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 ciągu ostatnich kilku lat, z powodu epidemii Covid-19, duża część społeczeństwa znacznie ograniczyła swoją aktywność fizyczną. Jest to spowodowane między innymi pracą zdalną oraz nawykami niewychodzenia z domu. Od niedawna część z nich, w obawie o swoje zdrowie oraz z powodu chęci poprawienia swojej kondycji, zaczęła uprawiać sport w różnych formach, takich jak spacery, jazda na rowerze czy chodzenie na siłownię. Ta ostatnia grupa spotyka się z problemem dobierania ćwiczeń do treningu, a co za tym idzie, przeszukuje różne strony internetowe o tematyce siłowni, w celu znalezienia odpowiedniego dla siebie planu treningowego.

W pracy przedstawiono planowanie, projektowanie oraz implementację aplikacji internetowej zbudowanej przy pomocy narzędzi *ReactJS* oraz *Springboot*. Aplikacja ma pozwolić użytkownikowi na wygenerowanie planu treningowego w zależności od sylwetki, wieku oraz stopnia zaawansowania. Będzie to możliwe dzięki dużej liczbie ćwiczeń znajdujących się w bazie danych *PostgreSQL*, która zostanie skonfigurowana za pomocą narzędzia konteneryzacji – *Dockera*. Ćwiczenia uzupełniać będzie *Flyway*, czyli narzędzie migracji bazy danych.

Omówiono również narzędzia użyte do stworzenia aplikacji. Najważniejsze części kodu zostały opisane i przedstawione na listingach. Na koniec zaprezentowano testy najważniejszych funkcjonalności.

ABSTRACT

In the last few years, due to Covid-19 pandemic, a large part of society has significantly reduced its physical activity. Home office and the habit of staying at home are two of the reasons why it happens. Recently a part of that society in fear of health and willingness to improve physical condition, has engaged in sports like going for a walk, biking, or going to the gym. The last activity mentioned may cause problems when selecting exercises to create a workout plan. To solve this problem many people are searching the Web to find a site that will help them find the best plan.

In this paper planning, designing and implementation of web application are presented. The application will be built using *ReactJS* and *Springboot* and its main goal is to generate a training plan based on figure, age, and advancement in training. It will be possible thanks to enormous number of exercises contained in the *PostgreSQL* database, which is going to be configured using *Docker* containerization. Exercises will be filled using *Flyway* – a database migration tool.

Tools used to create application have also been discussed in this paper. The most significant portions of the code will be described and presented in listings. Ultimately, tests of key functionalities will be introduced.

SPIS TREŚCI

1 Wprowadzenie 7

1.1 Przegląd aktualnych rozwiązań 9

2 Faza projektowa 10

2.1 Wymagania funkcjonalne 10
2.2 Wymagania niefunkcjonalne 10
2.3 Diagram ER 10
2.4 Diagram przypadków użycia 13
2.5 Opis i definicja scenariuszy przypadków użycia 14

3 Interfejs użytkownika 17

Widok strony głównej 17
Widok szczegółów oferty 17
Widok przeglądarki ofert 18
Widoki ekranu tworzenia ogłoszenia 18

4 Wykorzystane technologie 21

4.1 Wybrane języki 21
4.2 Narzędzia strony interfejsu użytkownika 22

ReactJS 22
Angular 22
Vue 22
Biblioteki do budowania komponentów 23

4.3 Narzędzia strony serwerowej 23

ASP.NET Core 23
Model-View-Controller (MVC) 23
Web API 23

4.4 Narzędzia bazy danych 24

PostgreSQL 24
Redis 24

4.5 Narzędzia Konteneryzacji 24

Docker 25
Docker Compose 25

5 Implementacja aplikacji 26

5.1 Środowiska i narzędzia programistyczne 26
5.2 Podział na pakiety i strony 26

Backend 26
Frontend 27

5.3 Implementacja i konteneryzacja bazy danych 28

5.4 Konteneryzacja serwera ... 30
5.5 Implementacja serwera ... 31
5.6 Implementacja interfejsu użytkownika ... 35
5.7 Implementacja serwera komunikującego się z bazą danych redis ... 41
6 Analiza porównawcza frameworków frontendowych ... 42
6.1 Wydajność ... 42
6.1.1 Wydajność aplikacji otoauto ... 42
6.1.2 Test pustej aplikacji ... 47
6.1.3 Test wypełnionej aplikacji ... 50
6.2 Rozmiar paczek produkcyjnych ... 52
6.3 Struktura aplikacji ... 52
6.4 Implementacja DOM ... 53
6.5 Popularność i wsparcie społeczności ... 54
6.6 Bezpieczeństwo ... 55
6.7 Podsumowanie ... 55
7 Podsumowanie ... 59
7.1 Wnioski ... 59
7.2 Koncepcja rozwoju aplikacji ... **Błąd! Nie zdefiniowano zakładki.**
Bibliografia ... 60
Spis rysunków ... 62
Spis listingów ... 63

WSTĘP

Podczas pracy jako programista aplikacji internetowych w 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 „Który narzędzie do pisania aplikacji internetowych jest najlepszy?". 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 zasadach. W niniejszej pracy przedstawiono proces budowania aplikacji, porównywania narzędzi w wielu aspektach oraz tworzenia zbioru rozmytego, który będzie miał na celu uproszczenie wyboru odpowiedniego narzędzia.

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 pod aspektem:

- 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 frontedowe 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,

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

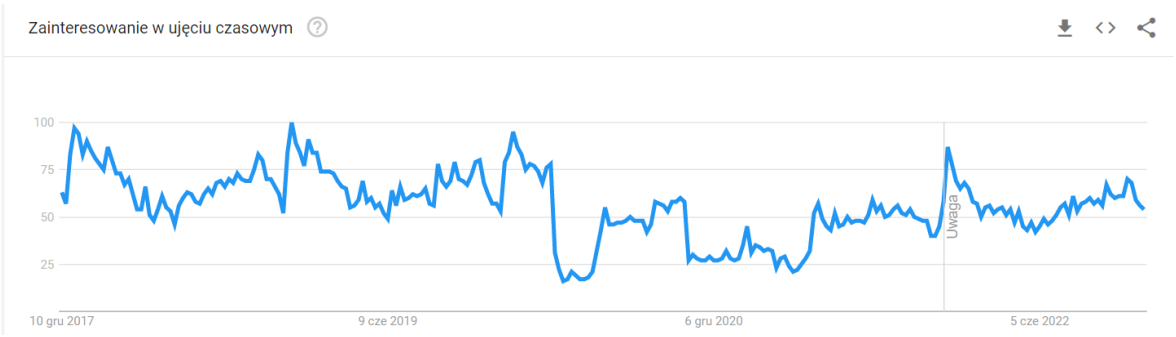
skończonego projektu. Rozmiar zmieni się też dla paczki produkcyjnej. Porównane zostaną popularność, bezpieczeństwo i sposób implementacji DOM.

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

Wyniki badań przedstawione w raporcie *Polski rynek sportu* Polskiego Instytutu Ekonomicznego z 2019 r. pokazały, że zaledwie 28% Polaków uprawia regularnie sport. Spowodowane pandemią Covid-19, przejście znacznej części społeczeństwa na pracę zdalną wpłynęło na dodatkowe ograniczenie aktywności fizycznej. Przyjmuje się, że odsetek ludzi uprawiających sport lub inną aktywność fizyczną przynajmniej raz w tygodniu zmalał aż o 16%. Wykazano też, że spacery są najczęściej uprawianą formą aktywności fizycznej. Z przeprowadzonych badań społecznych zauważono, że prawie 50% uczestników badania ćwiczy w celu poprawy samopoczucia lub zdrowia. Niecałe 25% ćwiczyło, by uniknąć problemów ze zdrowiem. Udowodnione zostało, że sytuacja epidemiczna, skutkująca przejściem na pracę i naukę zdalną spowodowała spadek aktywności fizycznej wśród uczestników badania. Aż 13% badanych nie uprawiało żadnego sportu, a tylko 9% ćwiczyło w przerwach pomiędzy pracą. **Błąd! Nie można odnaleźć źródła odwołania.**

Ludzie w momencie wybuchu pandemii na krótki czas przestali interesować się siłownią (Rysunek 1.1). Od tamtego momentu znaczny wzrost zainteresowania siłownią jest zauważalny jedynie podczas okresu świąt Bożego Narodzenia, gdzie mniejszy stopień zainteresowania w grudniu, obfituje wysokim jego poziomem w styczniu. Oprócz tych miesięcy, siłownią interesuje się średnio taka sama liczba osób.



Rysunek 1.1: Popularność siłowni w Polsce na przestrzeni lat 2017-2022

Proces budowy strony internetowej składa się z sześciu etapów. Pierwszym z nich jest badanie potrzeb i analiza, której zadaniem jest ocenienie zapotrzebowania danego produktu oraz wymagań postawionych przez grupę docelową. Jednym z zadań analizy jest dokładne przeanalizowanie produktów oferowanych przez konkurencyjne firmy. Krokiem drugim jest planowanie projektu. Plan powinien zawierać przeanalizowane ważne kwestie, takie jak funkcjonalności produktu, kosztorys oraz nieprzekraczalny termin wykonania. Po planowaniu następuje projekt informatyczny. Gdy cele projektu zostały już określone następuje zebranie ich w całość w odpowiednim dokumencie. Ten krok jest ważny, ponieważ stworzenie czytelnej i zrozumiałej dokumentacji na początku procesu budowy ułatwia programistom implementacje rozwiązania z zachowaniem najwyższych standardów w dalszej części prac nad stroną internetową. Po ukończeniu dokumentacji należy wybrać właściwą firmę lub osoby, które podejmą się wykonania produktu. Wybór powinien zostać podjęty po dokładnym przeanalizowaniu portfolio partnera oraz opinii na jego temat, co wymaga odpowiednich kwalifikacji z zakresu zarządzania zasobami ludzkimi. Po podjęciu decyzji dotyczącej zespołu, który zaimplementuje aplikację, należy wybrać technologie, które pozwolą na wytworzenie wysokiej jakości produktu, który będzie spełniać najnowsze

standardy implementacyjne i bezpieczeństwa. Ostatnim krokiem jest zaangażowanie ludzi tworzących produkt. Zgrany i prawidłowo komunikujący się zespół szybko rozwiąże każdy napotkany problem, a oddane rozwiązanie będzie działać prawidłowo, szybko i efektownie. **Błąd! Nie można odnaleźć źródła odwołania.**

Zgodnie z jednym z punktów procesu budowy strony internetowej najważniejszym w budowaniu strony internetowej jest wybór odpowiednich technologii, który zależy od kilku czynników.

Pierwszym z nich jest użyteczność narzędzia w zależności od wyznaczonych celów i problemów, które trzeba rozwiązać. Każda technologia pozwala na zrealizowanie innych zagadnień. Nie powinno się wybierać języka programowania, który na samym początku postawi przed deweloperem trudności związane z problemami implementacji danego rozwiązania. Kolejnym czynnikiem jest znajomość technologii. Większość projektów informatycznych można zbudować za pomocą różnorodnych narzędzi. Języki programowania, frameworki i biblioteki różnią się od siebie. Jedna technologia może być wspierana lepiej niż inna. Znajomość takich różnic pozwoli na odpowiedni wybór. Popularność to ważny element, którego nie wolno pominąć przy wybieraniu technologii. Popularne technologie gromadzą wokół siebie dużą ilość ludzi z pasją do programowania przy użyciu danego narzędzia. Jest to ważne z jednego, trywialnego powodu. Ważkość tego zagadnienia można przedstawić za pomocą przykładu – pisanie skryptów w języku *Bash* oraz języku *Python*. Pisząc kod w *Pythonie* znacznie łatwiej jest znaleźć rozwiązanie danego problemu w sieci. Natomiast *Bash* budzi mniejsze zainteresowanie społeczności, co powoduje, że w przypadku napotkania błędu jest większa szansa, że nie uda się szybko odnaleźć odpowiedzi. Istotnym elementem, na który trzeba zwrócić uwagę jest też dojrzałość i stabilność technologii rozumianą przez jakość narzędzia, ilość dostępnych frameworków i bibliotek. Gdy rozwiązanie jest stabilne, tworzy się wokół niego społeczność, która dba o jego rozwój. Nie ma wtedy obaw o to, że technologia przestanie być wspierana w ciągu kilku następnych lat. Skalowalność produktu jest cechą poszukiwaną przez firmy, które zamierzają go w przyszłości rozwijać. Przy wyborze technologii należy upewnić się, czy pozwala ona na proste i szybkie skalowanie rozwiązania. Budując produkt należy myśleć o przyszłości, ponieważ problem ze skalowalnością będzie generował duże koszty związane z refaktoryzacją kodu. Zastosowana technologia w dużym stopniu wpływa na koszt budowy projektu. Jest to spowodowane między innymi stawkami godzinowymi specjalistów danej dziedziny. Wysokie stawki powodują wzrost kosztów implementowania rozwiązania. Czas implementowania danego rozwiązania może różnić się w zależności od wybranej technologii. Porównując *Javę* i *Pythona* można zauważyć, że ten sam kod zostanie zaimplementowany szybciej w języku *Python*. Jest to spowodowane jego charakterystyczną zwięzłą i czytelną składnią, która sprawia, że programowanie postępuje znacznie szybciej. Poza samym programowaniem zmienia się też czas potrzebny na wykonywanie czynności naprawiających błędy programu. Wysokie wynagrodzenie specjalisty może być zrekompensowane używając technologii, w której rozwiązania implementuje się szybciej. **Błąd! Nie można odnaleźć źródła odwołania.**

1.1 PRZEGLĄD AKTUALNYCH ROZWIĄZAŃ

W sieci nie istnieje strona internetowa, która umożliwiałaby generowanie planu treningowego. Istnieją aplikacje o tematyce ćwiczeń i siłowni, które oferują jego ułożenie. Niestety otrzymanie takiego planu będzie wymagało dużych nakładów pieniężnych.

Przykładem takiej strony jest *fabrykasily.pl* (Rysunek 1.2), która za opłatą oferuje układanie indywidualnego planu treningowego wraz z dietą i wsparciem trenera personalnego.

Rysunek 1.2. Strona główna fabrykasily.pl

Jednak przy kliku tysiącach użytkowników trudno jest, by faktycznie stosować indywidualne podejście do klienta. Nie każda osoba aktywna fizycznie może pozwolić sobie na dietę oraz opłatę za plany treningowe co miesiąc.

Stroną oferującą pomoc w doborze ćwiczeń jest *budujmase.pl*. Użytkownik może za opłatą otrzymać jedną z 3 ofert. Dostępne do wyboru są plan treningowy i dietetyczny, tylko plan treningowy lub tylko plan dietetyczny. Z powtarzających się wad ze strony *fabrykasily.pl* należy podkreślić dwukrotnie większą sumę do zapłaty za wybraną ofertę (**Błąd! Nie można odnaleźć źródła odwołania.**).

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, opis ogłoszenia czy informacje o dealerze. 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 rozszerzeniem 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 zdjęciu powinien zostać stworzony wspólny wolumin, z którego kontener będzie pobierał zdjęcia z dysku 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. Testy jednostkowe aplikacji oraz logowanie błędów po stronie serwera są również niezbędne.

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.

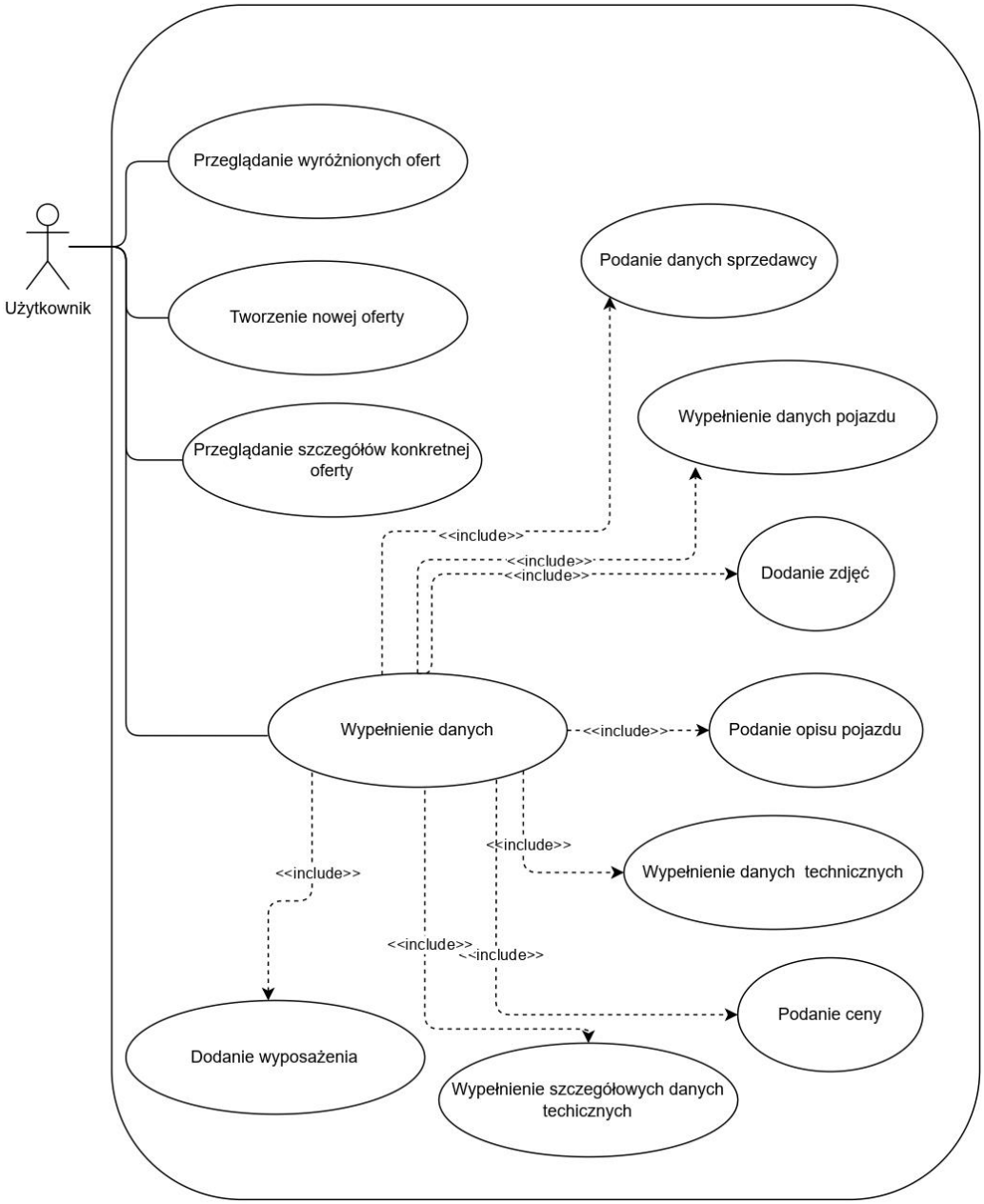
Każdy pojazd ma wyposażenie. W przypadku tego systemu tabela *Vehicle* jest w relacji jeden-do-wielu z tabelą krzyżową *Vehicle_Equipment* posiadającą atrybuty łączące id tabeli pojazdu oraz tabeli wyposażenia. Tabela krzyżowa jest znanym sposobem omijania relacji wiele-do-wielu, która normalnie występowałaby 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:

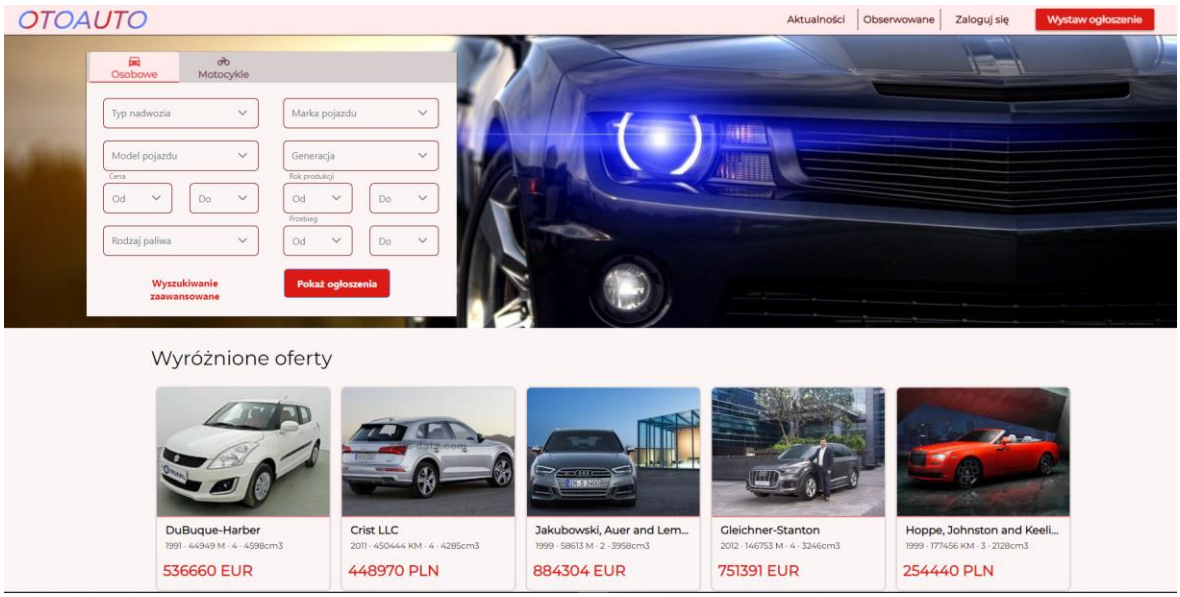
1. Użytkownik rozwija zakładkę „Wyposażenie” w sekcji „Wyświetl dodatkowe szczegóły.
2. Użytkownik rozwija zakładki znajdujące się w zakładce „Wyposażenie”.
3. Użytkownik zaznacza wyposażenie swojego pojazdu.

3 INTERFEJS UŻYTKOWNIKA

Design interfejsu użytkownika ma wpływa 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 starsze osoby, które z niej korzystają mogą mieć problem, by się po niej poruszać. Skomplikowany i nieczytelny interfejs może powodować frustracje. By jej uniknąć aplikacja została stworzona ze zrozumiałych, dużych i podpisanych komponentów.

WIDOK STRONY GŁÓWNEJ

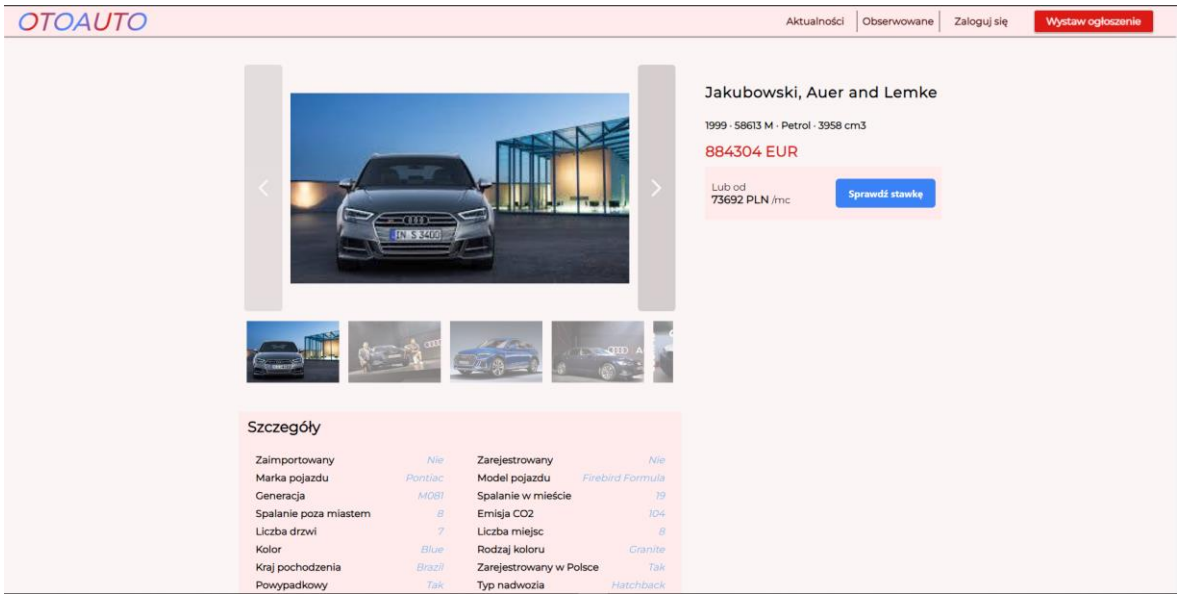
Po wejściu na stronę główną użytkownik ma do wyboru wyszukanie interesujących go marek samochodu używając wyszukiwarki znajdującej się po lewej stronie ekranu. Może on też przeglądać wyróżnione oferty znajdujące się w dolnej części strony. Kliknięcie w którąś z ofert spowoduje przeniesienie użytkownika 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

WIDOK SZCZEGÓŁÓW OFERTY

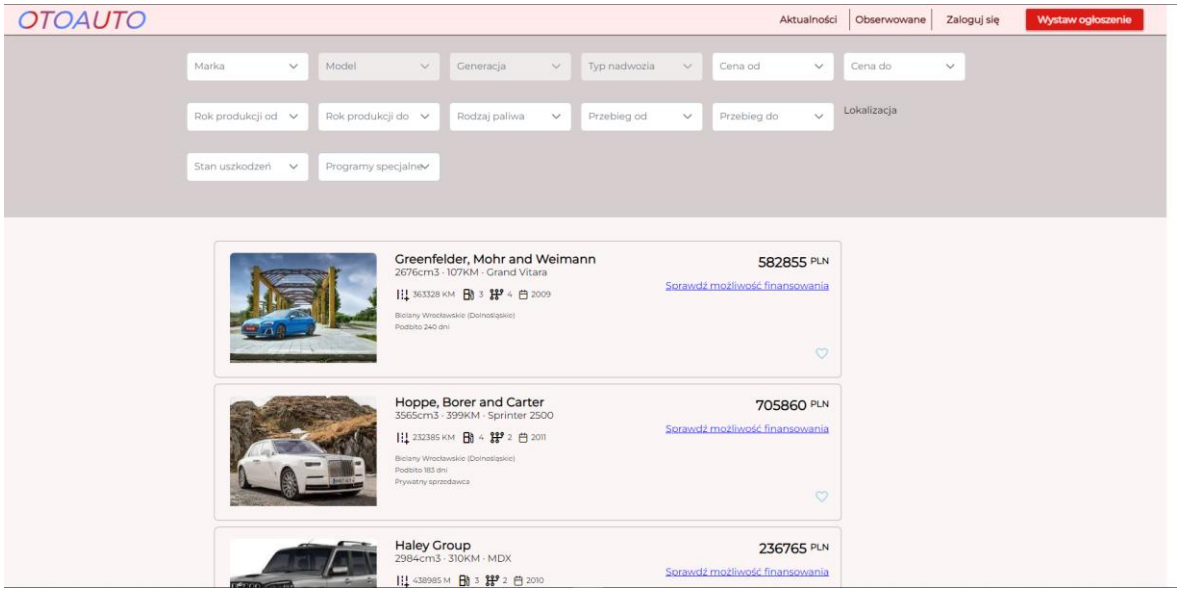
Głównym elementem ekranu szczegółów oferty jest galeria zdjęć, która pozwala na zapoznanie się z wyglądem i stanem pojazdu. Pod zdjęciami znajduje się komponent wyświetlający szczegóły danego pojazdu. Informacje zawarte w tym miejscu dotyczą marki, modelu, mocy czy roku produkcji sprzedawanej maszyny. 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

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).

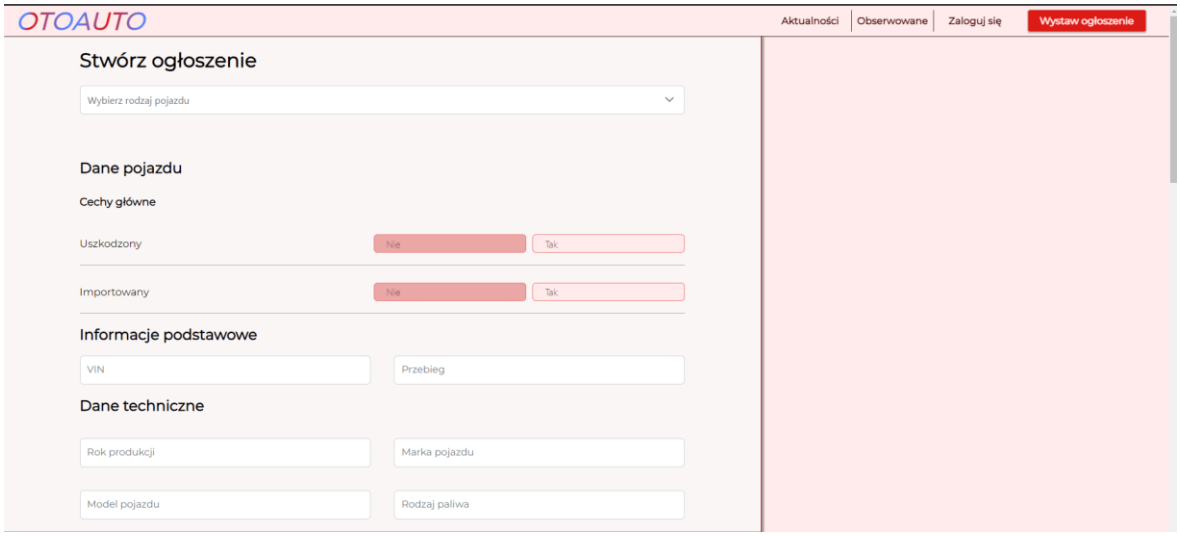


Rysunek 3.3: Widok strony przeglądania ofert

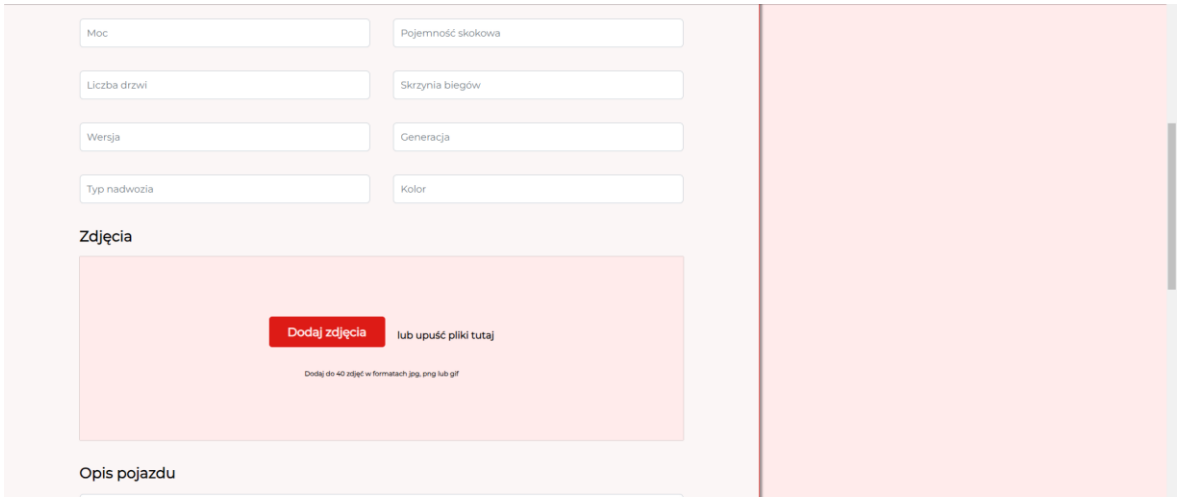
WIDOKI EKRANU TWORZENIA OGŁOSZENIA

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:

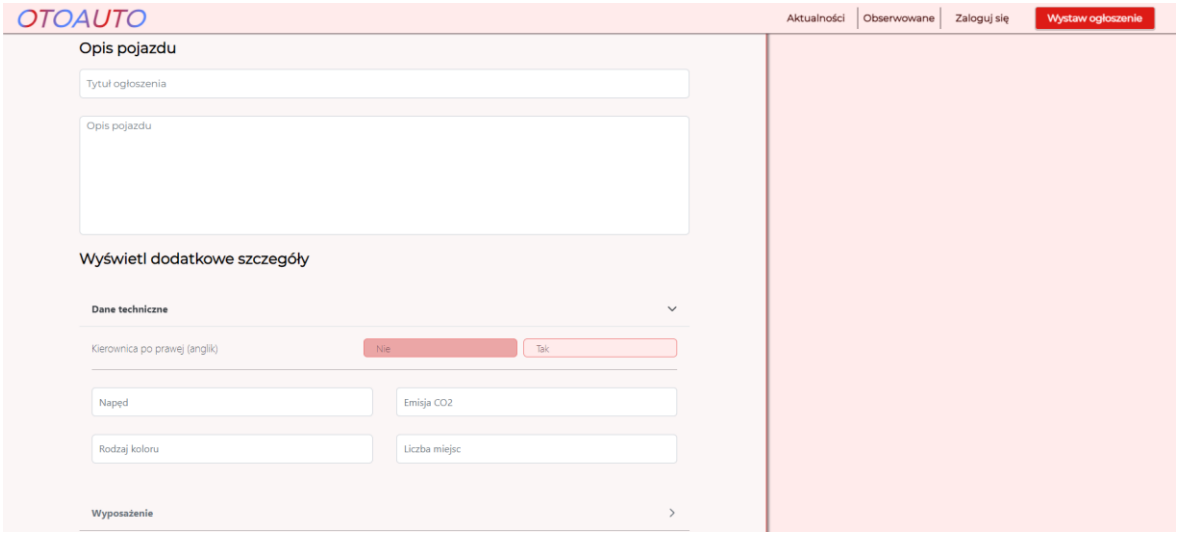
- „*Zdjęcia*” – umożliwiające dodawanie zdjęć do formularza poprzez przeciągnięcie pliku o formacie .jpg, lub .png i upuszczenie go w tej sekcji,
- „Wyświetl dodatkowe szczegóły”, podsekcja „Wyposażenie” – dynamicznie uzupełniana sekcja, która pozwala na wybranie wyposażenia, które ma sprzedawany pojazd.



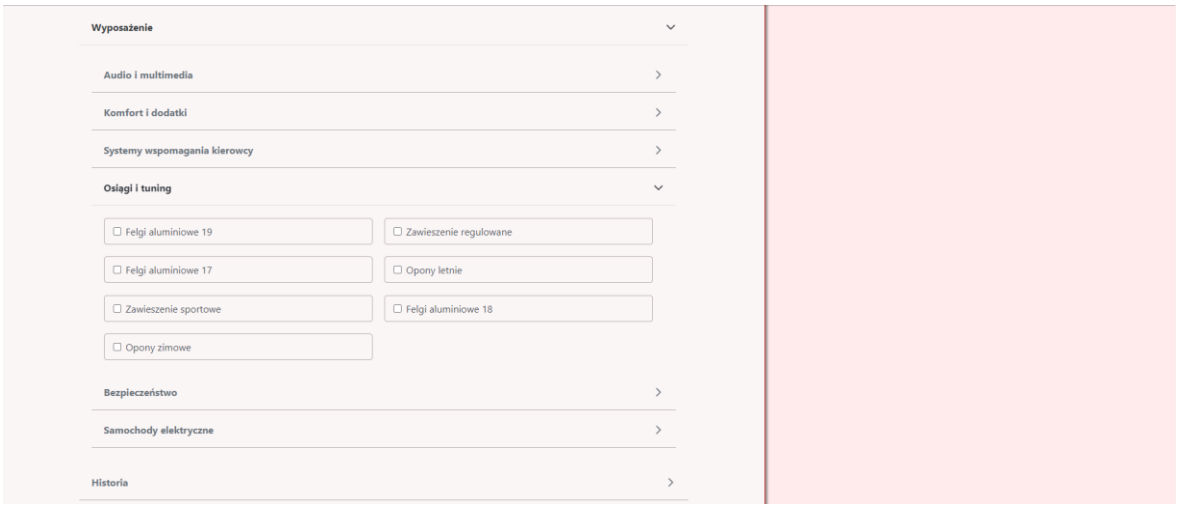
Rysunek 3.4: Ekran tworzenia ogłoszenia cz.1



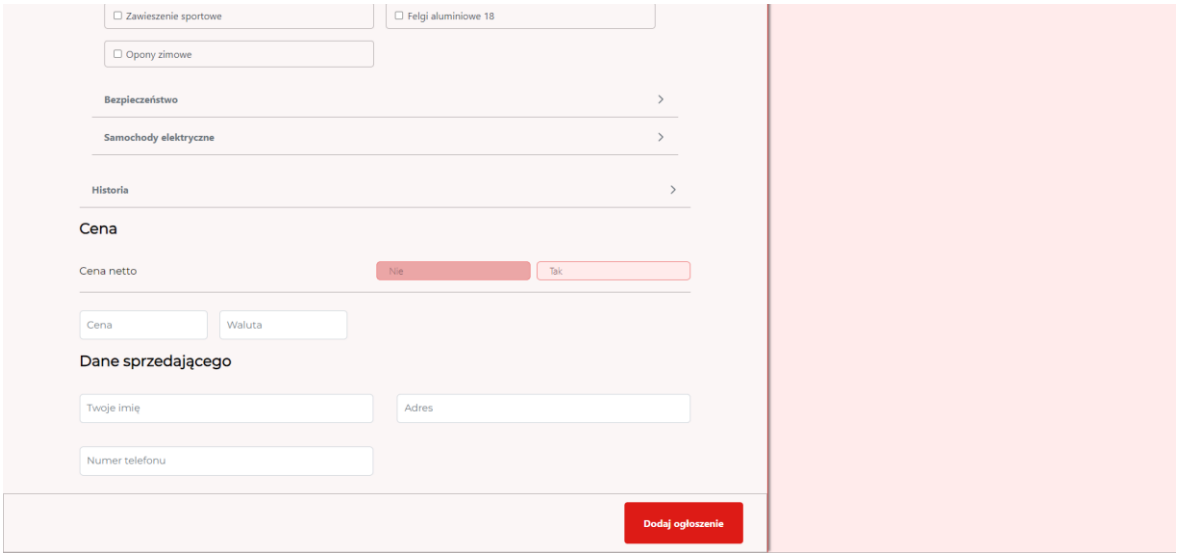
Rysunek 3.5: Ekran tworzenia ogłoszenia cz.2



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

Wybór technologii odpowiednich do potrzeb projektu informatycznego jest procesem złożonym i zależnym od wielu czynników, takich jak użyteczność czy znajomość rozwiązań. Pod pojęciem technologii kryje się szeroki zakres dostępnych metod i narzędzi. Zaliczają się do nich języki programowania, biblioteki, frameworki oraz narzędzia skoncentrowane na obsłudze baz danych i służące do testowania zaimplementowanych rozwiązań w obszarze funkcjonowania aplikacji internetowej. Z perspektywy budowania programu, technologie pełnią istotną rolę w kwestii usprawnienia przepływu informacji i udoskonalenia przebiegających procesów. Ponadto umożliwiają użytkownikom uzyskanie łatwego dostępu do danych odpowiedniego typu, co bezpośrednio wpływa na relacje z klientem oraz ogólny odbiór aplikacji przez użytkownika. Warto zaznaczyć, że technologie funkcjonujące w branży IT są znacznym udogodnieniem dla 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 mocno zależy na komponentach w formie 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]

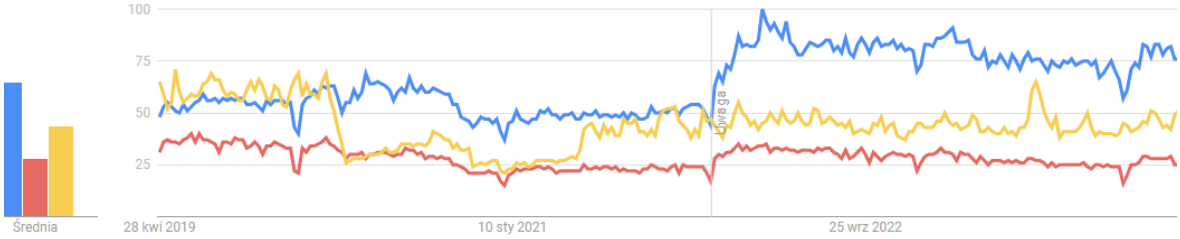
Typescript jest nakładką na język programowania *Javascript*. Powoduje to, że każdy poprawny program Javascript będzie też poprawny w Typescript (przynajmniej w większości przypadków). Dostarcza on mechanizmy strukturyzujące kod. Dodaje obiekty oparte o klasy, interfejsy i moduły. Te cechy pozwolą na 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 jest językiem programowania sieci. Przeważająca część stron internetowych używa tego języka, a każda z współcześnie używanych przeglądarek jest wyposażona w interpreter *Javascriptu*, czyniąc go najczęściej wdrażanym językiem programowania w historii. W ciągu ostatniej dekady Node.js przyczynił się do używania *Javascriptu* poza przeglądarkami. Jego ogromny sukces oznaczał, że *Javascript* jest najczęściej używanym językiem programowania wśród programistów. *Javascript* jest dynamicznym, interpretowanym językiem wysokiego poziomu, który idealnie wpasowuje się w obiektowe i funkcjonalne style programowania. Jego składnia opiera się na *Javie*, lecz

w przeciwieństwie do niej, *Javascript* jest nietypowany. Języki te nie mają ze sobą nic więcej 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)

REACTJS

React jest popularną biblioteką używaną do tworzenia interfejsu graficznego użytkownika. Stworzyła go firma *Facebook*, która borykała się z problemami związanymi z wysoko skalowalnymi aplikacjami opierających się na dużej ilości danych. *React* został wydany w 2013 roku.

Biblioteka nie dostarcza ze sobą dużej ilości narzędzi, których można by się spodziewać po tradycyjnym frameworku języka *Javascript*. Oznacza to, że deweloper ma dowolność w sprawie importowania dodatkowych narzędzi i bibliotek. [8]

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]

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]

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 naszego 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*.

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]

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.

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. Dane zapisuje i odczytuje się elektronicznie. Design bazy danych obejmuje techniki formalne i względy praktyczne, wliczając w to modelowanie danych, wydajne reprezentowanie danych i ich przechowywanie, języki *query*, bezpieczeństwo i prywatność poufnych danych.

POSTGRESQL

Istotą każdej aplikacji, która nawiązuje interakcje z użytkownikiem jest pokazywanie, pobieranie, usuwanie i aktualizowanie odpowiednich informacji wyświetlanych na UI. Nie byłoby to możliwe bez narzędzia przechowującego dużej ilości danych.

PostgreSQL to system zarządzania relacyjnej bazy danych (ang. *object-relational database management system*) ORDBMS oparta na POSTGRES, opracowana na Uniwersytecie Kalifornijskim w Berkeley na wydziale informatyki.

PostgreSQL to narzędzie o otwartym kodzie źródłowym. Wspiera dużą część standardów SQL oferując przy tym wiele nowych opcji, tj.:

- rozbudowane zapytania *query*,
- klucze obce,
- wyzwalacze,
- widoki, które można zaktualizować.

Na dodatek, *PostgreSQL* może być rozszerzony przez użytkownika o nowe:

- typy danych,
- funkcje,
- operatory,
- metody indeksowe,
- języki proceduralne. [12]

REDIS

Bazy danych *no-sql* są idealnym rozwiązaniem problemu cachowania danych w aplikacji. W przypadku tego projektu zapamiętywane 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 wytężania umysłu 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.

DOCKER

Nie każdy użytkownik aplikacji będzie miał zainstalowane narzędzia *PostgreSQL* i *PgAdmin* (graficzne narzędzie zarządzania serwami z bazami danych *PostgreSQL*). By aplikacja mogła zawsze połączyć się z bazą danych użyto narzędzia *Docker*.

Docker to technologia wirtualizacji kontenerów, innymi słowy jest to maszyna wirtualna zajmująca bardzo mało przestrzeni dyskowej. *Docker* powstał w celu rozwiązania problemu, którym jest ogromna ilość pamięci zajętej przez zwykłą maszynę wirtualną. Maszyna wirtualna obejmuje zwykle system operacyjny, na którym można znaleźć aplikację. Powoduje to kilka problemów dla szybkości i wydajności. *Docker* ma na celu wyprodukowanie środowiska, które będzie lekkie oraz bardziej elastyczne. Kontener *Dockerowy* uruchamia się szybko i możliwe jest uruchomienie wielu kontenerów jednocześnie. Powoduje to dużą skalowalność tego narzędzia.

Jeden z przypadków, dla którego deweloper będzie chciał użyć *Dockera* to ciągła integracja i ciągłe wdrażanie. Będąc tak lekkim zasobem, programiści mogą zbudować wiele kontenerów na ich maszynach, które będą kopią niektórych środowisk produkcyjnych. [14]

DOCKER COMPOSE

Docker Compose to narzędzie definiowania i uruchamiania aplikacji wielo-kontenerowych. 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 wielo-kontenerowych 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 ostatnimi etapami projektu informatycznego. Implementacja polega na przeniesieniu założeń projektowych i dokumentacji do kodu, który swoim prawidłowym działaniem zwieńczy zakończenie pracy nad projektem.

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* zostały napisane 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 zarządzaniem 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 PODZIAŁ NA PAKIETY I STRONY

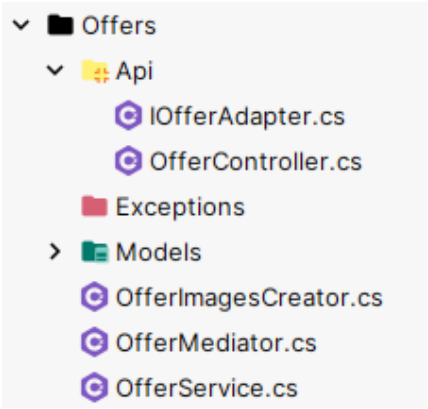
BACKEND

Struktura plików serwera została podzielona na foldery:

- *Properties* – pakiet przechowujące konfiguracje 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 5.1) można zauważyć, że oprócz tej klasy, jest w nim również klasa *Service* oraz *ImageCreator*.



Rysunek 5.1: Pakiet Offers

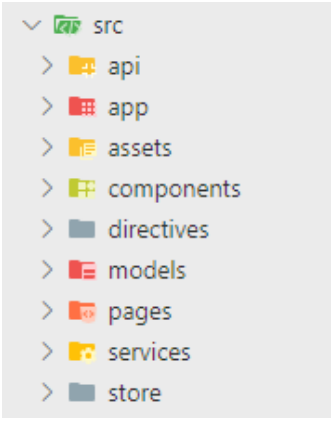
W folderze *Api* znajduje się interfejs *Adapter* definiujący metody klasy *Mediator* oraz klasa *Controller*, która wystawia endpointy, z którymi komunikuje się aplikacja webowa.

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 foldery: 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 roszerzeniem .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 5.2: 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 ustawiane 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* – by dane zawarte w kontenerze były trwałe należy użyć *volume*, które jest w tworzone i zarządzane przez *dockera*, każdy nowy *volume* zapisywany jest na dysku hosta, co oznacza, że w przypadku zniszczenia kontenera, nasze dane nie zostaną naruszone.
- *ports* – pierwszy adres oznacza, na którym porcie dostępny będzie baza danych na maszynie lokalnej. Drugi oznacza port po stronie kontenera,
- *networks* – odwołuje się do sieci stworzonej w sekcji *networks*,
- *restart*: określa sposób uruchomienia kontenera.

Poniżej przedstawiona została zawartość pliku *docker-compose.yaml* definująca 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 komendą *psql*.

5.4 KONTENERYZACJA SERWERA

Podobnie jak bazy danych konteneryzacja serwera wymagała zdefiniowania kontener 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 tego pliku 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: Dockerfile aplikacji serwerowej.

W pliku *docker-compose.yaml* dodana została sekcja *server* (Listing 5.5) podobnie jak bazy danych konteneryzacja serwera wymagała zdefiniowania jego kontenera. Obraz pobrany został z repozytorium na *dockerhub'ie*. Jednym z zadań serwera jest zapisywanie zdjęć nowych ofert, ponieważ powinny zostać one zapisane na dysku maszyny, a nie kontenerze, został utworzony 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 5.1), 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 nieistotnych 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 operacje, 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*.

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.

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

```
[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* dziedzicząca 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*.

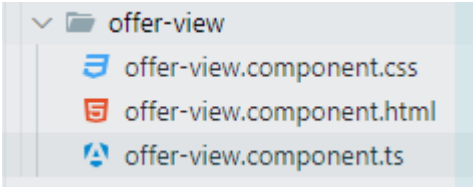
```
"ConnectionStrings":{
    "Connection":
"Host=postgres;Port=5432;Database=oto_auto;Username=dominik;Password=12345;
"
}
```

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 5.2. 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 5.3). Jeden z nich definiuje logikę, drugi szablon, a trzeci styl komponentu.



Rysunek 5.3: 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, i „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 Listing 5.12 i Listing 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 wykonywana 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 to *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 Listing 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 przechowujące 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 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]

6.1 WYDAJNOŚĆ

Aplikacje testowane 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* umożliwiającą 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*.

6.1.1 WYDAJNOŚĆ APLIKACJI OTOAUTO

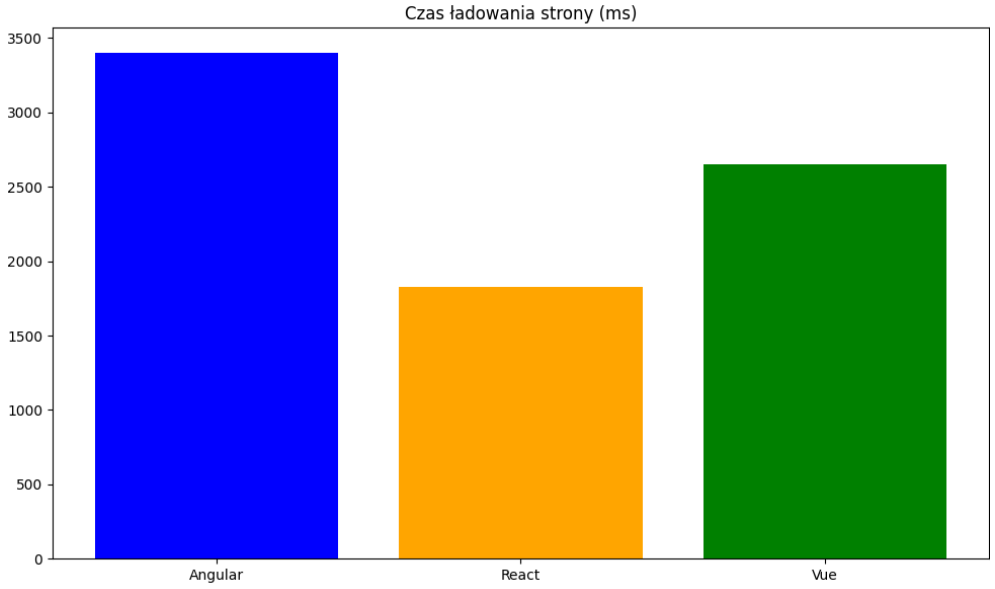
Wydajność aplikacji została przetestowana tylko na stronie głównej aplikacji. Uwzględnione zostało 9 parametrów, które zostaną omówione w dalszych podrozdziałach.

Tabela 6.1: 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

6.1.1.1 Czas ładowania strony

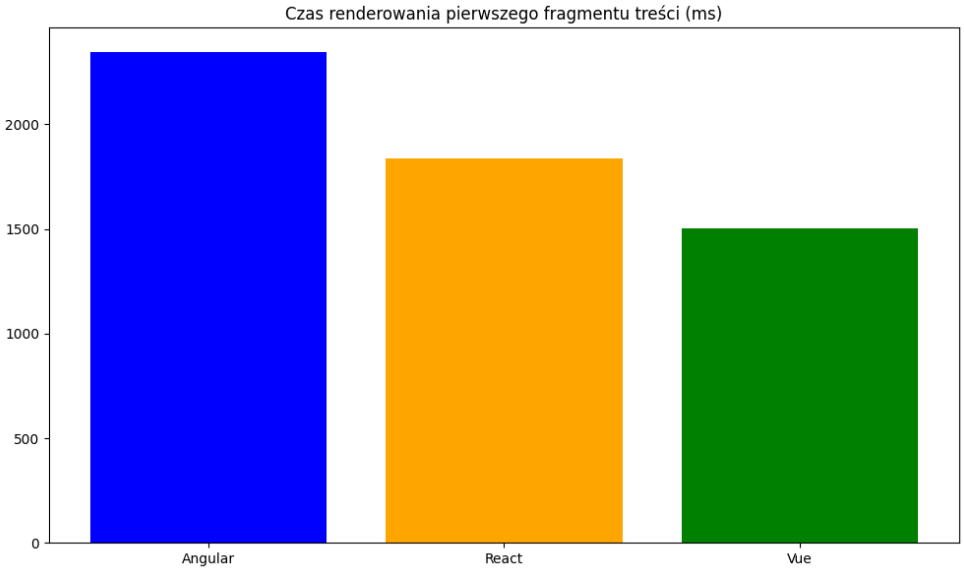
Ten parametr oznacza czas jaki strona potrzebuje by w pełni załadować stronę od rozpoczęcia ładowania. Jest to istotny aspekt wydajności, ponieważ długi czas ładowania może zniechęcić użytkowników z korzystania z aplikacji. W przypadku omawianej aplikacji najkrótszy czas ładowania ma *React*, a najdłuższy *Angular*.



Rysunek 6.1: Wyniki czasu ładowania strony otoauto

6.1.1.2 Czas renderowania pierwszego fragmentu treści

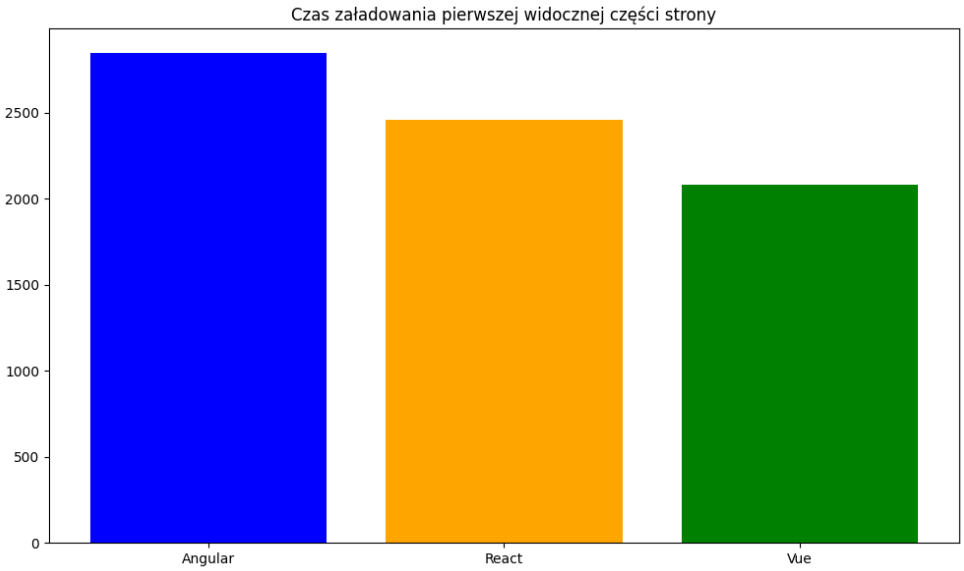
Parametr ten przedstawia czas, w którym przeglądarka renderuje pierwszy fragment treści strony. Innymi słowy jest to czas po jakim użytkownik po raz pierwszy widzi coś na stronie. Z wszystkich porównywanych narzędzi *Vue* osiągnął najkrótszy czas, a *Angular* najdłuższy.



Rysunek 6.2: Wynik czasu renderowania pierszego fragmentu treści strony otoauto

6.1.1.3 Czas załadowania pierwszej widocznej części strony

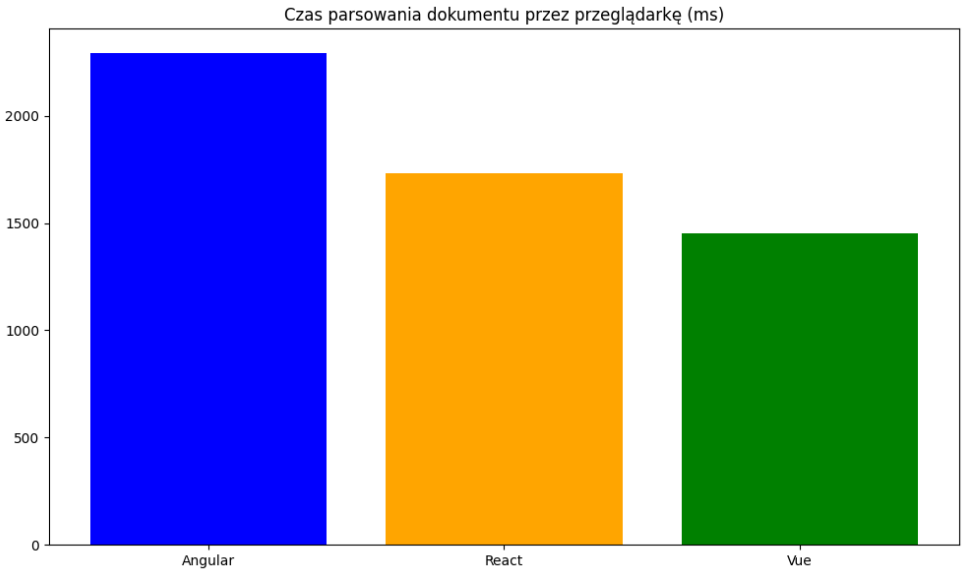
Ten parametr przedstawia czas, jest potrzebny przeglądarce do wyświetlenia zawartości strony. Mały czas tego parametru jest istotny by użytkownik mógł jak najszybciej korzystać z tego co znajduje się na stronie. Ponownie *Vue* ma najlepszy czas, a Angular najdłuższy.



Rysunek 6.3: Wynik czasu załadowania pierwszej widocznej części strony otoauto

6.1.1.4 Czas parsowania dokumentu przez przeglądarkę

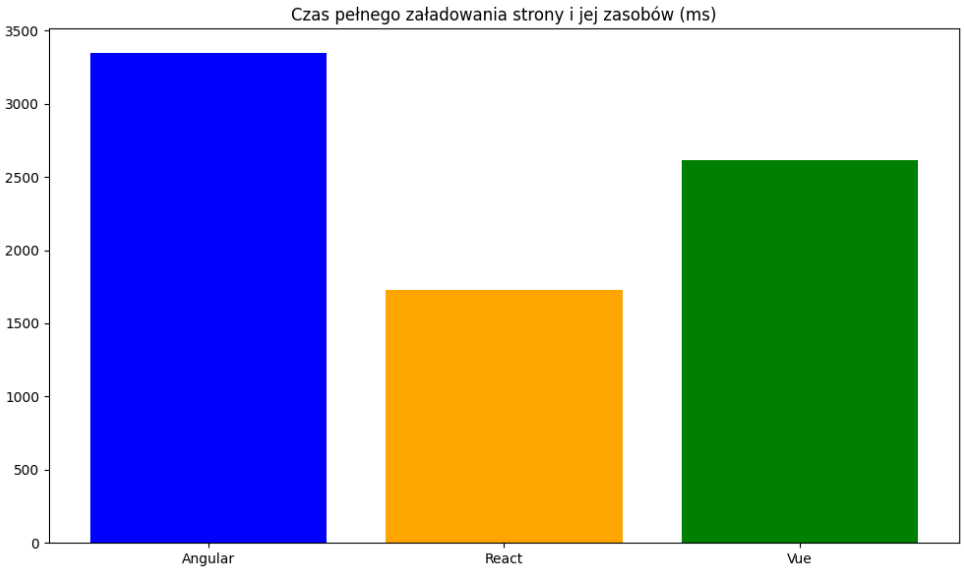
Parametr oznaczający czas potrzebny do załadowania wszystkich zasobów, np. skryptów. Szybkie parsowanie pozwala przeglądarce na szybsze wykonywanie skryptów, a co za tym idzie, poprawienie interaktywności strony. Czas parsowania jest najkrótszy dla *vue*, a najdłuższy dla Angulara.



Rysunek 6.4: Wynik czasu parsowania dokumentu przez przeglądarkę – otoauto

6.1.1.5 Czas pełnego załadowania strony i jej zasobów

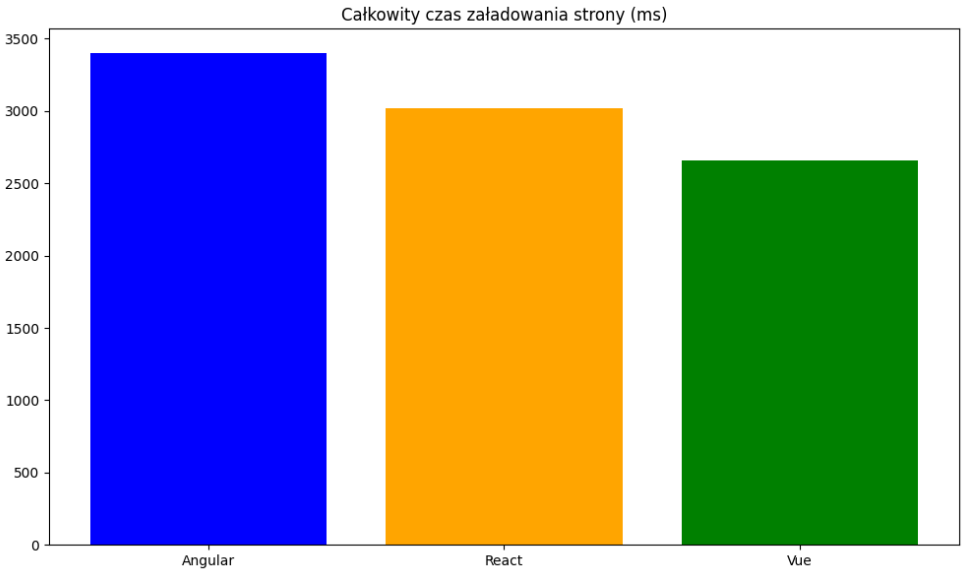
To parametr wskazujący na czas, po którym strona została załadowana wraz z obrazami, stylami lub skryptami. Szybkie załadowanie całości strony jest kluczowe dla użytkownika, ponieważ pozwala ono na dostęp do wszystkich funkcji strony. Dla omawianej aplikacji całą zawartość najszybciej ładuje *React*, a najwolniej *Angular*, którego czas jest prawie dwukrotnie większy od czasu *Reacta*.



Rysunek 6.5: Wynik czasu pełnego załadowana strony oraz jej zasobów – otoauto

6.1.1.6 Całkowity czas załadowania strony

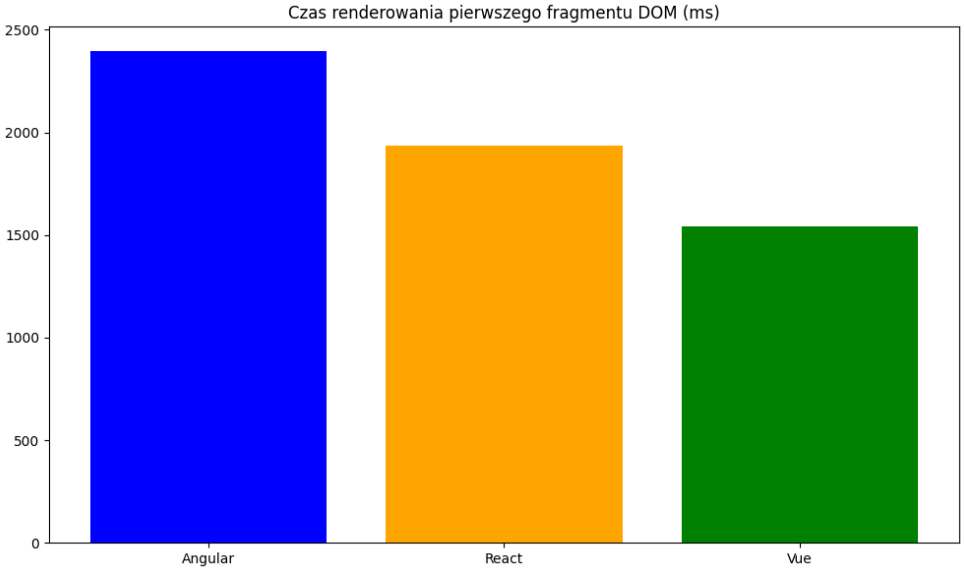
Jest to parametr, który oprócz czasu pełnego załadowania strony i jej zasobów, bierze też pod uwagę czas do momentu załadowania zasobów pobieranych asynchronicznie. Metryka ta pomaga ocenić obciążenie strony. Badanie czasu wykazało, że *Vue* poradził sobie z załadowaniem strony najlepiej.



Rysunek 6.6: Wynik całkowitego czasu załadowania strony - otoauto

6.1.1.7 Czas renderowania pierwszego fragmentu DOM

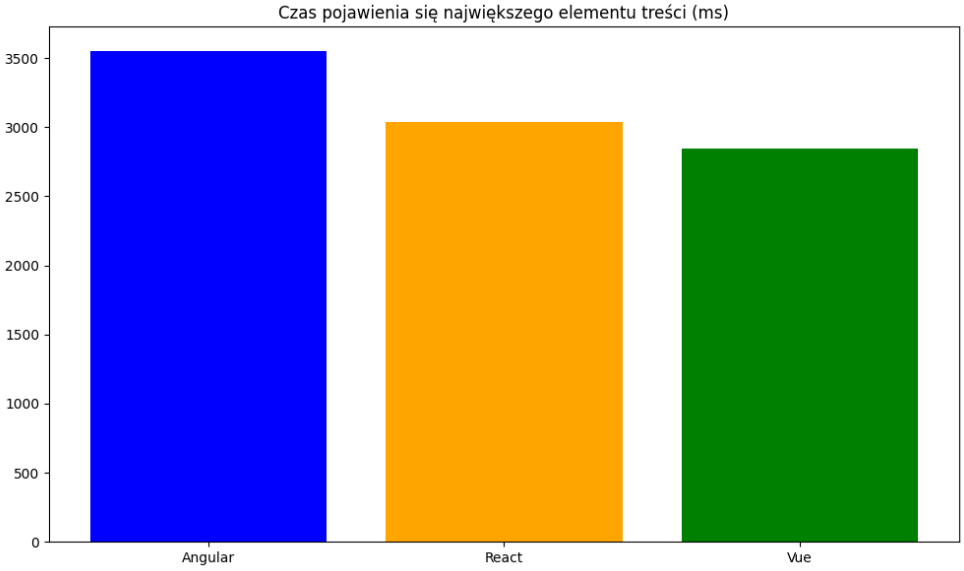
Parametr ten wskazuje czas, w którym przeglądarka renderuje pierwszy fragment treści DOM (tekst, obraz lub grafika SVG). Wpływa on na postrzeganie szybkości ładowania przez użytkownika i jeśli jest on niski to może znacząco poprawić pierwsze wrażenie z korzystania z strony. Z załadowaniem pierwszego fragmentu DOM najszybciej poradził sobie *Vue*.



Rysunek 6.7: Wynik czasu renderowania pierwszego fragmentu DOM - otoauto

6.1.1.8 Czas pojawienia się największego elementu treści

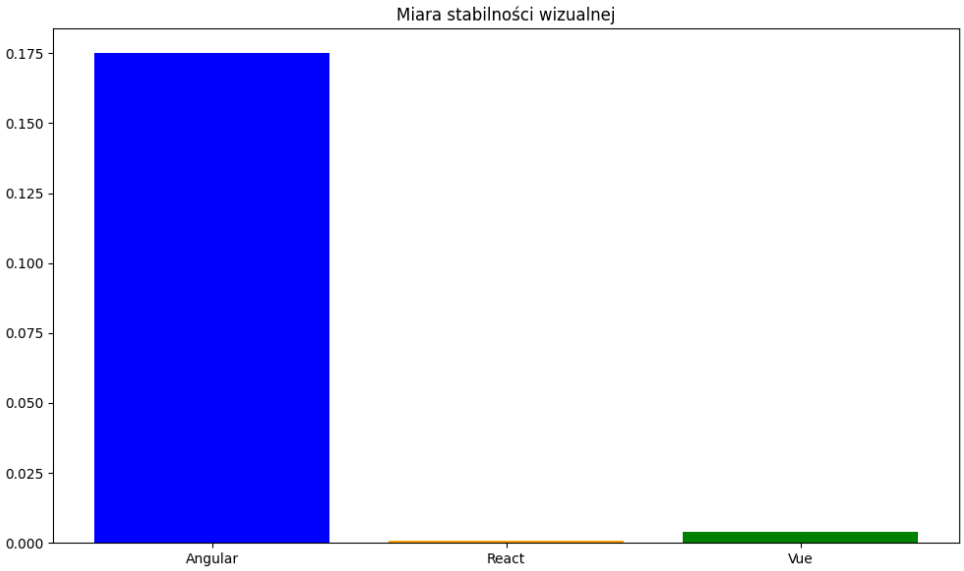
Jest to parametr określający czas, w którym największy element na stronie staje się widoczny. Często największy element jest najważniejszy. Z tego powodu jego szybkie załadowanie może mieć kluczową rolę na wrażenia użytkownika. Najlepiej w tym aspekcie poradził sobie *Vue*, a najgorzej ponownie *Angular*.



Rysunek 6.8: Wynik czasu pojawienia się największego elementu treści – otoauto

6.1.1.9 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 6.9: Wynik miary stabilności wizualnej - otoauto

6.1.2 TEST PUSTEJ APLIKACJI

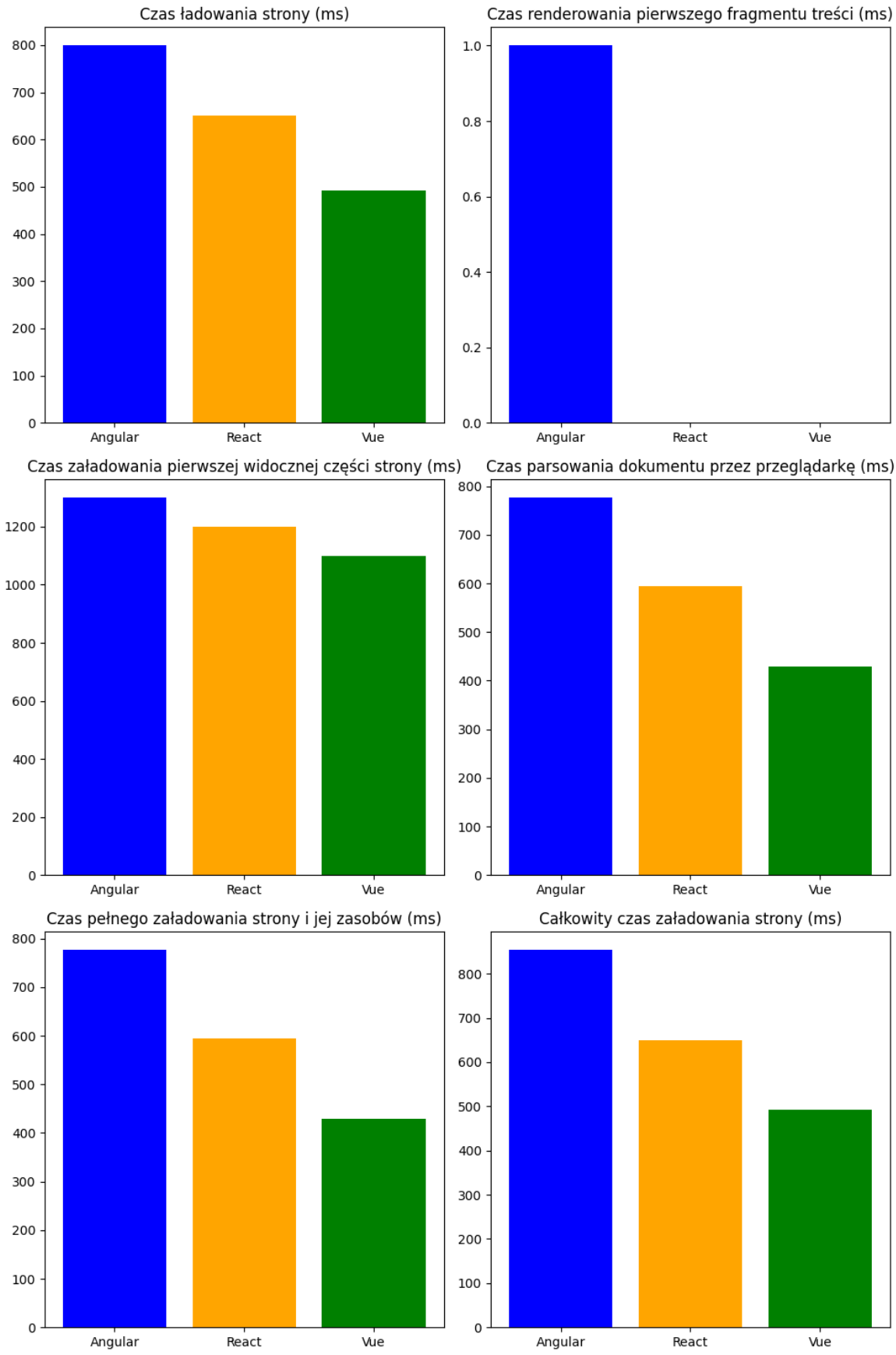
Sprawdzenie wydajności pustej aplikacji ma na celu sprawdzenie podstawowej wydajności danego frameworka. Narzędzia te mają różne architektury, a to może mieć znaczenie na czas ładowania i wydajność. Jest to również idealny sposób na sprawdzenie jak szybko działa każdy z frameworków, ponieważ nie występują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

Tabela 6.2: 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
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 był około 32% szybszy od *Angulara* oraz 19% 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 6.10: Wyniki testów wydajności pustej aplikacji

6.1.3 TEST WYPEŁNIONEJ APLIKACJI

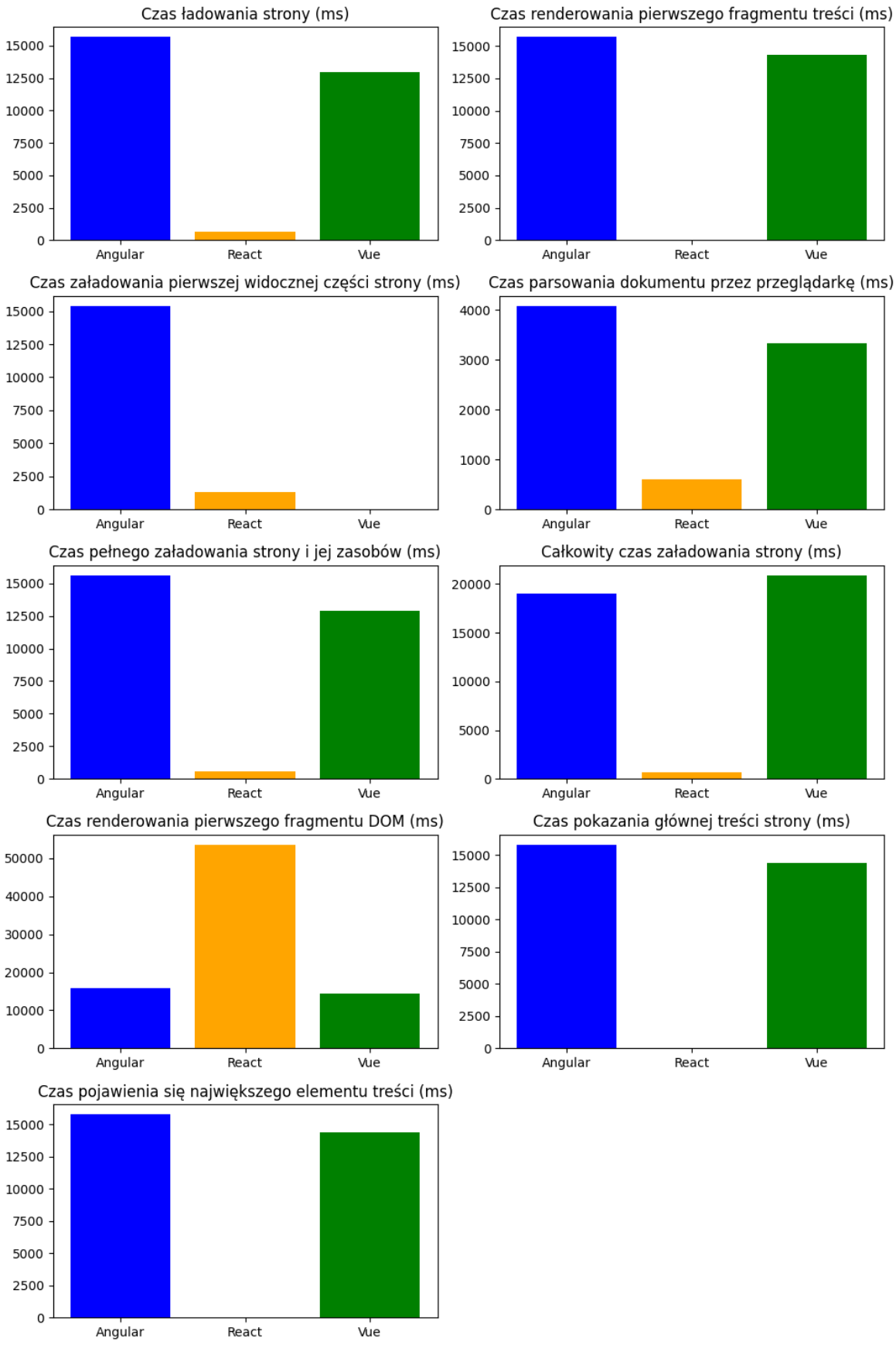
Testy wydajności strony wypełnionej dużą ilością treści (w tym przypadku 100 000 linijek tekstu i pól tekstowych) pozwalają sprawdzić jak dobrze dany framework radzi sobie z renderowaniem i jak wpływa to na doświadczenie z korzystania strony przez użytkownika. Czas renderowania ma duże znaczenie dla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

Tabela 6.3: 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
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 *React*a 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 6.11: Wyniki testów wypełnionej aplikacji

6.2 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. Wydajność będzie lepsza 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 6.4: 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.

6.3 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 dockerowym 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 --force

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 6.5: 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.

6.4 IMPLEMENTACJA DOM

DOM (ang. „*Document Object Model*") dostarcza ustrukturyzowaną reprezentacje 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."* 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 6.6).

Tabela 6.6: 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

6.5 POPULARNOŚĆ I WSPARCIE SPOŁECZNOŚCI

Artykuł *"Comparison: Angular Vs. React Vs. Vue. Which Framework Is The Best Choice."*CINCOVIĆ, JELICA, AND MARIJA PUNT. "COMPARISON: ANGULAR VS. REACT VS. VUE. WHICH FRAMEWORK IS THE BEST CHOICE." UNIVERSIDAD DE BELGRADE (2020). 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 wyszukiwani 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 się 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 6.7: 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

6.6 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 sanizacja 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. Sanizacja 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.

6.7 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 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”* 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 testów oraz artykułu są zgodne. W obu przypadkach Vue wyróżnia się najlepszą wydajnością, a Angular największym rozmiarem paczek. Dodatkowo można zauważyć, że 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.

7 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 kontrolowania systemów, 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.

7.1 ZMIENNE LINGWISTYCZNE

Do zbudowania modelu potrzebne są zmienne lingwistyczne, które zostaną użyte do zdefiniowania reguł. Będą się one odnosić do aspektów porównanych w rozdziale 6. 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 – przyjmujący 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),
    - krzywa uczenia - przyjmująca wartości:
        - „łatwa” – (0, 0, 10, 30),
        - „umiarkowana” – (10, 30, 60, 80),
        - „trudna” – (60, 80, 100, 100),
    - wbudowane bezpieczeństwo - przyjmujące wartość:
        - „brak” - (0, 0, 5),
        - „wbudowane” - (5, 10, 10),
    - popularność – przyjmująca wartości:
        - „malejąca” – (0, 0, 20, 40),
        - „stagnacja” – (20, 40, 70, 90),
        - „rosnąca” – (70, 90, 100, 100),
    - elastyczność – przyjmująca wartości:
        - „sztywna” – (0, 0, 20, 40),
        - „umiarkowana” – (20, 40, 70 90),
        - „elastyczna” – (70, 90, 100, 100).

7.2 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.

7.2.1 ANGULAR

7.2.2 REACT

7.2.3 VUE

8 PODSUMOWANIE

8.1 WNIOSKI

BIBLIOGRAFIA

[1] TOBIAŃSKA, MONIKA, AND JAKUB SMOŁKA. "EFEKTYWNOŚĆ TWORZENIA WARSTWY PREZENTACJI APLIKACJI WE FRAMEWORKACH ANGULARJS, ANGULAR2, 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.reactjs.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

Rysunek 1.1: Popularność siłowni w Polsce na przestrzeni lat 2017-2022 7
Rysunek 1.2. Strona główna fabrykasily.pl 9
Rysunek 2.1: Diagram związków encji 12
Rysunek 2.2: Diagram przypadków użycia 13
Rysunek 3.1: Widok strony głównej 17
Rysunek 3.2: Widok strony szczegółów oferty 18
Rysunek 3.3: Widok strony przeglądania ofert 18
Rysunek 3.4: Ekran tworzenia ogłoszenia cz.1 19
Rysunek 3.5: Ekran tworzenia ogłoszenia cz.2 19
Rysunek 3.6: Ekran tworzenia ogłoszenia cz.3 20
Rysunek 3.7: Ekran tworzenia ogłoszenia cz.4 20
Rysunek 3.8: Ekran tworzenia ogłoszenia cz.5 20
Rysunek 4.1: Porównanie zainteresowania Angularem (linia czerwona), Reactem (linia niebieska) oraz Vue(linia żółta) 22
Rysunek 5.1: Pakiet Offers 27
Rysunek 5.2: Podział frontendu na foldery 28
Rysunek 5.3: Komponent offer-view (Angular) 35
Rysunek 6.1: Wyniki czasu ładowania strony otoauto 43
Rysunek 6.2: Wynik czasu renderowania pierszego fragmentu treści strony otoauto 43
Rysunek 6.3: Wynik czasu załadowania pierwszej widocznej części strony otoauto 44
Rysunek 6.4: Wynik czasu parsowania dokumentu przez przeglądarkę – otoauto 44
Rysunek 6.5: Wynik czasu pełnego załadowana strony oraz jej zasobów – otoauto 45
Rysunek 6.6: Wynik całkowitego czasu załadowania strony - otoauto 45
Rysunek 6.7: Wynik czasu renderowania pierwszego fragmentu DOM - otoauto 46
Rysunek 6.8: Wynik czasu pojawienia się największego elementu treści – otoauto 46
Rysunek 6.9: Wynik miary stabilności wizualnej - otoauto 47
Rysunek 6.10: Wyniki testów wydajności pustej aplikacji 49
Rysunek 6.11: Wyniki testów wypełnionej aplikacji 51

SPIS LISTINGÓW

Listing 5.1: Definicja kontenerów baz danych 29
Listing 5.2: Fragment skryptu tworzącego tabele. 29
Listing 5.3: Skrypt wykonujący komendy SQL w plikach znajdujących się w folderze SQL_Scripts. 30
Listing 5.4: Dockerfile aplikacji serwerowej. 31
Listing 5.5: Definicja kontenera serwera 31
Listing 5.6: Fragment klasy OfferController 32
Listing 5.7 Klasa Offer 33
Listing 5.8: Metoda z repozytorium oferty pobierająca ofertę z pojazdem po id 33
Listing 5.9: Fragment kodu pliku Program.cs 34
Listing 5.10: Zawartość pliku OtoAutoContext 34
Listing 5.11: Definicja połączenia z bazą danych w pliku appsettings.json 35
Listing 5.12: Komponent OfferView (Angular) 35
Listing 5.13: Klasa GetOfferById zdefiniowana w pliku src/store/actions/offer-actions.ts 36
Listing 5.14: Akcja GetOfferById zdefiniowana w klasie OfferState 37
Listing 5.15: Komponent OfferView (React) 38
Listing 5.16: Wywołanie metody createAsyncThunk 39
Listing 5.17: Obiekt przekazany do metody createSlice 40
Listing 5.18 Fragment modułu przechowującego dane ofert 40

SPIS TABEL

Tabela 6.1: Wynik testu strony głównej aplikacji otoauto 42
Tabela 6.2: Wynik testu wydajności pustej strony 47
Tabela 6.3: Wynik testu wypełnionej aplikacji..... 50
Tabela 6.4: Wynik pomiaru rozmiaru paczek produkcyjnych..... 52
Tabela 6.5: Wynik analizy Sonarqube 53
Tabela 6.6: Wyniki testów wydajności operacji manipulujących DOM 54
Tabela 6.7: Porównanie frameworków pod kątem popularności, wsparcia społeczności, krzywej uczenia się i elastyczności 54