w mechanizmy stanu i cyklu życia *Reacta*, z wewnątrz komponentów funkcyjnych.” [17]

Wyspecyfikowaniem tego, w jaki sposób stan aplikacji zostaje zaktualizowany w odpowiedzi na akcję zajmują się funkcje „reduktory” (ang. „*reduce*”). [18].

```
export const OfferViewComponent: React.FC = () => {
    const toast: MutableRefObject<null> = useRef(null);
    const params = useParams()
    const offer = useSelector((state: RootState) => state.offerCard.offer);
    const dispatch = useDispatch<ThunkDispatch<RootState, undefined,
AnyAction>>();

    useEffect(() => {
        if (!params)
            return
        const id: number = parseInt(params.id!);
        dispatch(getOfferById({id, toast}))
      }, [params])

    return <>
        <Toast ref={toast} />
        {
        offer &&
            <div className="offer-view-wrapper">
                <div className="offer-view-details">
                    <div className="gallery">
                        <VehicleImagesGalleryComponent
                          vehicleImages={offer?.offerImages} />
                    </div>
                    <OfferDetailsComponent
                          vehicleAttributes={offer?.vehicleAttributes} />
                </div>

                <div className="offer-view-price">
                    <OfferViewPriceComponent offer={offer} />
                </div>
            </div>
        }
    </>
}
```

Listing 5.15: Komponent OfferView (*React*)

GetOfferById to obiekt zwrócony przez metodę *createAsyncThunk*. Metoda ta akceptuje jako parametr typ string akcji *Redux* oraz funkcję zwrotną, a zwraca obietnicę (ang. „*promise*”). Ta funkcja generuje obietnicę typu akcji cyklu życia opartego na podanym prefiksie. Zwrócony obiekt uruchomi funkcję zwrotną, a następnie wykona odpowiednie akcje cyklu życia w zależności od wartości zwróconej w obietnicy. [19]

W parametrze funkcji *createAsyncThunk* (Listing 5.16) przekazany został typ akcji „*offerCard/getOfferById*” oraz funkcja zwrotna wykonująca pobieranie szczegółów oferty z serwera lub pamięci podręcznej *Redis*. Wynik tej metody zwraca pobraną ofertę, a także uruchamia odpowiednią akcję cyklu życia.

Metoda *createSlice* to funkcja przyjmująca początkowy stan, obiekt reduktorów (lub, jak w przypadku Listingu 5.17, extra reduktorów), nazwę „slice”, automatycznie tworzy akcje kreatorów, które odpowiadają reduktorom i stanowi. [19]

```
export const getOfferById = createAsyncThunk(
    'offerCard/getOfferById',
    async ({id, toast}: {id: number, toast: any}, { getState }) => {
        const state: OfferCardComponentStateModel = getState() as
OfferCardComponentStateModel
        if (state.redisNotResponding) {
            return getOfferByIdFromServer(id)
        }
        try {
            const redisResp: AxiosResponse<OfferActivityComponentModel |
undefined> = await
RedisCacheService.get<OfferActivityComponentModel>(RedisCacheKeys.OFFER +
id);
            if (redisResp.data) {
                return redisResp.data;
            }
            const offer = await getOfferByIdFromServer(id);
            cacheOfferDetails(offer, state.redisNotResponding);
            return offer;
        } catch (error) {
            if (!state.redisNotResponding) {
                toast.current.show({severity:'error', detail: "Redis nie
odpowiada. Kolejna próba połączenia nastąpi za 5 minut"});
            }
            state.redisNotResponding = true;
            setTimeout(() => {
                state.redisNotResponding = false;
            }, 5 * 60 * 1000);
            try {
               return await getOfferByIdFromServer(id);
            } catch (e) {
                toast.current.show({severity:'error', detail:'Błąd podczas
wyświetlania oferty'});
                throw e;
            }
        }
    }
)
```

Listing 5.16: Wywołanie metody createAsyncThunk

W obiekcie *Slice* (Listing 5.17) zdefiniowano tylko jeden przypadek dla „*thunka*” *getOfferById*. Jest to akcja *fulfilled*, której skutkiem jest przypisanie danych zwróconych przez funkcję zwrotną do obiektu przechowywanego w stanie aplikacji. Obiekt *store* wymaga przekazania mu reduktorów, z tego powodu należy wyeksportować reduktor z zmiennej *offerSlice*.

Stworzenie *Store Redux* wymaga wykonanie metody *configureStore*. Zazwyczaj konfiguracja wymaga połączenia *slice* reduktorów do głównego reduktora *roota*. [19]

W przypadku omawianej aplikacji do stworzenia *store* wystarczyło podanie trzech reduktorów i wyeksportowanie stanu *roota*, a także typu *dispatch* wymaganego przez inne komponenty w celu wywołania akcji.

```
const offerSlice = createSlice({
    name: "offerCard",
    initialState: initialOfferStateValue,
    reducers: {},
    extraReducers: (builder) => {
        builder.addCase(getAwardedOffers.fulfilled, (state, { payload }) =>
{
            state.offerCardsComponent = payload
        }),
        builder.addCase(getOfferById.fulfilled, (state, { payload }) => {
            state.offer = payload!
        })
    }
})
export default offerSlice.reducer
```

Listing 5.17: Obiekt przekazany do metody createSlice

W aplikacji napisanej w frameworku *Vue* do zarządzania stanem wykorzystano bibliotekę *Vuex*. Stworzenie *store* wymaga podania modułów (Listing 5.18) do metody *createStore*. Moduł to obiekt, w którym mogą być zdefiniowane:

- obiekt przechowujący dane,
- mutacje, w których znajdują się funkcje zmieniające stan,
- akcje, w których znajdują się funkcje wywołujące odpowiednie mutacje,
- akcesory zwracające dane ze stanu.

Ostatnią operacją, jaką trzeba wykonać, by *store* działał poprawnie, jest wywołanie funkcji *use* z parametrem *store* na obiekcie *App* w pliku *main.ts*.

```
const offerCardModule = {
    state: () => (initialOfferStateValue),
    mutations: {
        getOfferById(state: OfferCardComponentStateModel, offer:
                                        OfferActivityComponentModel) {
            state.offer = offer
        }
},
    actions: {
        async getOfferById(context, id: number) {
            // Logika pobrania oferty
        }
    },
    getters: {
        offer(state: OfferCardComponentStateModel) {
            return state.offer
        }
    }
  }
```

Listing 5.18 Fragment modułu przechowującego dane ofert

5.7 IMPLEMENTACJA SERWERA KOMUNIKUJĄCEGO SIĘ Z BAZĄ DANYCH *REDIS*

Z bazą danych *nosql* komunikuje się serwer postawiony za pomocą biblioteki *express*. Zawarte są w nim dwie dodatkowe biblioteki:

- *Redis* – która jest użyta by stworzyć klienta łączącego się z bazą danych,
- *cors* – eliminująca błędy *Cross-Origin Resource Sharing*.

Serwer udostępnia dwa punkty końcowe, z których korzystają aplikacje webowe. Służą one do wywoływania metod *get* i *set* na kliencie *Redis*. Są to dwie podstawowe komendy tej bazy danych, które odpowiednio pobierają lub zapisują dane z kluczem podanym w parametrze metody.

Pierwszy punkt końcowy obsługuję metodę GET, która umożliwia pobranie danych z klienta *Redis* na podstawie klucza przekazanego w parametrze ścieżki żądania. Jeśli klucz istnieje w bazie danych, to serwer zwraca dane w formacie JSON, jeśli nie, zwracana jest wartość *null*.

Drugi punkt obsługuje metodę *POST* umożliwiającą zapisanie danych przesłanych w formacie *JSON*. Klucz i wartość pobierane są z ciała żądania. Klient *Redis* pozwala na ustawienie dodatkowych flag podczas zapisywania. Jedną z nich jest *EX*, która definiuje na jaki czas dany klucz i jego wartość są zapisane w bazie danych. W tym przypadku czas został ustawiony na 5 minut.

Serwer po uruchomieniu nasłuchuje na określonym porcie, a informacje o jego uruchomieniu oraz obsługiwanych żądaniach są wyświetlane w konsoli.

6 TESTY APLIKACJI

6.1 TESTY JEDNOSTKOWE FRONTENDU

Testy jednostkowe dla frontendu nie są aż tak istotne jak testy manualne, czy funkcjonalne, jednak zawsze warto uwzględnić je w projekcie. Dla każdego frameworku napisano testy sprawdzające między innymi to, czy serwisy i store zwracają odpowiednie dane. Do napisania testów użyta została biblioteka *Jest*, która jest głównie używana do testowania aplikacji napisanych w języku Javascript oraz Typescript. Pozwala ona na mockowanie, asercje oraz uruchamianie testów równolegle. Przykład pliku z testami aplikacji w Angularze przedstawiono poniżej (Listing 6.1). Plik ten zawiera zbiór testów oraz metody *beforeEach* wykonujące się przed każdym z nich, Pierwsza metoda *beforeEach* konfiguruje moduł testowy. Wykonuje się asynchronicznie i tworzy moduł z użyciem *TestBed*. *TestBed* tworzy instancje modułu testowego z odpowiednimi deklaracjami komponentów, importami, oraz dostawcami. Drugi *beforeEach* wykonuje się synchronicznie. Tworzy on instancje testowanego komponentu (w tym przypadku *AwardedOfferGridComponent*) oraz ustawia początkowy stan testu. Pierwszy test sprawdza, czy komponent został utworzony, drugi, czy *GetAwardedOffers* jest wywoływane po załadowaniu komponentu. Ostatni test sprawdza, czy zwrócone przez serwis dane są poprawnie zasubskrybowane w komponencie.

```
describe('AwardedOfferGridComponent', () => {
  let component: AwardedOfferGridComponent;
  let fixture: ComponentFixture<AwardedOfferGridComponent>;
  let store: Store;
  const mockOffers: OfferCardComponentModel[] = [
    { //mockowane oferty }]

beforeEach(async () => {
       await TestBed.configureTestingModule({
       declarations: [AwardedOfferGridComponent, AwardedOfferComponent],
       imports: [
       NgxsModule.forRoot([]),
       NoopAnimationsModule
       ],
       providers: [
       {
       provide: Store,
       useValue: {
       dispatch: jasmine.createSpy('dispatch'),
       select: () => of(mockOffers)
       }|]
       }).compileComponents();
       });
beforeEach(() => {
    fixture = TestBed.createComponent(AwardedOfferGridComponent);
    component = fixture.componentInstance;
    store = TestBed.inject(Store);
    fixture.detectChanges();
  });
```

Listing 6.1: Deklaracje komponentu testowego AwardedOfferGridComponent – Angular

```
  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should dispatch GetAwardedOffers action on init', () => {
    expect(store.dispatch).toHaveBeenCalledWith(new GetAwardedOffers());
  });

  it('should select offers from the store', () => {
    component.offers$.subscribe(offers => {
      expect(offers).toEqual(mockOffers);
    });
  });
```

Listing 6.2: Testy komponentu AwardedOfferGrid – Angular

Testy jednostkowe Reacta dla komponentu *AwardedOfferGrid* wymagają stworzenia *mock-store'a* za pomocą biblioteki *redux-mock-store*. Jest on tworzony w metodzie *beforeEach* za pomocą funkcji *mockStore*. Sztuczny *store* jako jeden z swoich parametrów przyjmuje zamockowane ofery. Następnie w tej metodzie przypisywany jest typ akcji zwracany przez zamockowaną akcję *getAwardedOffers*. Metoda *beforeEach* i testy przedstawione są na listingu 6.3.

```
beforeEach(() => {
    store = mockStore({
      offerCard: {
        offerCardsComponent: mockOffers,
        redisNotResponding: false,
        offer: null,
        offerImages: []
       }
    });

    (getAwardedOffers as jest.Mock).mockReturnValue({ type:
'GET_AWARDED_OFFERS' });
  });

  it('should create', () => {
    const { container } = render(
      <Provider store={store}>
        <AwardedOfferGridComponent />
      </Provider>
    );
    expect(container).toBeTruthy();
  });
  it('should dispatch getAwardedOffers action on init', () => {
    render(
      <Provider store={store}>
        <AwardedOfferGridComponent />
      </Provider>
    );
    expect(store.getActions()).toContainEqual({ type: 'GET_AWARDED_OFFERS'
});
  });
```

Listing 6.3: Fragment pliku testów jednostkowych komponentu AwardedOfferGrid – React

Testy jednostkowe ostatniego frameworka wymagają zdefiniowana akcesoriów, akcji oraz store'a w metodzie *beforeEach*. Akcesor *awardedOffers* zwraca zamockowane dane zwróconych ofert, a akcje natomiast zdefiniowane są metodą *jest.fn()*. Fragment pliku testującego aplikację Vue przedstawiony jest na listingu 6.4.

```
beforeEach(() => {
    getters = {
      awardedOffers: () => [
        {
            offerId: 1,
            offerMainImage: {
                imageBytes: '',
                isMainImage: true
            },
            offerTitle: 'offer1',
            yearOfProduction: '2000',
            mileage: '1000',
            mileageUnit: "km",
            fuelType: 'benzyna',
            engineCapacity: '2000',
            offerPrice: "100000",
            offerCurrency: "pln"
        },
        {
            offerId: 2,
            offerMainImage: {
                imageBytes: '',
                isMainImage: true
                                        },
            offerTitle: 'offer2',
            yearOfProduction: '2010',
            mileage: '100000',
            mileageUnit: "km",
            fuelType: 'benzyna',
            engineCapacity: '2001',
            offerPrice: "300000",
            offerCurrency: "pln"
        },
      ],
    };

    actions = {
      getAwardedOffers: jest.fn(),
    };

    store = new Vuex.Store({
      getters,
      actions,
    });
  });
```

Listing 6.4: Implementacja metody beforeEach AwardedOfferGrid – Vue

```
 it('should create', () => {
      const wrapper = shallowMount(AwardedOfferGrid, { store, localVue,
router });
      expect(wrapper.exists()).toBeTruthy();
  });

  it('should dispatch getAwardedOffers action on init', () => {
    shallowMount(AwardedOfferGrid, { store, localVue, router });
    expect(actions.getAwardedOffers).toHaveBeenCalled();
  });

  it('should select awardedOffers from the store', () => {
    const wrapper = shallowMount(AwardedOfferGrid, { store, localVue,
router });
    const offers = wrapper.findAllComponents(AwardedOfferGrid);
    expect(offers.length).toBe(2);
    expect(offers.at(0).props().offer).toEqual( {
        // zawartość pierwszej oferty
    });
    expect(offers.at(1).props().offer).toEqual({
       // zawartość drugiej oferty
    });
  });
```

Listing 6.5: Testy komponentu AwardedOfferGrid - Vue

6.2 TESTY JEDNOSTKOWE BACKENDU

Testy jednostkowe są istotnym elementem zapewnienia jakości kodu. Weryfikują one działanie pojedynczych jednostek kodu, takich jak funkcje, czy klasy, w celu potwierdzenia, że działają one zgodnie z oczekiwaniami. Do napisania testów jednostkowych w C# wykorzystano bibliotekę *Moq* oraz *Microsoft.VisualStudio.TestTools.UnitTesting*. *Moq* pozwala na tworzenie obiektów zastępujących te prawdziwe (mocków). Umożliwia to symulowanie zachowań Poniżej przedstawiony został przykład jednego z testów jednostkowych aplikacji serwerowej, sprawdzającego, czy metoda *GetAwardedOffers* klasy *OfferMediator* zwraca i mapuje dane w odpowiedni sposób (Listing 6.6). Na początek mockowane są wszystkie zmienne potrzebne do przekazania do konstruktora klasy *OfferMediator*. Obiekt ten tworzony jest w metodzie *SetUp*, która jest uruchamiana przed każdym testem jednostkowym. Test jednostkowy znajduje się poniżej z adnotacją *TestMethod*. Pierwszy krok testu (*Arrange*) wymaga utworzenia danych, które będą zwracane przez funkcje mockowanych obiektów (metody *Setup* i *ReturnsAsync*). W drugim kroku testu wywoływana jest testowana metoda, w tym przypadku - *_offerMediator.GetAwardedOffers()*. Trzecia sekcja to asercje sprawdzające poprawność działania testu. Pierwsza asercja sprawdza czy zwrócony obiekt nie jest pusty, druga - czy zwrócona jest taka sama ilość wierszy oraz trzecia, czy sprawdzana metoda wywołuje wymaganą liczbę razy zamockowane funkcje.

```
namespace oto_auto_c_sharp_server.Tests.Logic.Offers {
    [TestClass]
    public class OfferMediatorTest {
        //Deklaracja mock'ów

    [TestInitialize]
     public void SetUp()
     {
         //inicjalizacja mock'ów
        _offerMediator = new OfferMediator(
        // przekazanie mock'ów do obiektu
      );
     }
        [TestMethod]
        public async Task
GetAwardedOffers_ShouldReturnListOfOfferCardComponentModel()
        {
            // Arrange
            var awardedOffers = new List<Offer>
            {
                // deklaracja dwóch przykładowych ofert
            };
            offerRepository
                .Setup(repo => repo.GetAwardedOffers())
                .ReturnsAsync(awardedOffers);
            foreach (var offer in awardedOffers)
            {
                var offerCardComponentModel = new OfferCardComponentModel
                {
                   Tworzenie testowego obiektu
                };
                mapper
                    .Setup(m => m.Map<OfferCardComponentModel>(offer))
                    .Returns(offerCardComponentModel);
            }
            // Act
            var result = await _offerMediator.GetAwardedOffers();
            // Assert
            Assert.IsNotNull(result);
            Assert.AreEqual(awardedOffers.Count, result.Count);
            offerRepository.Verify(repo => repo.GetAwardedOffers(),
            Times.Once);
        }}}
```

Listing 6.6: Przykład testu jednostkowego backendowego - GetAwardedOffers

6.3 TESTY FUNKCJONALNE

Testy funkcjonalne odgrywają kluczową rolę w procesie weryfikacji aplikacji. Walidują jej poprawne działanie, zapewniając, że wszystkie funkcje działają zgodnie z wymaganiami. Testy przeprowadzone zostaną manualnie tylko dla *Angulara*, pomimo zaimplementowania ich także w frameworkach *React* i *Vue*. Jest to spowodowane faktem, że aplikacje mają identyczną logikę biznesową i interfejs użytkownika. Różnice są tylko w zastosowanych technologiach i nie byłyby widoczne na ekranach. Wyniki testów *Angulara* mogą być uznane za reprezentatywne dla pozostałych frameworków.

6.3.1 FUNKCJONALNOŚĆ PRZEGLĄDANIA OFERT SPRZEDAŻY POJAZDÓW

Pierwszym wymaganiem funkcjonalnym serwisu sprzedaży pojazdów jest możliwość przeglądania owych pojazdów. Aplikacja udostępnia tą funkcjonalność na ekranie głównym (Rysunek 3.1**Błąd! Nie można odnaleźć źródła odwołania.**), gdzie można przeglądać wyróżnione oferty oraz na ekranie przeglądarki ofert (Rysunek 3.3).

6.3.2 FUNKCJONALNOŚĆ PRZEGLĄDANIA SZCZEGÓŁÓW OFERTY

Aplikacja powinna udostępniać możliwość wyświetlenia szczegółów oferty (Rysunek 3.2). Szczegóły uwzględniają większą ilość zdjęć, wyposażenie pojazdu, opis, czy też większą ilość informacji na temat samego pojazdu. Te dane wyświetlone są na ekranie szczegółów oferty, gdzie na głównym planie jest dobrze widoczna galeria zdjęć, tytuł, szczegóły i cena ogłoszenia.

6.3.3 FUNKCJONALNOŚĆ TWORZENIA OFERTY

Serwis ogłoszeń musi udostępniać opcję tworzenia tych ogłoszeń. Jest to możliwe na ekranie tworzenia aplikacji przedstawionym w rozdziale 3.4. Na tym ekranie znajdują się sekcje „Dane pojazdu", „Informacje podstawowe", „Dane techniczne", „Zdjęcia", „Opis pojazdu", „Dane sprzedającego" czy „Cena". Wypełnienie tych sekcji pozwoli na dodanie nowego ogłoszenia.

6.3.4 FUNKCJONALNOŚĆ PRZEDSTAWIAJĄCA WYRÓŻNIONE OFERTY

Funkcjonalność przedstawiania wyróżnionych ofert ma pozwolić na zwiększenie prawdopodobieństwa wybrania i zakupu pojazdu użytkownika, który dodatkowo wypromował swoją ofertę. Wyróżnione oferty są wyświetlane na ekranie głównym pod formularzem do wyszukiwania ofert (Rysunek 3.1).

6.3.5 FUNKCJONALNOŚĆ ZAŁĄCZANIA ZDJĘĆ OFERTY

Tworzenie oferty wymaga przesłania zdjęć pojazdu. Na ekranie tworzenia oferty istnieje sekcja „Zdjęcia" (Rysunek 3.5) umożliwiająca załączenie zdjęć poprzez przeciągnięcie i upuszczenie pliku z odpowiednim rozszerzeniem.

6.3.6 FUNKCJONALNOŚĆ UDOSTĘPNIAJĄCA FORMULARZ DO WYSZUKIWANIA OFERT

Na ekranie głównym znajduje się formularz (Rysunek 3.1), którego uzupełnienie przeniesie użytkownika na stronę przeglądarki ofert (Rysunek 3.3). Formularz posiada pola filtrujące oferty po: typie nadwozia, marce i modelu pojazdu, generacji, cenie, roku produkcji, rodzaju paliwa i przebiegu.

7 ANALIZA PORÓWNAWCZA FRAMEWORKÓW FRONTENDOWYCH

Analiza porównawcza frameworków do tworzenia stron internetowych ma duże znaczenie w wyborze odpowiedniego narzędzia. Taka analiza jest zawsze obiektywna i pozwala poznać mocne i słabe strony poszczególnych technologii. Często skutkuje to bardziej zrównoważonym i efektywnym procesem rozwoju oprogramowania. [20]

7.1 WYDAJNOŚĆ

Aplikacje przetestowane zostały przez platformę *webpagetest.org*. W tym celu wymagane było *hostowanie* aplikacji wraz z serwerami i bazami danych w chmurze. *Hosting* odbył się na platformie *DigitalOcean* umożliwiającej tworzenie maszyn wirtualnych nazwanych „kropelkami" (ang. „*droplets*"). Maszyna wirtualna wykorzystana w trakcie testów utworzona została w Frankfurcie. Jej obraz to *Ubuntu* w wersji 24.04, a podzespoły to dwa procesory AMD, 4GB pamięci RAM oraz 80GB przestrzeni dyskowej na dysku NVMe SSD. Aplikacja została uruchomiona przy użyciu narzędzia *Docker Compose*.

7.1.1 WYDAJNOŚĆ APLIKACJI OTOAUTO

Wydajność aplikacji została przetestowana tylko na stronie głównej aplikacji. Uwzględnione zostało dziewięć parametrów, które zostaną omówione w dalszych podrozdziałach.

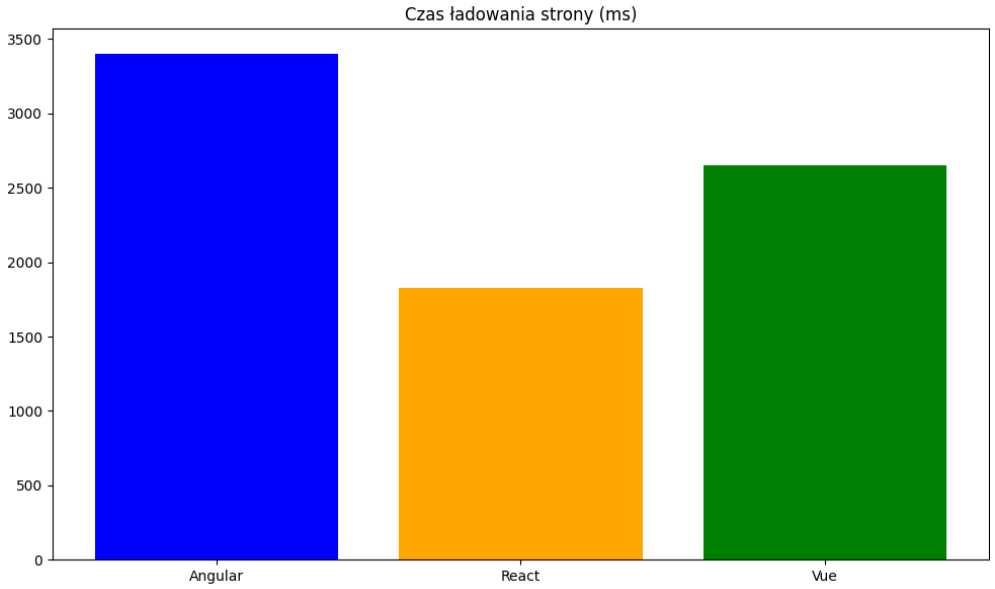
Tabela 7.1: Wynik testu strony głównej aplikacji otoauto

PARAMETR [MS]	*ANGULAR*	*REACT*	*VUE*
Czas ładowania strony	3399	1829	2653
Czas renderowania pierwszego fragmentu treści	2345,9	1837	1504,5
Czas załadowania pierwszej widocznej części strony	2847	2460	2081
Czas parsowania dokumentu przez przeglądarkę	2291	1730	1451
Czas pełnego załadowania strony i jej zasobów	3349	1730	2615
Całkowity czas załadowania strony	3399	3017	2658
Czas renderowania pierwszego fragmentu DOM	2395	1936	1542
Czas pojawienia się największego elementu treści	3552	3034	2842
Miara stabilności wizualnej	0,175171	0,000659	0,004091

7.1.1.1 Czas ładowania strony

Ten parametr oznacza czas, jaki strona potrzebuje, by w pełni załadować stronę od rozpoczęcia ładowania. Jest to istotny aspekt wydajności, ponieważ długi czas ładowania

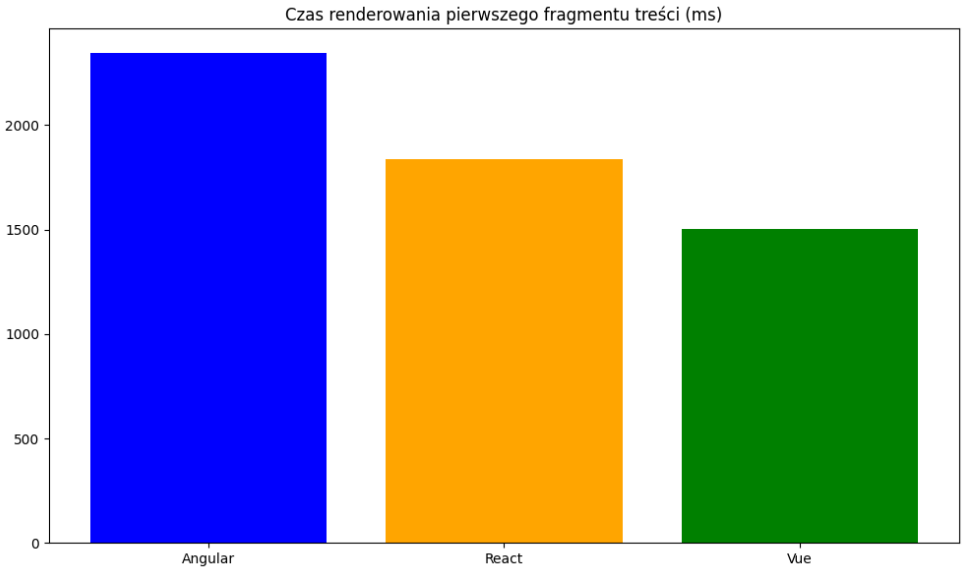
może zniechęcić użytkowników z korzystania z aplikacji. W przypadku omawianej aplikacji najkrótszy czas ładowania ma *React*, a najdłuższy *Angular*.



Rysunek 7.1: Wyniki czasu ładowania strony otoauto

7.1.1.2 Czas renderowania pierwszego fragmentu treści

Parametr ten przedstawia czas, w którym przeglądarka renderuje pierwszy fragment treści strony. Innymi słowy jest to czas, po jakim użytkownik po raz pierwszy widzi coś na stronie. Z wszystkich porównywanych narzędzi *Vue* osiągnął najkrótszy czas, a *Angular* najdłuższy.

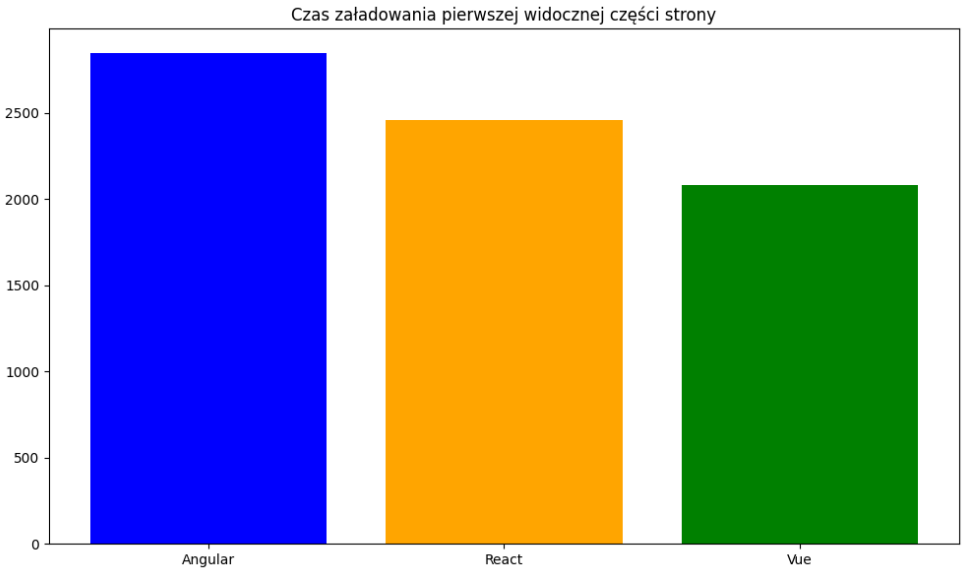


Rysunek 7.2: Wynik czasu renderowania pierszego fragmentu treści strony otoauto

7.1.1.3 Czas załadowania pierwszej widocznej części strony

Ten parametr przedstawia czas, który jest potrzebny przeglądarce do wyświetlenia zawartości strony. Mały czas tego parametru jest istotny, by użytkownik mógł jak

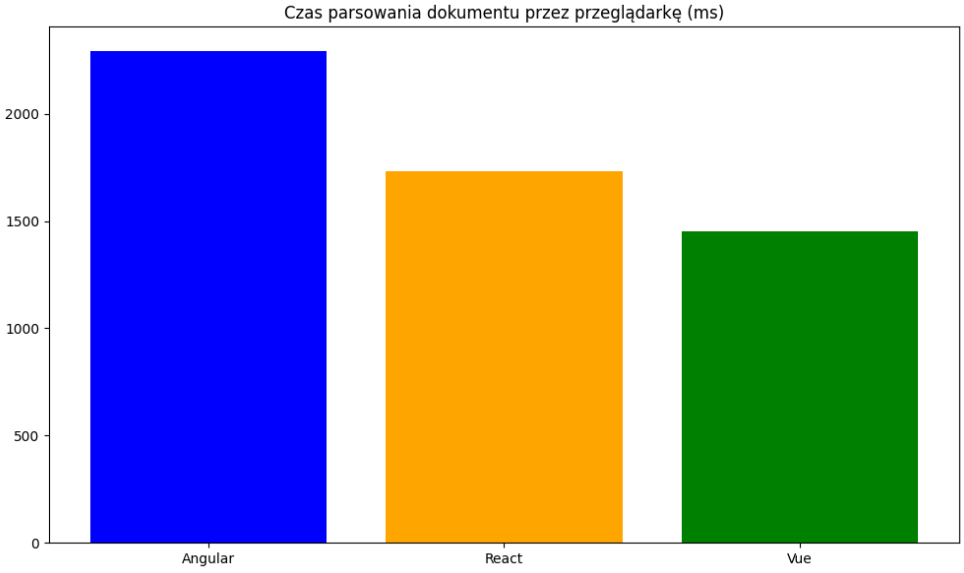
najszybciej korzystać z tego co znajduje się na stronie. Ponownie *Vue* ma najlepszy czas, a *Angular* najdłuższy.



Rysunek 7.3: Wynik czasu załadowania pierwszej widocznej części strony otoauto

7.1.1.4 Czas parsowania dokumentu przez przeglądarkę

Parametr oznaczający czas potrzebny do załadowania wszystkich zasobów, np. skryptów. Szybkie parsowanie pozwala przeglądarce na szybsze wykonywanie skryptów, a co za tym idzie, poprawienie interaktywności strony. Czas parsowania jest najkrótszy dla *Vue*, a najdłuższy dla *Angulara*.

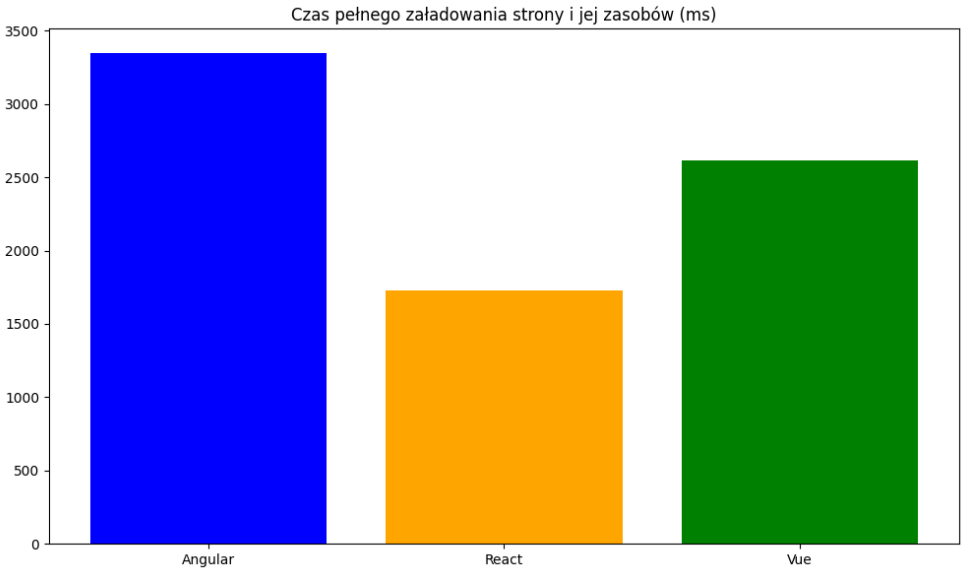


Rysunek 7.4: Wynik czasu parsowania dokumentu przez przeglądarkę – otoauto

7.1.1.5 Czas pełnego załadowania strony i jej zasobów

To parametr wskazujący na czas, po którym strona została załadowana wraz z obrazami, stylami lub skryptami. Szybkie załadowanie całości strony jest kluczowe dla użytkownika, ponieważ pozwala ono na dostęp do wszystkich funkcji strony. Dla omawianej

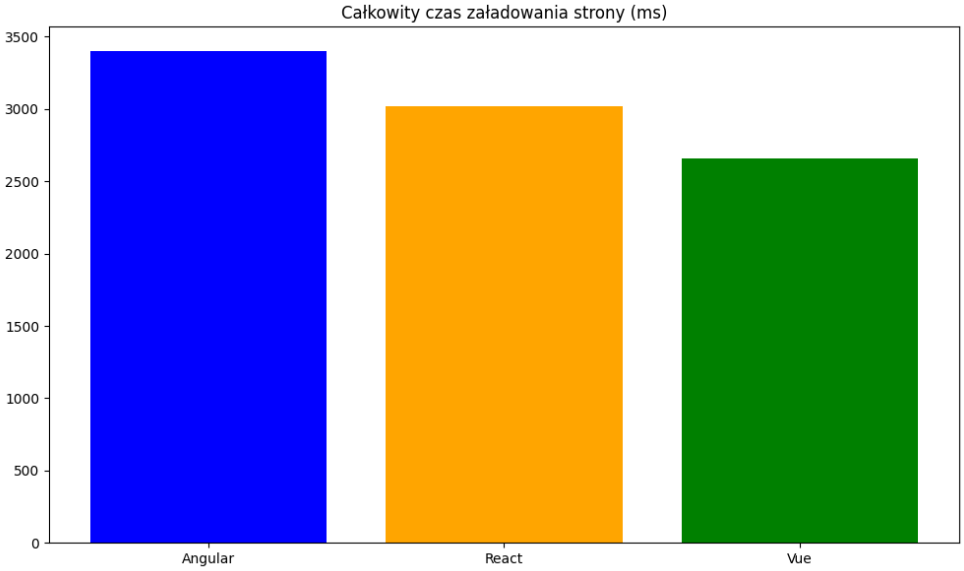
aplikacji całą zawartość najszybciej ładuje *React*, a najwolniej *Angular*, którego czas jest prawie dwukrotnie większy od czasu *Reacta*.



Rysunek 7.5: Wynik czasu pełnego załadowana strony oraz jej zasobów – otoauto

7.1.1.6 Całkowity czas załadowania strony

Jest to parametr, który oprócz czasu pełnego załadowania strony i jej zasobów, bierze też pod uwagę czas do momentu załadowania zasobów pobieranych asynchronicznie. Metryka ta pomaga ocenić obciążenie strony. Badanie czasu wykazało, że *Vue* poradził sobie z załadowaniem strony najlepiej.

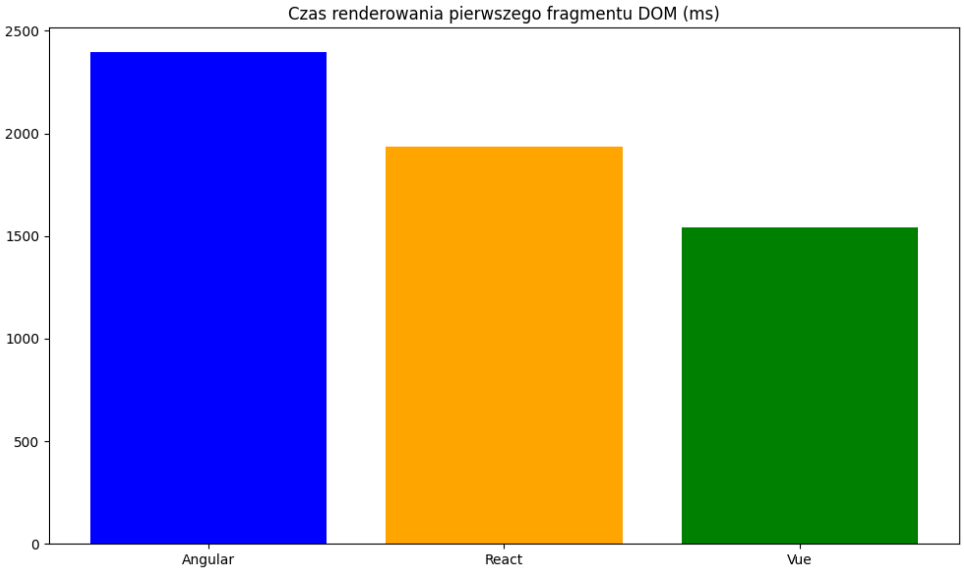


Rysunek 7.6: Wynik całkowitego czasu załadowania strony - otoauto

7.1.1.7 Czas renderowania pierwszego fragmentu DOM

Parametr ten wskazuje czas, w którym przeglądarka renderuje pierwszy fragment treści DOM (tekst, obraz lub grafika SVG). Wpływa on na postrzeganie szybkości ładowania przez użytkownika i jeśli jest on niski to może znacząco poprawić pierwsze wrażenie

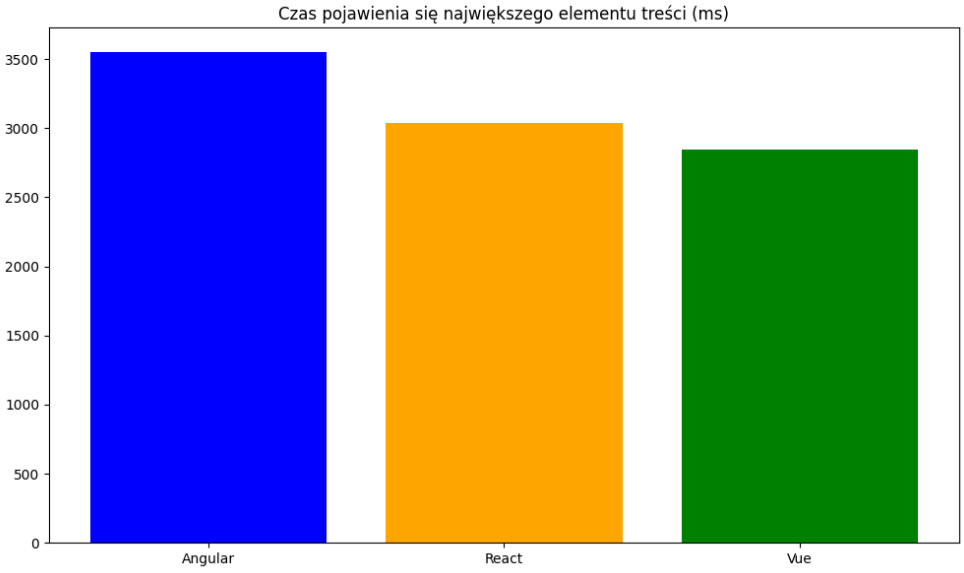
z korzystania z strony. Z załadowaniem pierwszego fragmentu DOM najszybciej poradził sobie *Vue*.



Rysunek 7.7: Wynik czasu renderowania pierwszego fragmentu DOM - otoauto

7.1.1.8 Czas pojawienia się największego elementu treści

Jest to parametr określający czas, w którym największy element na stronie staje się widoczny. Często największy element jest najważniejszy. Z tego powodu jego szybkie załadowanie może mieć kluczową rolę w kontekście wpływu na wrażenia użytkownika. Najlepiej w tym aspekcie poradził sobie *Vue*, a najgorzej ponownie *Angular*.

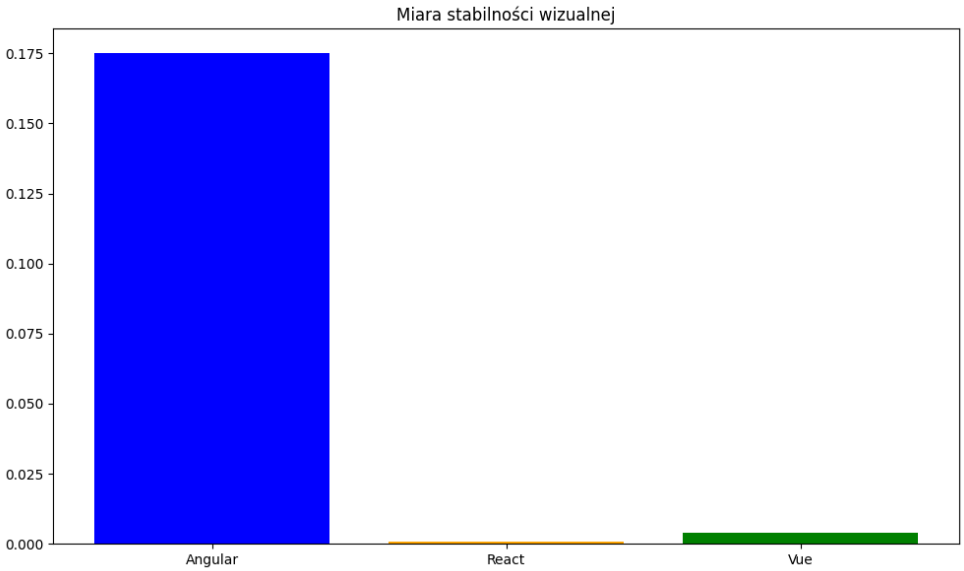


Rysunek 7.8: Wynik czasu pojawienia się największego elementu treści – otoauto

7.1.1.9 Miara stabilności wizualnej

Parametr ocenia niespodziewane przesunięcia w komponentach/układzie na stronie. Niższa wartość tej miary wskazuje na większą stabilność. Niespodziewane zmiany w widoku strony mogą być frustrujące dla użytkownika, dlatego ważne jest, by je

minimalizować. Wynik tego testu wskazuje na dużą wartość tego parametru dla frameworku *Angular*, jednak należy mieć na uwadze, że może być to spowodowane różnicami implementacyjnymi i nie mieć bezpośredniego przełożenia na to, które z narzędzi jest lepsze w tym aspekcie.



Rysunek 7.9: Wynik miary stabilności wizualnej - otoauto

7.1.2 TEST PUSTEJ APLIKACJI

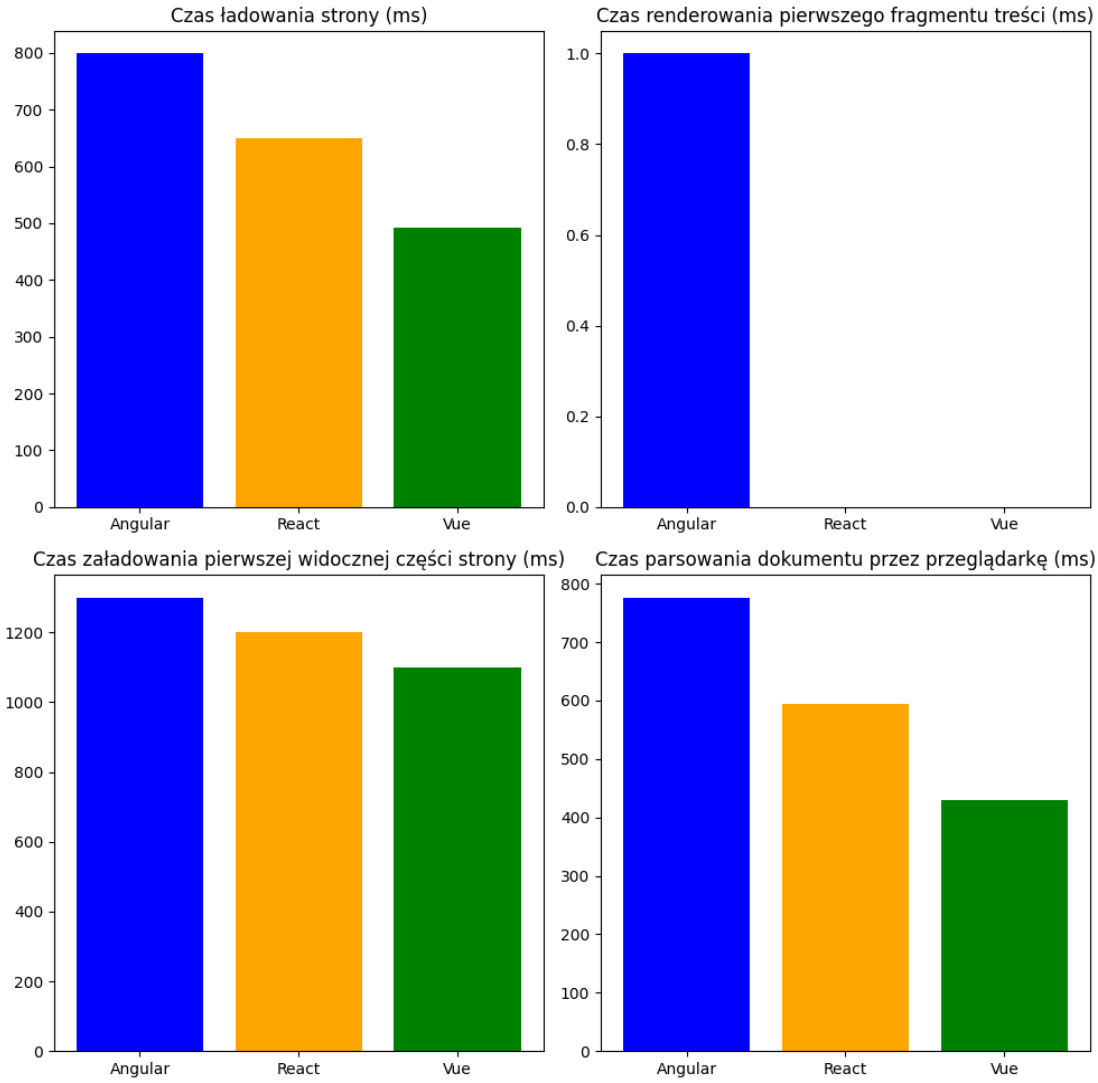
Sprawdzenie wydajności pustej aplikacji ma na celu sprawdzenie podstawowej wydajności danego frameworka. Narzędzia te mają różne architektury, a to może mieć znaczenie dla czasu ładowania i wydajności. Jest to również idealny sposób na sprawdzenie jak szybko działa każdy z frameworków, ponieważ nie występują żadne, nawet najmniejsze, różnice implementacyjne, co skutkuje minimalizacją opóźnień.

Tabela 7.2: Wynik testu wydajności pustej strony

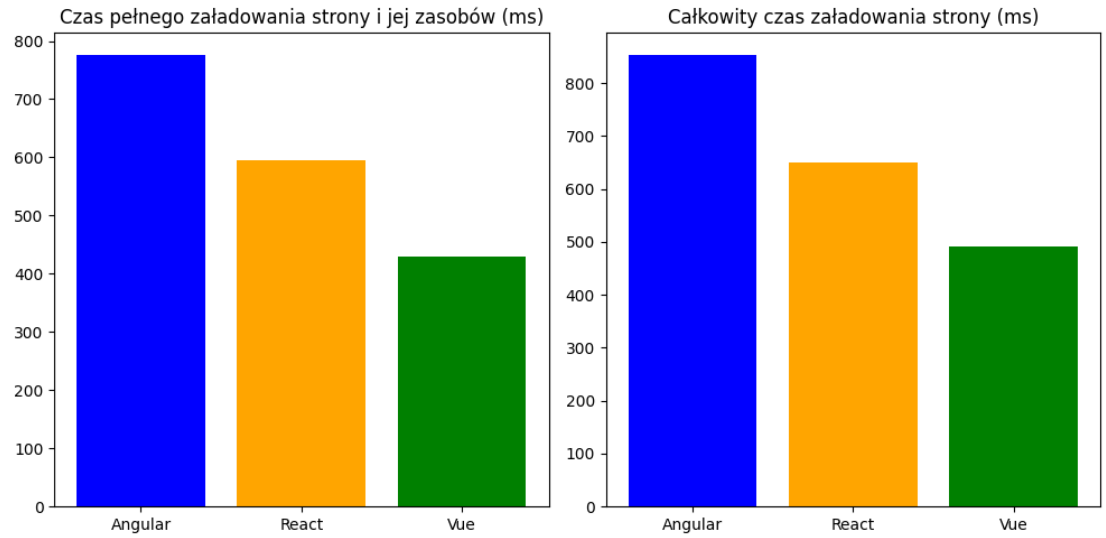
PARAMETR [MS]	*ANGULAR*	*REACT*	*VUE*
Czas ładowania strony	799	650	492
Czas renderowania pierwszego fragmentu treści	1	0	0
Czas załadowania pierwszej widocznej części strony	1299	1199	1099
Czas parsowania dokumentu przez przeglądarkę	776	595	430
Czas pełnego załadowania strony i jej zasobów	776	595	430
Całkowity czas załadowania strony	3399	3017	2658

PARAMETR [MS]	*ANGULAR*	*REACT*	*VUE*
Czas renderowania pierwszego fragmentu DOM	853	650	492

Dla pustej strony *Vue* wydajnością przewyższa *Angulara* i *Reacta*. Ma najkrótsze czasy ładowania strony oraz renderowania treści, które się na niej znajdują. *Angular* w tym przypadku znów poradził sobie najgorzej. *Vue* według przedstawionych wyników dla całkowitego czasu załadowania strony był około 22% szybszy od *Angulara* oraz ok. 12% szybszy od *Reacta*. Najciekawszym przypadkiem jest „*Czas renderowania pierwszego fragmentu treści*", gdzie *React* i *Vue* wykazały czas 0ms, a *Angular* 1ms. Jest to prawdopodobnie spowodowane tym, że ten ostatni jest pełnym frameworkiem, który musi załadować podstawowe moduły, które mogą wprowadzić to minimalne opóźnienie.



Rysunek 7.10: Wyniki testów wydajności pustej aplikacji cz.1



Rysunek 7.11: Wyniki testów wydajności pustej aplikacji cz.2

7.1.3 TEST WYPEŁNIONEJ APLIKACJI

Testy wydajności strony wypełnionej dużą ilością treści (w tym przypadku 100 000 linijek tekstu i pól tekstowych) pozwalają sprawdzić jak dobrze dany framework radzi sobie z renderowaniem i jak wpływa to na doświadczenie z korzystania strony przez użytkownika. Czas renderowania ma duże znaczenie dla stron internetowych, a ten test może być wyznacznikiem tego jak narzędzia będą zachowywać się w przypadku wyświetlania na przykład dużej liczby tabel lub dużego formularza.

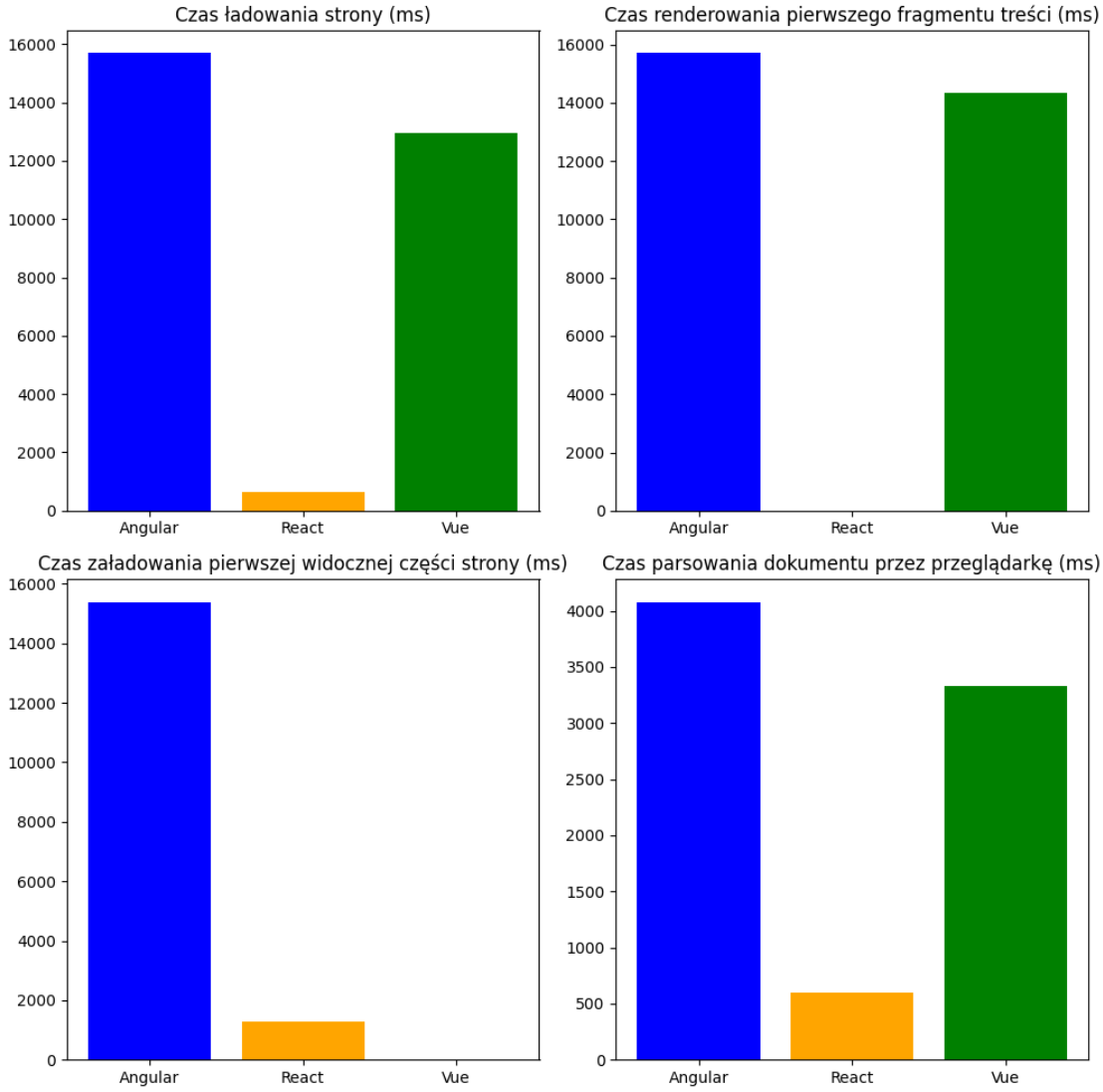
Tabela 7.3: Wynik testu wypełnionej aplikacji

PARAMETR [MS]	*ANGULAR*	*REACT*	*VUE*
Czas ładowania strony	15691	647	12941
Czas renderowania pierwszego fragmentu treści	15705	0	14342
Czas załadowania pierwszej widocznej części strony	15381	1299	0
Czas parsowania dokumentu przez przeglądarkę	4076	601	3330
Czas pełnego załadowania strony i jej zasobów	15599	601	12884
Całkowity czas załadowania strony	19024	647	20868
Czas renderowania pierwszego fragmentu DOM	15797	53561	14399
Czas pokazania głównej treści strony	15797	-	14399

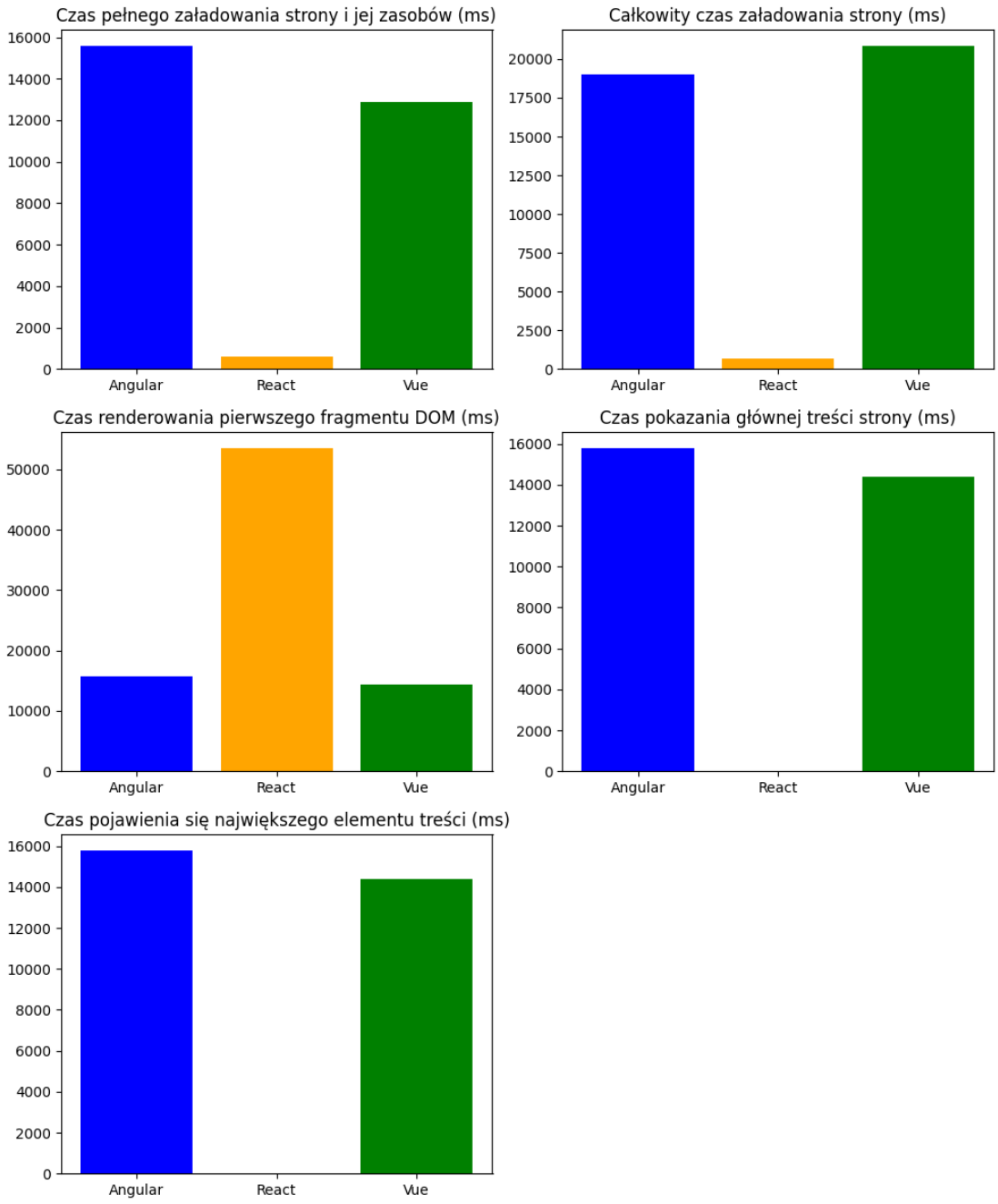
PARAMETR [MS]	*ANGULAR*	*REACT*	*VUE*
Czas pojawienia się największego elementu treści	15797	-	14399

Niestety testy wydajności *Reacta* wyświetlającego sto tysięcy wierszy nie powiodły się, prawdopodobnie ze względu na zbyt długi czas oczekiwania na załadowanie stron, co można stwierdzić parametrem „*Czas renderowania pierwszego fragmentu DOM*”. Z tego powodu realnie wydajność porównać można tylko dla frameworku *Angular* i *Vue*. Wydajność *Reacta* w przypadku tego testu można uznać za niewystarczającą.

Testy wydajności dla strony wypełnionej wierszami z tekstem i polem tekstowym wykazały, że zarówno *Angular* jak i *Vue* mają bardzo długi czas ładowania i renderowania, co wpływa negatywnie na doświadczenie użytkownika. W kontekście dużych zestawów danych, należy zastosować techniki optymalizujące w celu poprawienia wydajności, płynności i działania aplikacji.



Rysunek 7.12: Wyniki testów pełnej aplikacji cz.1



Rysunek 7.13: Wyniki testów wypełnionej aplikacji cz.2

7.2 ROZMIAR PACZEK PRODUKCYJNYCH

Ważnym aspektem analizy wydajności aplikacji webowych jest porównanie rozmiaru paczek produkcyjnych. Rozmiar ma wpływ na czas ładowania strony, ponieważ im jest mniejszy, tym przeglądarka musi pobrać mniej danych. Powoduje to poprawę doświadczenia użytkownika. Dzięki mniejszemu rozmiarowi, wydajność będzie wystarczająca na starszych urządzeniach lub urządzeniach mobilnych, które mają mniejszą moc obliczeniową oraz mniej zasobów. Mniejsze paczki będą się szybciej ładować, a także zużywać mniej pamięci.

Paczki produkcyjne dla każdego z frameworków zostały wygenerowane komendą

npm run build –prod

Wynikiem tej komendy jest folder wyjściowy *dist*, zawierający zoptymalizowane pliki *Javascript*, *HTML* i *CSS*. Jego zawartość różni się nieznacznie pomiędzy frameworkami, jednak każdy z nich zawiera wszystkie niezbędne elementy do poprawnego działania aplikacji.

Tabela 7.4: Wynik pomiaru rozmiaru paczek produkcyjnych

ROZMIAR PACZEK PRODUKCYJNYCH [B]	*ANGULAR*	*REACT*	*VUE*
Pusta aplikacja	254 189	144 528	54 236
Aplikacja otoauto	2 854 147	1 615 460	1 414 663
Aplikacja wypełniona wierszami	266 652	145 053	56 501

Vue w każdym przypadku generuje najmniejsze paczki produkcyjne. Jest najlepiej zoptymalizowany. *React* oferuje stosunkowo małe paczki, co może zapewniać lepszą wydajność. *Angular*, który ma najwięcej wbudowanych funkcji, ma również największe paczki, co będzie wpływać na czas ładowania i wydajność aplikacji.

7.3 STRUKTURA APLIKACJI

Sonarqube to jedno z najpopularniejszych narzędzi o udostępnionym kodzie, analizujące zgodność kodu z zbiorem ustalonych zasad. Jeśli kod łamie zasadę, to jako część długu technicznego zostaje dodany czas potrzebny na poprawę tego kodu. Dodatkowo *Sonarqube* podczas testów mierzy kilka metryk, takich jak liczba linii kodu i jego złożoność. [21] Narzędzie do analizy zostało uruchomione w kontenerze *Docker*owym komendą:

*Docker run -d --name sonarqube *

*-e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true *

-p 9000:9000 sonarqube:latest

Do wykonania testów wymagane jest dodatkowo narzędzie *SonarScanner*, które do każdego projektu *otoauto* zostało dodane komendą:

npm install sonar-scanner --save-dev

Dodatkowo należy skonfigurować skanowany projekt plikiem *sonar-project.properties*, w którym zdefiniowane zostały takie parametry jak nazwa projektu, ścieżka do plików źródłowych, login oraz hasło potrzebne do zalogowania się w *Sonarqube* oraz *url*, pod którym się on znajduje. Po wykonaniu tych kroków wystarczyło wykonać skrypt inicjujący narzędzie *SonnarScanner*.

Wyniki pomiarów zostały przedstawione w tabeli poniżej.

Tabela 7.5: Wynik analizy Sonarqube

PARAMETR	*ANGULAR*	*REACT*	*VUE*
Liczba linii kodu	3393	2422	2306
Liczba funkcji	166	119	94
Liczba klas	69	3	3
Złożoność cykliczna	200	148	133
WMC	2,41	39,67	31,34

Dane zmierzone przez podstawową wersję narzędzia pozwalają na pomiar jednej z metryk *Chidambera* i *Kemererea*. WMC (ang. „*Weighted Methods per Class*") odpowiada liczbie metod w jednej klasie. *React* oraz *Vue* mają znacznie wyższą wartość tego parametru w porównaniu do *Angulara*. Jest to spowodowane faktem, że dla *Angulara* stworzenie komponentu wymaga stworzenia klasy, gdzie dla *Reacta* i *Vue* same funkcje mogą tworzyć komponenty. Wartość WMC dla *Angulara* sugeruje, że komponenty są bardziej modularne oraz podzielone na mniejsze części. Wartości dla pozostałych narzędzi sugerują, że do stworzenia aplikacji nie jest potrzebna duża liczba klas. Złożoność cykliczna odpowiada złożoności logicznej kodu. *Angular* przyjmuje największą wartość tego parametru. Wyniki dla *Reacta* i *Vue* pod tym względem są zbliżone.

7.4 IMPLEMENTACJA DOM

DOM (ang. „*Document Object Model*") dostarcza ustrukturyzowaną reprezentację dokumentów HTML Ta reprezentacja jest osiągnięta poprzez uporządkowanie elementów dokumentu w drzewo DOM. Elementy tego drzewa mogą mieć właściwości i metody, które mogą być użyte do modyfikacji drzewa. Pozwala to na dynamiczny dostęp do DOMu i dokonywania zmian w jego strukturze, treści i stylu. Innym podejściem jest wirtualny DOM, którego implementacja na celu miała poprawę wydajności i szybkości aktualizacji elementów DOM. Z tego rodzaju DOM korzystają frameworki *React* i *Vue*. *Angular* z niego nie korzysta. Głównym powodem jest fakt, że *Angular* jest następcą *AngularJS*, który został wydany w 2010 roku i nie korzystał z wirtualnego DOMu [22].

Artykuł *"Dom Benchmark Comparison of The Front-End Javascript Frameworks React, Angular, Vue, And Svelte."* [22] zawiera wyniki testów wydajności frameworków *Angular*, *React* i *Vue*. Testy obejmowały takie operacje, jak dodawanie elementów do DOM, edytowanie jednego elementu i wielu elementów, usuwaniu jednego i wielu elementów.

React i *Vue* wykazały się lepszą wydajnością dla przypadków manipulacji dużej ilości danych. Jest to szczególnie widoczne dla operacji edytowania 10 000 elementów. *Angular* okazał się zaskakująco szybki dla operacji edytowania i usuwania jednego elementu. Miał jednak problem z wydajnością przy operacjach z większą liczbą elementów (Tabela 7.6).

Tabela 7.6: Wyniki testów wydajności operacji manipulujących DOM

OPERACJA [MS]	*ANGULAR*	*REACT*	*VUE*
Dodawanie 10 000 elementów *div* z tekstem *p* do DOM	52.75	30.96	25.36
Edytowanie jednego elementu *div* z tekstem *p*	6.08	22.23	16.58
Edytowanie 10 000 elementów *div* z tekstem *p*	896.76	17.86	20.64
Usuwanie jednego elementu *div* z tekstem *p*	0.09	16.54	24.51
Usuwanie 10 000 elementów *div* z tekstem *p*	23.83	7.39	33.33

7.5 POPULARNOŚĆ I WSPARCIE SPOŁECZNOŚCI

Artykuł *"Comparison: Angular Vs. React Vs. Vue. Which Framework Is The Best Choice"* [23]. porusza istotne kwestie dotyczące popularności, wsparcia społeczności,

łatwości nauki oraz elastyczności frameworków *Angular, React* i *Vue*. Wyniki badań pokazały, że *React* jest najpopularniejszym frameworkiem według liczby wyszukiwań w Google i liczby wątków na forum StackOverflow. *Angular* zajął drugie, a *Vue* trzecie miejsce. Należy jednak zaznaczyć, że według badań z 2020 roku popularność *Angulara* zaczęła się stabilizować, a pozostałych narzędzi rosnąć. Wsparcie społeczności również jest największe dla *Reacta.* Jest to sugerowane liczbą repozytoriów na GitHubie. Najłatwiejszą krzywą uczenia może pochwalić się *Vue*, który wymaga znajomości tylko *HTML, CSS* i *Javascriptu*. *React* wymaga nauki *JSX* i *hooków*, a *Angular* wymaga znajomości *Typescriptu*. *Angular* jest też najmniej elastyczny i najbardziej restrykcyjny, wymaga określonej architektury kodu. Nie można tego powiedzieć o *React* i *Vue*, gdyż nie mają one narzuconej struktury aplikacji, co pozwala na łatwą integrację z bibliotekami zewnętrznymi.

Tabela 7.7: Porównanie frameworków pod kątem popularności, wsparcia społeczności, krzywej uczenia się i elastyczności

PARAMETR	*ANGULAR*	*REACT*	*VUE*
Popularność	Stagnacja	Wzrost	Wzrost
Wsparcie społeczności	Średnie	Duże	Małe
Krzywa uczenia się	*Typescript*	JSX, hooki	Brak nowych technologii
Elastyczność	Mała	Duża	Duża

7.6 BEZPIECZEŃSTWO

Bezpieczeństwo aplikacji jest kluczowym aspektem i powinno być priorytetem ze względu na rosnącą liczbę ataków cybernetycznych. Każdy z trzech frameworków – *Angular, React* i *Vue* – inaczej podchodzi do kwestii bezpieczeństwa. W dokumentacji frameworku pierwszego z wymienionych narzędzi bezpieczeństwo zostało uwzględnione za pomocą udostępnienia programiście wbudowanych mechanizmów bezpieczeństwa. Jednym z tych mechanizmów jest sanityzacja danych służąca do ochrony aplikacji przed atakami XSS (Cross-Site Scripting). Ataki te polegają na wstrzykiwaniu złośliwego kodu do strony internetowej. Taki kod może służyć na przykład do uzyskania informacji na temat danych logowania użytkownika lub wykonywania akcji udając użytkownika. Sanityzacja jest to inspekcja wartości, które nie są zaufane i zmiana ich w wartości, które są bezpieczne to włożenia do DOMu. Dla przykładu w *Angularze* istnieje właściwość elementu *innerHTML*, która pozwala na przekazanie wartości string, w której może znajdować się kod HTML. Jeśli w takim kodzie znajdzie się niebezpieczna wartość (taka jak tag <*script*>) to *Angular* automatycznie ją rozpoznaje i sanitazuje, usuwając niebezpieczny tag. Ten framework pozwala też na zaufanie wartościom za pomocą wbudowanych funkcji:

- *bypassSecurityTrustHtml,*
- *bypassSecurityTrustScript,*
- *bypassSecurityTrustStyle,*
- *bypassSecurityTrustUrl,*
- *bypassSecurityTrustResourceUrl.*

Wymagane jest użycie poprawnej metody w zależności od kontekstu. [24]

Niestety *React* oraz *Vue* nie mają wbudowanych mechanizmów bezpieczeństwa w takim stopniu jak *Angular*. Są one bardziej elastyczne, lecz co za tym idzie, wymagają od programistów świadomego działania i szanowania zasad bezpieczeństwa. Dokumentacje tych frameworków zawierają sekcje dotyczące zabezpieczeń, lecz po więcej informacji

należy udać się na zewnętrzne fora i przewodniki omawiające dobre praktyki bezpieczeństwa.

W przypadku każdego z frameworków kluczowe jest regularne aktualizowanie frameworków, gdyż zapewnia to bezpieczeństwo i odporność aplikacji na nowe zagrożenia.

7.7 PODSUMOWANIE

Wydajność aplikacji została przetestowana uwzględniając dziewięć parametrów. *React* i *Vue* osiągały najlepsze wyniki, zwłaszcza dla czasu ładowania i renderowania treści. *Angular* w większości testów potrzebował najwięcej czasu do ukończenia operacji. Testy wydajności dla pustej aplikacji pokazały, że *Vue* osiąga znacznie lepsze wyniki od swoich konkurentów. W przypadku testów aplikacji wypełnionej *React* nie był w stanie ukończyć operacji wyświetlania wierszy w czasie przewidzianym na jeden test przez stronę *webpagetest.org*. *Angular* w tych testach osiągał podobne wyniki do *Vue*.

Rozmiar paczek produkcyjnych jest najmniejszy dla *Vue*. Skutkuje to lepszą wydajnością na starszych urządzeniach lub urządzeniach mobilnych mających gorsze podzespoły. *React* generuje małe paczki, natomiast *Angular* ma największe paczki. Przyczyną tego jest fakt, że ten framework ma najwięcej wbudowanych funkcji.

Artykuł *„Evaluating the performance of web rendering technologies based on Javascript: Angular, React, and Vue”* [25] miał na celu między innymi sprawdzenie, który framework działa najlepiej oraz jest najlepiej zoptymalizowany, jeśli chodzi o tworzenie paczek. Wnioskami tego artykułu były:

- *Vue* przewyższa *Reacta* i *Angulara* w czasie manipulacji DOM,
- *React* wyróżnia się doskonałą wydajnością, przewyższając *Vue* o 33%, a *Angulara* o 50%,
- *Angular* ma największy rozmiar paczek w porównaniu do *Vue* i *Reacta*.

Wyniki własnych testów oraz artykułu są różne. W przypadku testów przeprowadzonych w tym rozdziale *Vue* wyróżnia się najlepszą wydajnością, a *Angular* największym rozmiarem paczek. *Angular* średnio wymaga najwięcej czasu na ukończenie operacji, co jest zgodne z obserwacjami badań i artykułu.

Analiza struktury aplikacji za pomocą narzędzia *Sonarqube* wykazała, że wartość WMC jest najniższa dla *Angulara*, co jest spowodowane tym, że jako jedyny z tych frameworków wymaga stworzenia klasy, by utworzyć komponent.

Implementacja wirtualnego DOM w *React* i *Vue* zwiększa wydajność manipulacji dużą ilością danych, zwłaszcza przy edycji wielu elementów w tym samym czasie. *Angular* osiągał najlepsze wyniki dla operacji na pojedynczych elementach.

Największa popularność i wsparcie społeczności według trendów Google i liczby wątków na StackOverflow należy do *Reacta*. *Vue* został wskazany jako najłatwiejszy do nauki, wymagając znajomości najbardziej podstawowych technologii frontendowych.

Angular oferuje najlepszą wbudowaną ochronę aplikacji przed atakami XSS dzięki implementacji sanizacji danych. *React* i *Vue* są bardziej elastyczne, jednak wymagają świadomego podejścia do zabezpieczeń aplikacji.

8 WNIOSKOWANIE ROZMYTE

Wnioskowanie rozmyte to matematyczne narzędzie służące do analizy niedokładnych lub dwuznacznych informacji. Opiera się na idei, że prawda może być wyznaczona jako stopień przynależności do zbioru rozmytego, a nie jako binarną wartość „prawda" lub „fałsz". Wnioskowanie rozmyte może być użyte do sterowania systemami, sztucznej inteligencji i podejmowania decyzji.

Kluczowymi pojęciami logiki rozmytej są:

- zbiory rozmyte – są to zbiory, w których elementy mają stopnie przynależności, a nie są ściśle w zbiorze lub poza nim.
- reguły rozmyte – to reguły definiujące relacje między zmiennymi wejściowymi i wyjściowymi przy użyciu wyrażeń lingwistycznych.

Zaletą logiki rozmytej jest zdolność do modelowania skomplikowanych systemów, radzenia sobie z niepewnością oraz zapewnianie bardziej ludzkiego podejścia do obliczeń.

Zbiory rozmyte są fundamentalnym konceptem pozwalającym na reprezentacje niepewności w danych. Różnią się od klasycznych zbiorów, gdyż umożliwiają elementom przypisanie wartości funkcji przynależności pomiędzy 0 a 1. Zbiory rozmyte są zdefiniowane za pomocą funkcji przynależności, która może przyjąć różne formy, takie jak funkcja trapezowa, trójkątna czy Gaussowska. [26]

Narzędzia do tworzenia aplikacji webowych charakteryzują się wieloma parametrami i trudno jest jednoznacznie je ocenić. Tradycyjne metody oceny mogą nie uwzględniać pewnych istotnych różnic. Wnioskowanie rozmyte pozwoli na ocenę subiektywnych ocen parametrów, takich jak wydajność, wsparcie społeczności czy krzywa uczenia. Logika ta pozwoli na integrację różnych informacji z różnych źródeł danych w jedną ocenę, co zwiększy wiarygodność wyników. Wyrażenia lingwistycznie, definiujące reguły rozmyte, pozwolą na stworzenie modelu, który w połączeniu z intuicyjną aplikacją, umożliwi wybór odpowiedniego frameworku dla użytkownika.

8.1 ZMIENNE LINGWISTYCZNE

Do zbudowania modelu potrzebne są zmienne lingwistyczne, które zostaną użyte do zdefiniowania reguł. Będą się one odnosić do aspektów porównanych w rozdziale 7. Te zmienne to:

- wydajność – przyjmująca wartości:
  - „niska" – (0, 0, 20, 40),
  - „średnia" – (20, 40, 60, 70),
  - „wysoka" – (60, 70, 100, 100),
- wydajność wyświetlania dużej ilości treści – przyjmująca wartości:
  - „niska" – (0, 0, 20, 40),
  - „średnia" – (20, 40, 60, 70),
  - „wysoka" – (60, 70, 100, 100),
- rozmiar paczki produkcyjnej – przyjmująca wartości:
  - „mały" – (0, 0, 50, 100) MB,
  - „średni" – (50, 100, 200, 300) MB,
  - „duży" – (200, 300, 1000, 1000) MB,
- wydajność DOM - przyjmująca wartości:
  - „niska" – (0, 0, 20, 40),
  - „średnia" – (20, 40, 60, 70),
  - „wysoka" – (60, 70, 100, 100),
- wydajność DOM wyświetlania dużej ilości treści - przyjmująca wartości:

- „niska” – (0, 0, 20, 40),
- „średnia” – (20, 40, 60, 70),
- „wysoka” – (60, 70, 100, 100),

o krzywa uczenia - przyjmująca wartości:
- „łatwa” – (0, 0, 10, 30),
- „umiarkowana” – (10, 30, 60, 80),
- „trudna” – (60, 80, 100, 100),

o wbudowane bezpieczeństwo - przyjmująca wartość:
- „brak” - (0, 0, 3, 5),
- „wbudowane” - (3, 5, 10, 10),

o popularność – przyjmująca wartości:
- „malejąca” – (0, 0, 20, 40),
- „stagnacja” – (20, 40, 70, 90),
- „rosnąca” – (70, 90, 100, 100),

o elastyczność – przyjmująca wartości:
- „sztywna” – (0, 0, 20, 40),
- „umiarkowana” – (20, 40, 70 90),
- „elastyczna” – (70, 90, 100, 100).

8.2 REGUŁY WNIOSKOWANIA

Model wnioskowania rozmytego do poprawnego działania wymaga zbioru reguł. Umożliwiają one przełożenie zmiennych lingwistycznych i ich wartości na konkretne decyzje. Reguły powinny uwzględniać różnorodne scenariusze i zależności pomiędzy zmiennymi. W modelu zdefiniowane zostaną trzy różne zestawy reguł dla każdego z frameworków. Po podaniu danych, jedna z trzech funkcji wyjścia o największej wartości będzie oznaczać największe dopasowanie do danego frameworka. Efekt działania modelu pozwoli użytkownikowi wybrać najbardziej pasujące mu narzędzie. W poniższych rozdziałach przedstawione zostaną reguły dla każdego z frameworków w postaci IF-THEN.

8.2.1 ANGULAR

1. IF wydajność jest dobra AND wydajność dla dużej ilości danych jest średnia AND rozmiar paczki jest duży THEN wybierz *Angular*.
2. IF wydajność jest średnia AND rozmiar paczki jest średni AND wydajność DOM jest dobra THEN wybierz *Angular*.
3. IF wydajność DOM jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest trudna THEN wybierz *Angular*.
4. IF wydajność jest średnia AND rozmiar paczki jest średni AND bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
5. IF (krzywa uczenia jest umiarkowana OR krzywa uczenia jest trudna) AND popularność jest stagnacja THEN wybierz *Angular*.
6. IF wydajność jest słaba AND elastyczność jest sztywna THEN wybierz *Angular*.
7. IF wydajność DOM jest dobra AND (wydajność DOM wyświetlania dużej ilości treści jest średnia OR wydajność DOM wyświetlania dużej ilości treści jest słaba) AND rozmiar paczki jest duży THEN wybierz *Angular*.
8. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND rozmiar paczki jest mały THEN nie wybieraj *Angulara*.
9. IF popularność jest stagnacja THEN wybierz *Angular*.
10. IF bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
11. IF bezpieczeństwo jest wbudowane AND (krzywa uczenia jest trudna OR krzywa uczenia jest umiarkowana) THEN wybierz *Angular*.

12. IF rozmiar paczki jest duży AND bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
13. IF rozmiar paczki jest mały AND wbudowane bezpieczeństwo jest brak THEN nie wybieraj *Angulara*.
14. IF wydajność jest słaba AND wydajność wyświetlania dużej ilości treści jest średnia AND rozmiar paczki jest średni THEN wybierz *Angular*.
15. IF wydajność jest średnia AND wydajność wyświetlania dużej ilości treści jest średnia AND rozmiar paczki jest duży THEN wybierz *Angular*.
16. IF wydajność jest średnia AND wydajność DOM jest dobra AND bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
17. IF wydajność jest słaba AND wydajność DOM jest dobra AND bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
18. IF wydajność jest średnia AND rozmiar paczki jest średni AND popularność jest stagnacja THEN wybierz *Angular*.
19. IF wydajność jest średnia AND elastyczność jest umiarkowana AND krzywa uczenia jest umiarkowana THEN wybierz *Angular*.
20. IF wydajność jest słaba AND elastyczność jest sztywna AND krzywa uczenia jest trudna THEN wybierz *Angular*.
21. IF wydajność jest średnia AND rozmiar paczki jest duży AND bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
22. IF wydajność jest słaba AND wydajność wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest trudna THEN wybierz *Angular*.
23. IF wydajność jest średnia AND elastyczność jest umiarkowana AND bezpieczeństwo jest wbudowane THEN wybierz *Angular*.
24. IF wydajność jest słaba AND rozmiar paczki jest duży AND krzywa uczenia jest umiarkowana THEN wybierz *Angular*.
25. IF wydajność jest średnia AND wydajność wyświetlania dużej ilości treści jest średnia AND wydajność DOM jest dobra THEN wybierz *Angular*.
26. IF wydajność jest słaba AND wydajność DOM jest dobra AND popularność jest stagnacja THEN wybierz *Angular*.
27. IF wydajność jest średnia AND wydajność wyświetlania dużej ilości treści jest średnia AND elastyczność jest umiarkowana THEN wybierz *Angular*.
28. IF wydajność jest słaba AND rozmiar paczki jest duży AND wydajność DOM jest dobra THEN wybierz *Angular*.
29. IF wydajność jest średnia AND wydajność wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest trudna THEN wybierz *Angular*.
30. IF wydajność jest słaba AND elastyczność jest sztywna AND popularność jest stagnacja THEN wybierz *Angular*.
31. IF wydajność jest średnia AND bezpieczeństwo jest wbudowane AND krzywa uczenia jest umiarkowana THEN wybierz *Angular*.
32. IF wydajność jest słaba AND wydajność wyświetlania dużej ilości treści jest średnia AND elastyczność jest sztywna THEN wybierz *Angular*.
33. IF wydajność jest średnia AND rozmiar paczki jest duży AND elastyczność jest umiarkowana THEN wybierz *Angular*.
34. IF wydajność jest słaba AND wydajność wyświetlania dużej ilości treści jest średnia AND wydajność DOM jest dobra THEN wybierz *Angular*.
35. IF wydajność jest średnia AND bezpieczeństwo jest wbudowane AND elastyczność jest umiarkowana THEN wybierz *Angular*.
36. IF wydajność jest słaba AND popularność jest stagnacja AND elastyczność jest sztywna THEN wybierz *Angular*.

37. IF wydajność jest średnia AND wydajność wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest trudna THEN wybierz *Angular*.

8.2.2 REACT

1. IF wydajność jest słaba AND wydajność wyświetlania dużej ilości treści jest średnia AND rozmiar paczki jest duży THEN nie wybieraj *Reacta*.
2. IF wydajność jest dobra AND rozmiar paczki jest średni AND wydajność DOM jest dobra THEN wybierz *Reacta*.
3. IF wydajność DOM jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest średnia AND (krzywa uczenia jest łatwa OR krzywa uczenia jest umiarkowana) THEN wybierz *Reacta*.
4. IF wydajność jest średnia AND rozmiar paczki jest mały AND wbudowane bezpieczeństwo jest brak THEN wybierz *Reacta*.
5. IF (krzywa uczenia jest umiarkowana OR krzywa uczenia jest łatwa) AND popularność jest rosnąca THEN wybierz *Reacta*.
6. IF wydajność jest słaba AND elastyczność jest sztywna THEN nie wybieraj *Reacta*.
7. IF wydajność DOM jest średnia AND (wydajność DOM wyświetlania dużej ilości treści jest średnia OR wydajność DOM wyświetlania dużej ilości treści jest dobra) AND rozmiar paczki jest mały THEN wybierz *Reacta*.
8. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND rozmiar paczki jest mały THEN wybierz *Reacta*.
9. IF popularność jest stagnacja OR popularność jest malejąca THEN nie wybieraj *Reacta*.
10. IF bezpieczeństwo jest wbudowane THEN nie wybieraj *Reacta*.
11. IF bezpieczeństwo jest wbudowane AND (krzywa uczenia jest łatwa OR krzywa uczenia jest umiarkowana) AND wydajność jest dobra THEN wybierz *Reacta*.
12. IF rozmiar paczki jest duży AND bezpieczeństwo jest wbudowane THEN nie wybieraj *Reacta*.
13. IF rozmiar paczki jest mały AND wbudowane bezpieczeństwo jest brak THEN wybierz *Reacta*.
14. IF wydajność jest dobra AND rozmiar paczki jest średni AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Reacta*.
15. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND krzywa uczenia jest łatwa THEN wybierz *Reacta*.
16. IF wydajność jest dobra AND rozmiar paczki jest mały AND krzywa uczenia jest łatwa THEN wybierz *Reacta*.
17. IF wydajność jest średnia AND rozmiar paczki jest mały AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Reacta*.
18. IF wydajność jest dobra AND elastyczność jest umiarkowana AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Reacta*.
19. IF wydajność DOM wyświetlania dużej ilości treści jest dobra AND krzywa uczenia jest łatwa AND popularność jest rosnąca THEN wybierz *Reacta*.
20. IF wydajność jest dobra AND rozmiar paczki jest średni AND wbudowane bezpieczeństwo jest brak THEN wybierz *Reacta*.
21. IF wydajność jest średnia AND elastyczność jest umiarkowana AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Reacta*.
22. IF wydajność jest dobra AND popularność jest rosnąca AND krzywa uczenia jest umiarkowana THEN wybierz *Reacta*.
23. IF wydajność jest średnia AND rozmiar paczki jest średni AND elastyczność jest umiarkowana THEN wybierz *Reacta*.

24. IF wydajność jest dobra AND rozmiar paczki jest mały AND wydajność DOM wyświetlania dużej ilości treści jest średnia THEN wybierz *Reacta*.
25. IF wydajność jest dobra AND elastyczność jest umiarkowana AND krzywa uczenia jest łatwa THEN wybierz *Reacta*.
26. IF wydajność jest dobra AND popularność jest rosnąca AND elastyczność jest umiarkowana THEN wybierz *Reacta*.
27. IF wydajność jest średnia AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND wbudowane bezpieczeństwo jest brak THEN wybierz *Reacta*.
28. IF wydajność DOM wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest łatwa AND elastyczność jest umiarkowana THEN wybierz *Reacta*.
29. IF wydajność jest dobra AND elastyczność jest umiarkowana AND popularność jest rosnąca THEN wybierz *Reacta*.
30. IF wydajność jest średnia AND rozmiar paczki jest mały AND popularność jest rosnąca THEN wybierz *Reacta*.
31. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND wbudowane bezpieczeństwo jest brak THEN wybierz *Reacta*.
32. IF wydajność jest średnia AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND popularność jest rosnąca THEN wybierz *Reacta*.
33. IF wydajność jest dobra AND krzywa uczenia jest umiarkowana AND elastyczność jest umiarkowana THEN wybierz *Reacta*.

8.2.3 VUE

1. IF wydajność jest słaba AND wydajność wyświetlania dużej ilości treści jest średnia AND rozmiar paczki jest duży THEN nie wybieraj *Vue*.
2. IF wydajność jest dobra AND wydajność wyświetlania dużej ilości treści jest dobra AND rozmiar paczki jest mały THEN wybierz *Vue*.
3. IF wydajność jest dobra AND rozmiar paczki jest mały THEN wybierz *Vue*.
4. IF wydajność jest dobra AND rozmiar paczki jest średni AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Vue*.
5. IF wydajność DOM jest słaba AND wydajność DOM wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest łatwa THEN wybierz *Vue*.
6. IF wydajność jest dobra AND rozmiar paczki jest mały AND wbudowane bezpieczeństwo jest brak THEN wybierz *Vue*.
7. IF wydajność jest średnia AND wydajność wyświetlania dużej ilości treści jest dobra AND rozmiar paczki jest średni AND wbudowane bezpieczeństwo jest brak THEN wybierz *Vue*.
8. IF (krzywa uczenia jest umiarkowana OR krzywa uczenia jest łatwa) AND popularność jest rosnąca THEN wybierz *Vue*.
9. IF wydajność jest słaba AND elastyczność jest sztywna THEN nie wybieraj *Vue*.
10. IF wydajność DOM jest średnia AND (wydajność DOM wyświetlania dużej ilości treści jest średnia OR wydajność DOM wyświetlania dużej ilości treści jest dobra) AND rozmiar paczki jest mały THEN wybierz *Vue*.
11. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND rozmiar paczki jest mały THEN wybierz *Vue*.
12. IF (popularność jest stagnacja OR popularność jest malejąca) THEN nie wybieraj *Vue*.
13. IF bezpieczeństwo jest wbudowane THEN nie wybieraj *Vue*.
14. IF bezpieczeństwo jest wbudowane AND (krzywa uczenia jest łatwa OR krzywa uczenia jest umiarkowana) AND (wydajność jest dobra OR wydajność jest średnia) THEN wybierz *Vue*.

15. IF rozmiar paczki jest duży AND bezpieczeństwo jest wbudowane THEN nie wybieraj *Vue*.
16. IF (rozmiar paczki jest mały OR rozmiar paczki jest średni) AND brak bezpieczeństwa THEN wybierz *Vue*.
17. IF wydajność jest dobra AND rozmiar paczki jest średni AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Vue*.
18. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
19. IF wydajność jest dobra AND rozmiar paczki jest mały AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
20. IF wydajność jest średnia AND rozmiar paczki jest mały AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Vue*.
21. IF wydajność jest dobra AND elastyczność jest umiarkowana AND wydajność DOM wyświetlania dużej ilości treści jest średnia THEN wybierz *Vue*.
22. IF wydajność DOM wyświetlania dużej ilości treści jest dobra AND krzywa uczenia jest łatwa AND popularność jest stagnacja THEN wybierz *Vue*.
23. IF wydajność jest dobra AND rozmiar paczki jest średni AND wbudowane bezpieczeństwo jest brak THEN wybierz *Vue*.
24. IF wydajność jest średnia AND elastyczność jest umiarkowana AND wydajność DOM wyświetlania dużej ilości treści jest dobra THEN wybierz *Vue*.
25. IF wydajność jest dobra AND popularność jest stagnacja AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
26. IF wydajność jest średnia AND rozmiar paczki jest średni AND elastyczność jest umiarkowana THEN wybierz *Vue*.
27. IF wydajność jest średnia AND rozmiar paczki jest średni AND wydajność DOM wyświetlania dużej ilości treści jest średnia THEN wybierz *Vue*.
28. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
29. IF wydajność jest średnia AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
30. IF wydajność jest dobra AND popularność jest rosnąca AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
31. IF wydajność jest średnia AND rozmiar paczki jest średni AND elastyczność jest umiarkowana THEN wybierz *Vue*.
32. IF wydajność jest dobra AND rozmiar paczki jest mały AND wydajność DOM wyświetlania dużej ilości treści jest średnia THEN wybierz *Vue*.
33. IF wydajność jest dobra AND elastyczność jest umiarkowana AND krzywa uczenia jest umiarkowana THEN wybierz *Vue*.
34. IF wydajność jest dobra AND popularność jest rosnąca AND elastyczność jest umiarkowana THEN wybierz *Vue*.
35. IF wydajność jest średnia AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND wbudowane bezpieczeństwo jest brak THEN wybierz *Vue*.
36. IF wydajność DOM wyświetlania dużej ilości treści jest średnia AND krzywa uczenia jest łatwa AND elastyczność jest umiarkowana THEN wybierz *Vue*.
37. IF wydajność jest dobra AND elastyczność jest umiarkowana AND popularność jest rosnąca THEN wybierz *Vue*.
38. IF wydajność jest średnia AND rozmiar paczki jest mały AND popularność jest rosnąca THEN wybierz *Vue*.
39. IF wydajność jest dobra AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND wbudowane bezpieczeństwo jest brak THEN wybierz *Vue*.

40. IF wydajność jest średnia AND wydajność DOM wyświetlania dużej ilości treści jest dobra AND popularność jest rosnąca THEN wybierz *Vue*.
41. IF wydajność jest dobra AND krzywa uczenia jest umiarkowana AND elastyczność jest umiarkowana THEN wybierz *Vue*.

8.3 APLIKACJA DO WYBORU FRAMEWORKU

Aplikacja służąca do wyboru odpowiedniego frameworku została utworzona za pomocą biblioteki *SciKit-Fuzzy* w języku *Python*. Udostępnia ona zestaw narzędzi do implementacji modeli wnioskowania rozmytego.

Do stworzenia modelu wymagane jest zdefiniowanie zmiennych lingwistycznych (wejściowych i wyjściowych), a następnie przypisanie im funkcji przynależności. Te określają stopień, w jakim dane wejściowe są przypisywane do określonych zmiennych.

Kiedy zmienne lingwistyczne i funkcje są już zdefiniowane wymagane jest napisanie zestawu reguł rozmytych, które będą definiować zachowanie systemu. Reguły te przyjmują dwa parametry: funkcje przynależności konkretnych zmiennych lingwistycznych, tworzące zależności typu IF-THEN oraz funkcję wyjścia. Reguły określają jakie działania należy podjąć w zależności od danych wejściowych.

Aplikacja została stworzona jako aplikacja okienkowa. W tym celu użyto bibliotekę *Tkinter* umożliwiającą stworzenie interfejsu graficznego. Pozwala ona na szybkie utworzenie prostego interfejsu, który przyjmuje dane od użytkownika, a po kliknięciu przycisku, wyświetla mu proponowany framework.

W aplikacji zadeklarowane zostały trzy klasy (po jednej dla każdego z frameworków) definiujące logikę oraz jedna do stworzenia okna, w którym wyświetlane są pole tekstowe do uzupełnienia. Każda z klas frameworków podczas inicjalizacji tworzy zmienne lingwistyczne (obiekty klasy *Antecedent* i *Consequent*), przypisuje im odpowiednie funkcje przynależności, a następnie tworzy zestaw reguł. Zestaw ten jest później użyty do stworzenia zmiennej *ControlSystem*, a ta do utworzenia *ControlSystemSmulation*. Obiekt symulacji przyjmuje wartości zmiennych lingwistycznych. Po wywołaniu metody „*compute*" zostaje wykonana symulacja, która zwraca wartości każdego z trzech obiektów *ControlSystemSmulation*. Symulacja o najwyższym wyniku funkcji przynależności zmiennej wyjściowej wskazuje na framework, który może najbardziej odpowiadać wymaganiom użytkownika.

Tworzenie obiektu *Antecendent* wymaga przekazania dwóch parametrów. Pierwszy z nich tworzy zbiór wartości, które może przyjąć dana zmienna lingwistyczna (najczęściej jest to przedział od 0 do 100). Drugi parametr to podpis danej zmiennej lingwistycznej. Obiekt *Consequent* tworzony jest w sposób identyczny do *Antecendent*. Definiowanie funkcji przynależności polega na wywołaniu metody obiektu *fuzz* dostarczonego przez bibliotekę. Metoda ta definiuje wiele metod określających kształt danej funkcji. W przypadku tej aplikacji każda zmienna była określona funkcją trapezową, definiowaną metodą *trampf*, ale możliwe jest zdefiniowanie zmiennej na przykład metodą *trimpf*, która stworzy funkcje trójkątną. Funkcje obiektu *fuzz* wymagają przekazania dwóch parametrów. Pierwszy to pole *universe* zmiennej lingwistycznej, a drugi to lista zawierająca wartości definiujące obszar funkcji. W przypadku funkcji *trampf*, należy przekazać wektor czteroelementowy. Przykładowo, wektor [20, 40, 60, 70] określa trapez zaczynający się dla wartości 20, druga wartość (w tym przypadku 40) oznacza początek górnego boku trapezu. Wartość 60 oznacza koniec górnego boku, a 70 koniec dolnego boku trapezu. Tworzenie reguły wymaga stworzenia obiektu *Rule*, w którym łączone są warunki (antecedenty) z akcjami (consekwentami). Te warunki określane są za pomocą logiki rozmytej. Akcje definiują wartość funkcji wyjściowej. (Listing 8.1). W bibliotece scikit-fuzzy wartość następnika (consekwenta) zwykle obliczana jest przy użyciu operacji min i max.

Okienko aplikacji przedstawia prosty interfejs użytkownika (Rysunek 8.1). Zbudowane jest z obiektów *Label, Entry* oraz jednego przycisku. Po uzupełnieniu pól tekstowych, przyjmujących wartości liczbowe określające wartość zmiennej lingwistycznej, użytkownikowi przedstawiony zostanie wybrany framework po naciśnięciu przycisku. Akcja przycisku wywołuje metodę, która wywołuje symulacje dla każdego frameworku. Symulacje przyjmują wartości podane przez użytkownika. Symulacja frameworku z najwyższym wynikiem jest uznana za najlepszą, a framework przedstawiony użytkownikowi.

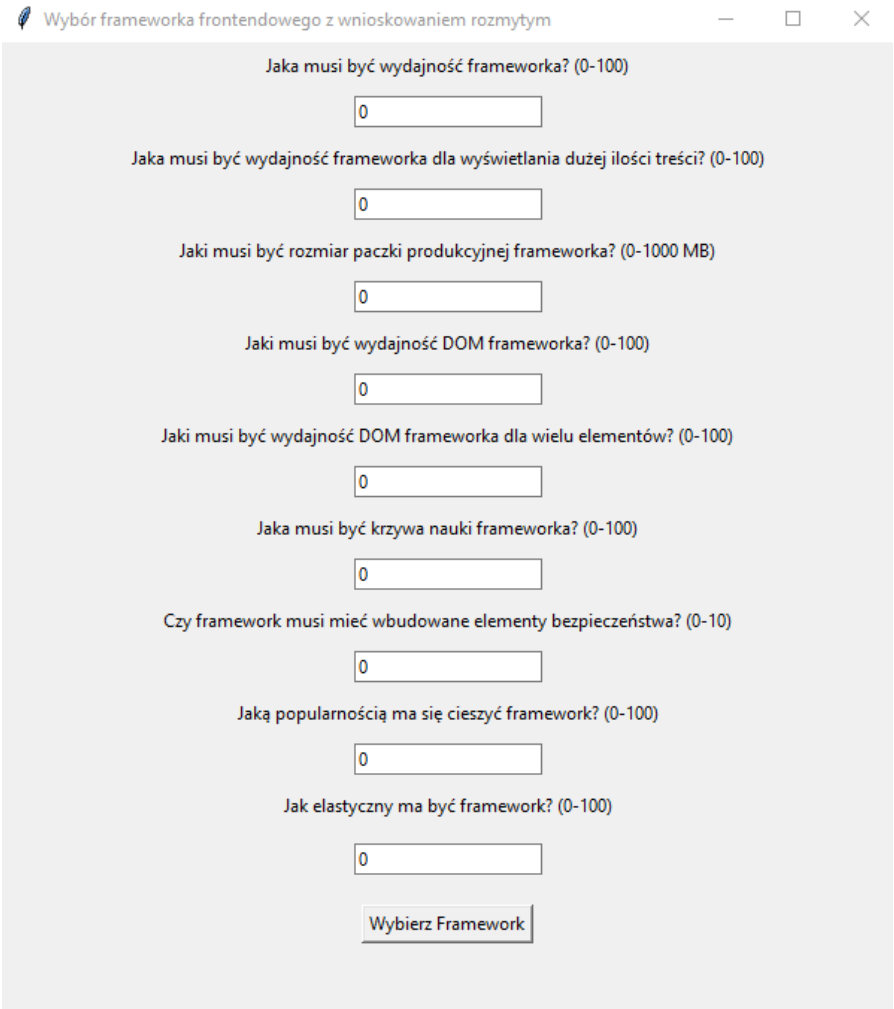
```
self.__performance = ctrl.Antecedent(np.arange(0, 101, 1)
self.__PERFORMANCE_LABEL)
self.__Angular = ctrl.Consequent(np.arange(0, 2, 1), self.__ANGULAR_LABEL)
self.__performance['bad'] = fuzz.trapmf(self.__performance.universe, [0, 0,
20, 40])
self.__performance['average'] = fuzz.trapmf(self.__performance.universe,
[20, 40, 60, 70])
self.__performance['good'] = fuzz.trapmf(self.__performance.universe, [60,
70, 100, 100])
ctrl.Rule(self.__performance['bad'] &
self.__large_data_performance['average'] & self.__package_size['large'],
self.__Angular['choose']
```

Listing 8.1: Przykład definicji zmiennych lingwistycznych, funkcji przynależności oraz reguły dla *wydajności* w *Pythonie*

8.4 WYNIKI DZIAŁANIA APLIKACJI

Poniżej przedstawione zostaną wyniki działania aplikacji służącej do wyboru frameworku, który będzie najbardziej pasował do wymagań użytkownika. Analiza będzie dotyczyła oceny skuteczności systemu wnioskowania rozmytego.

1. **Parametry:**
    - Wydajność: **40**
    - Wydajność wyświetlania dużej ilości treści: **20**
    - Rozmiar paczki: **500**
    - Wydajność DOM: **70**



Rysunek 8.1: Okno aplikacji wyboru frameworku

- Wydajność DOM dla wielu elementów: **40**
- Krzywa nauki: **5**
- Wbudowane bezpieczeństwo: **8**
- Popularność: **5**
- Elastyczność: **4**

Wynik: *Angular*

2. **Parametry:**
   - Wydajność: **80**
   - Wydajność wyświetlania dużej ilości treści: **80**
   - Rozmiar paczki: **200**
   - Wydajność DOM: **40**
   - Wydajność DOM dla wielu elementów: **70**
   - Krzywa nauki: **2**
   - Wbudowane bezpieczeństwo: **2**
   - Popularność: **8**
   - Elastyczność: **9**

Wynik: *React*

3. **Parametry:**
   - Wydajność: **50**

- Wydajność wyświetlania dużej ilości treści: **90**
- Rozmiar paczki: **300**
- Wydajność DOM: **70**
- Wydajność DOM dla wielu elementów: **90**
- Krzywa nauki: **4**
- Wbudowane bezpieczeństwo: **5**
- Popularność: **6**
- Elastyczność: **6**

Wynik: *Angular*

4. **Parametry:**
   - Wydajność: **60**
   - Wydajność wyświetlania dużej ilości treści: **90**
   - Rozmiar paczki: **350**
   - Wydajność DOM: **50**
   - Wydajność DOM dla wielu elementów: **80**
   - Krzywa nauki: **7**
   - Wbudowane bezpieczeństwo: **2**
   - Popularność: **8**
   - Elastyczność: **3**

Wynik: *Vue*

5. **Parametry:**
   - Wydajność: **20**
   - Wydajność wyświetlania dużej ilości treści: **30**
   - Rozmiar paczki: **100**
   - Wydajność DOM: **90**
   - Wydajność DOM dla wielu elementów: 5**0**
   - Krzywa nauki: **5**
   - Wbudowane bezpieczeństwo: **5**
   - Popularność: **7**
   - Elastyczność: **2**

Wynik: *Angular*

6. **Parametry:**
   - Wydajność: **85**
   - Wydajność wyświetlania dużej ilości treści: **70**
   - Rozmiar paczki: **100**
   - Wydajność DOM: **20**
   - Wydajność DOM dla wielu elementów: **70**
   - Krzywa nauki: **5**
   - Wbudowane bezpieczeństwo: **1**
   - Popularność: **3**
   - Elastyczność: **9**

Wynik: *React*

7. **Parametry:**
   - Wydajność: **35**
   - Wydajność wyświetlania dużej ilości treści: **75**
   - Rozmiar paczki: **500**
   - Wydajność DOM: **10**

- Wydajność DOM dla wielu elementów: **100**
- Krzywa nauki: **1**
- Wbudowane bezpieczeństwo: **1**
- Popularność: **9**
- Elastyczność: **9**

Wynik: *Angular*

8. **Parametry:**
   - Wydajność: **55**
   - Wydajność wyświetlania dużej ilości treści: **75**
   - Rozmiar paczki: **100**
   - Wydajność DOM: **10**
   - Wydajność DOM dla wielu elementów: **100**
   - Krzywa nauki: **1**
   - Wbudowane bezpieczeństwo: **1**
   - Popularność: **9**
   - Elastyczność: **9**

Wynik: *Vue*

9. **Parametry:**
   - Wydajność: **50**
   - Wydajność wyświetlania dużej ilości treści: **90**
   - Rozmiar paczki: **150**
   - Wydajność DOM: **5**
   - Wydajność DOM dla wielu elementów: **90**
   - Krzywa nauki: **2**
   - Wbudowane bezpieczeństwo: **3**
   - Popularność: **9**
   - Elastyczność: **9**

Wynik: *Vue*

10. **Parametry:**
    - Wydajność: **50**
    - Wydajność wyświetlania dużej ilości treści: **90**
    - Rozmiar paczki: **300**
    - Wydajność DOM: **5**
    - Wydajność DOM dla wielu elementów: **9**
    - Krzywa nauki: **2**
    - Wbudowane bezpieczeństwo: **9**
    - Popularność: **6**
    - Elastyczność: **4**

Wynik: *Angular*

8.5 WNIOSKI

Na podstawie testów aplikacji do wyboru odpowiedniego frameworku, zostały wyciągnięte wnioski opisane w tym podrozdziale.

W pierwszym teście parametry wskazały *Angulara* jako najbardziej pasujący framework. *Angular* został wybrany ze względu na dobre wbudowane zabezpieczenia i wysoki bufor dotyczący rozmiaru paczki produkcyjnej. Istotne było też pozwolenie na małą elastyczność i stagnacje popularności. W drugim teście aplikacja wskazała *Reacta* ze

względu na wysoką wydajność i elastyczność. Znaczące były też brak wbudowanego bezpieczeństwa oraz prosta krzywa nauki. Trzeci test wskazał *Angulara* ze względu na tolerancję dużego rozmiaru paczki produkcyjnej, wysoką wydajność DOM oraz średnią popularność i elastyczność. Czwarty test wskazał *Vue* jako najlepszy framework, ze względu na wysoką popularność oraz wydajność wyświetlania dużej ilości treści. *Vue* został wybrany mimo dużego rozmiaru paczki oraz średnią krzywą nauki. Piąty test wybrał *Angulara* jako najlepszy framework. Argumentami ku tej decyzji była wysoka wydajność DOM oraz preferencje narzuconych praktyk, bez dużej elastyczności. Tolerancja niskiej wydajności upewniła decyzję modelu. Wynikiem szóstego testu był framework *React,* wybrany ze względu na wysoką wydajność, elastyczność oraz brak wbudowanego bezpieczeństwa. Tolerancja niskiej wydajności DOM nie wpłynęła na wybór narzędzia. Siódmy test wskazał *Angular* jako najlepszy framework. Wpłynęła na to tolerancja słabej wydajności, słabej wydajności DOM oraz dużego rozmiaru paczki. Wynik ósmego testu to *Vue*. Wybór uzasadniony jest wysoką popularnością, elastycznością, wydajnością DOM dla wielu elementów, a także minimalną krzywą nauki oraz brakiem wbudowanego bezpieczeństwa. Parametry testu dziewiątego były zbliżone do parametrów ósmego testu, co spowodowało wybór *Vue*. Ostatni test wybrał *Angulara* jako najlepszy framework. Został wybrany ze względu na wysokie wbudowane bezpieczeństwo, średnią wydajność, elastyczność i popularność.

Wyżej przedstawione wyniki potwierdzają wyniki badań artykułów porównujących frameworki. Pierwsze cztery, a także szósty, ósmy i dziewiąty test wskazuje na wyniki podobne do tych znajdujących się w artykułach [23] i [25]. Wbudowane bezpieczeństwo i wysoka wydajność DOM to jedne z największych zalet frameworka Angular, stąd jego wybór dla testów, które najmocniej wskazywały na te parametry. Wybór Reacta jest oczekiwany, gdy wydajność aplikacji oraz elastyczność mają być wysokie. Vue jest dość nowy na rynku, co powoduje, że zostanie wybrany w momencie, gdy parametry wskazują na wysoką popularność i wydajność w wyświetlaniu dużej ilości treści.

Testy piąty i siódmy potwierdzają wyniki artykułów [22] i [25]. Aplikacja wskaże na Angulara, jako najbardziej odpowiedniego frameworka., gdy podane parametry wskazują wysoką wydajność DOM, małą elastyczność oraz tolerancję dużego rozmiaru paczki.

Ostatni test był zgodny ze wszystkimi artykułami podanymi powyżej. Jeśli uśrednimy wyniki wydajności, elastyczności czy popularności, a będziemy wymagać wbudowanego bezpieczeństwa, aplikacja wskaże Angulara, jako najlepszy wybór.

Analiza wyników wskazuje, że zdefiniowanie większej liczby szczegółowych reguł dla każdego frameworku może spowodować bardziej precyzyjny wynik. Obecnie model implementuje zestaw ogólnych reguł, które mogą skutkować podobnym wynikiem funkcji wyjścia, co czasem może powodować niedokładny wynik. Dodatkowe reguły mogłyby uwzględniać bardziej szczegółowe i złożone zależności. Większa liczba reguł może spowodować zwiększenie satysfakcji z korzystania aplikacji.

.

9 PODSUMOWANIE

W niniejszej pracy przeprowadzona została wielokryterialna analiza trzech popularnych frameworków frontendowych: *Angular, React* oraz *Vue*. Celem pracy było zbudowanie aplikacji serwisu ogłoszeniowego sprzedaży samochodów za pomocą każdego z nich. Następnie porównano te narzędzia pod kątem wydajności, rozmiaru paczek produkcyjnych, struktury aplikacji, implementacji DOM, popularności, wsparcia społeczności oraz bezpieczeństwa. Do badań wykorzystano wiele narzędzi, takich jak *Docker, DigitalOcean,* stronę *webpagetest.org* oraz *Sonarqube*. Pierwsze narzędzie pozwoliło na wdrożenie testowanych aplikacji na maszynę hostowaną w chmurze przez *DigitalOcean*. To pozwoliło na użycie pozostałych narzędzi w celu analizy kodu oraz analizy wydajności strony pod względem szybkości renderowania i ładowania treści.

Analiza wykazała, że *React* i *Vue* charakteryzują się lepszą wydajnością w przypadku ładowania i renderowania treści w porównaniu do *Angulara*. *Vue* osiągnął najlepsze wyniki w większości testów manipulacji DOM. *Angular* miał najdłuższe czasy ładowania i renderowania, jednak mógł pochwalić się najszybszymi czasami manipulacji pojedynczymi elementami DOM. Najmniejsze paczki produkcyjne generuje *Vue*. Jest to korzystne dla wydajności na urządzeniach mobilnych i starszych komputerach. *Angular* dostarczany jest z największą ilością wbudowanych funkcji, powoduje to największe paczki produkcyjne z trzech omawianych frameworków.

Analiza złożoności kodu za pomocą *Sonarqube* wskazała, że *Angular* ma największą złożoność cykliczną i WMC (ang. *Weighted methods per class)* w porównaniu do *Reacta* i *Vue*. Oznacza to większą modularność komponentów *Angulara*. Istotnym aspektem jest bezpieczeństwo. *Angular* wyróżnia się wbudowanymi mechanizmami ochrony, takimi jak sanityzacja danych, minimalizująca ryzyko ataków *Cross-site scripting*. *React* i *Vue* są bardziej elastycznymi frameworkami pod tym względem. Wymaga to jednak większej świadomości dotyczącej zasad bezpieczeństwa od programistów

9.1 WNIOSKI

Wyniki analizy wskazują, że wybór odpowiedniego frameworku może zależeć głównie od specyficznych potrzeb projektu. W wypadku, gdy potrzebna jest wysoka wydajność i elastyczność, preferowany jest wybór *Reacta* bądź *Vue*. Te frameworki idealnie nadadzą się do projektów wymagających szybkiego renderowania i dynamicznej manipulacji dużą ilością treści DOM. *Angular* jest proponowany dla projektów, które wymagają wysokiego poziomu bezpieczeństwa oraz modularności komponentów.

Rozmiar paczek produkcyjnych generowanych przez *Vue* sprawia, że jest on idealny dla aplikacji muszących działać sprawnie na urządzeniach mobilnych i starszych komputerach. *React* posiada wiele zasobów edukacyjnych i gotowych rozwiązań, dzięki swojej popularności i wsparciu społeczności. To może znacząco wpłynąć na czas tworzenia aplikacji.

Podsumowując, *React* i *Vue* jest idealnym wyborem dla aplikacji webowych wymagających dużej elastyczności i dobrej wydajności. *Angular* jest dojrzałą, ustrukturyzowaną technologią z bezpieczną architekturą. Sprawdzi się on dla bardziej złożonych projektów, których priorytetem jest bezpieczeństwo i modularność kodu.

Model wnioskowania rozmytego pozwolił na uwzględnienie mierzalnych i niemierzalnych parametrów, takich jak czas ładowania i opinie programistów. Dzięki temu udało się dostarczyć narzędzie wspomagające decyzję o wyborze odpowiedniego frameworku frontendowego.

9.2 KIERUNKI ROZWOJU PRACY

BIBLIOGRAFIA

[1] TOBIAŃSKA, MONIKA, AND JAKUB SMOŁKA. "EFEKTYWNOŚĆ TWORZENIA WARSTWY PREZENTACJI APLIKACJI WE FRAMEWORKACH *ANGULAR*JS, *ANGULAR*2, BACKBONEJS." JOURNAL OF COMPUTER SCIENCES INSTITUTE 8 (2018): 226-229.

[2] NAWROCKI, J., & OLEK, Ł. OPISYWANIE PROCESÓW BIZNESOWYCHZ WYKORZYSTANIEM PRZYPADKÓW UŻYCIA.

[3] EL-BAKRY, HAZEM M., ET AL. "ADAPTIVE USER INTERFACE FOR WEB APPLICATIONS." RECENT ADVANCES IN BUSINESS ADMINISTRATION: PROCEEDINGS OF THE 4TH WSEAS INTERNATIONAL CONFERENCE ON BUSINESS ADMINISTRATION (ICBA'10). 2010.

[4] PETZOLD, CHARLES. PROGRAMMING MICROSOFT WINDOWS WITH *C#*. REDMOND, WASHINGTON: MICROSOFT PRESS, 2002.

[5] JANSEN, REMO H., VILIC VANE, AND IVO GABE DE WOLFF. *TYPESCRIPT*: MODERN *JAVASCRIPT* DEVELOPMENT. PACKT PUBLISHING LTD, 2016.

[6] FLANAGAN, D. (2020). JAVA-SCRIPT: THE DEFINITIVE GUIDE.

[7] BAMPAKOS A., DEELEMAN P., POZNAJ *ANGULAR*. RZECZOWY PRZEWODNIK PO TWORZENIU APLIKACJI WEBOWYCH Z UŻYCIEM FRAMEWORKU *ANGULAR* 15. WYDANIE IV, 2023

[8] BANKS, A., & PORCELLO, E. (2017). LEARNING *REACT*: FUNCTIONAL WEB DEVELOPMENT WITH *REACT* AND REDUX. " O'REILLY MEDIA, INC."

[9] PABLO DAVID GARAGUSO, *VUE*.JS 3 DESIGN PATTERNS AND BEST PRACTICES¸ 2023

[10] LOCK, ANDREW. ASP. NET CORE IN ACTION. SIMON AND SCHUSTER, 2023.

[11] PATTANKAR, MITHUN, AND MALENDRA HURBUNS. MASTERING ASP. NET WEB API. PACKT PUBLISHING LTD, 2017.

[12] THE *POSTGRESQL* GLOBAL DEVELOPMENT GROUP, *POSTGRESQL* 15.1 DOCUMENTATION

[13] CARLSON, JOSIAH. *REDIS* IN ACTION. SIMON AND SCHUSTER, 2013.

[14] ANDERSON, C. (2015). *DOCKER* [SOFTWARE ENGINEERING]. IEEE SOFTWARE, 32(3), 102-C3.

[15] GKATZIOURAS, EMMANOUIL. A DEVELOPER'S ESSENTIAL GUIDE TO *DOCKER COMPOSE*: SIMPLIFY THE DEVELOPMENT AND ORCHESTRATION OF MULTI-CONTAINER APPLICATIONS. PACKT PUBLISHING LTD, 2022.

[16] https://www.ngxs.io/concepts/store (data dostępu: 11.05.2024 r.)

[17] https://pl.*React*js.org/docs/hooks-overview.html (data dostępu: 04.12.2022 r.)

[18] https://redux.js.org/ (data dostępu 12.05.2024 r.)

[19] https://redux-toolkit.js.org/ (data dostępu 12.05.2024 r.)

[20] BAIDA, ROMAN, MAKSYM ANDRIIENKO, AND MAŁGORZATA PLECHAWSKA-WÓJCIK. "ANALIZA PORÓWNAWCZA WYDAJNOŚCI FRAMEWORKÓW *ANGULAR* ORAZ *VUE*. JS." JOURNAL OF COMPUTER SCIENCES INSTITUTE 14 (2020): 59-64.

[21] LENARDUZZI, VALENTINA, ET AL. "ARE SONARQUBE RULES INDUCING BUGS?." 2020 IEEE 27TH INTERNATIONAL CONFERENCE

ON SOFTWARE ANALYSIS, EVOLUTION AND REENGINEERING (SANER). IEEE, 2020.
[22] LEVLIN, MATTIAS. "DOM BENCHMARK COMPARISON OF THE FRONT-END *JAVASCRIPT* FRAMEWORKS *REACT*, *ANGULAR*, *VUE*, AND SVELTE." (2020)
[23] CINCOVIĆ, JELICA, AND MARIJA PUNT. "COMPARISON: *ANGULAR* VS. *REACT* VS. *VUE*. WHICH FRAMEWORK IS THE BEST CHOICE." UNIVERSIDAD DE BELGRADE (2020).
[24] https://*Angular*.io/guide/security (data dostępu 27.05.2024 r.)
[25] "EVALUATING THE PERFORMANCE OF WEB RENDERING TECHNOLOGIES BASED ON *JAVASCRIPT*: *ANGULAR*, *REACT*, AND *VUE*." (2022)
[26] S., C., SWATHI, M. "FUZZY LOGIC." (2023)

SPIS RYSUNKÓW

SPIS LISTINGÓW

SPIS TABEL