**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**

[Streszczenie 2](#_Toc165673759)

[Abstract 2](#_Toc165673760)

[Wstęp 5](#_Toc165673761)

[Cel i zakres pracy 5](#_Toc165673762)

[1 Wprowadzenie 7](#_Toc165673763)

[1.1 Przegląd aktualnych rozwiązań 9](#_Toc165673764)

[2 Faza projektowa 10](#_Toc165673765)

[2.1 Wymagania funkcjonalne 10](#_Toc165673766)

[2.2 Wymagania niefunkcjonalne 10](#_Toc165673767)

[2.3 Diagram ER 10](#_Toc165673768)

[2.4 Diagram przypadków użycia 13](#_Toc165673769)

[2.5 Opis i definicja scenariuszy przypadków użycia 14](#_Toc165673770)

[3 Interfejs użytkownika 17](#_Toc165673771)

[Widok strony głównej 17](#_Toc165673772)

[Widok szczegółów oferty 17](#_Toc165673773)

[Widok przeglądarki ofert 18](#_Toc165673774)

[Widoki ekranu tworzenia ogłoszenia 18](#_Toc165673775)

[4 Wykorzystane technologie 21](#_Toc165673776)

[4.1 Wybrane języki 21](#_Toc165673777)

[4.2 Narzędzia strony interfejsu użytkownika 22](#_Toc165673778)

[ReactJS 22](#_Toc165673779)

[Angular 22](#_Toc165673780)

[Vue 22](#_Toc165673781)

[Biblioteki do budowania komponentów 23](#_Toc165673782)

[4.3 Narzędzia strony serwerowej 23](#_Toc165673783)

[ASP.NET Core 23](#_Toc165673784)

[Model-View-Controller (MVC) 23](#_Toc165673785)

[Web API 23](#_Toc165673786)

[4.4 Narzędzia bazy danych 24](#_Toc165673787)

[PostgreSQL 24](#_Toc165673788)

[Redis 24](#_Toc165673789)

[4.5 Narzędzia Konteneryzacji 24](#_Toc165673790)

[Docker 25](#_Toc165673791)

[Docker Compose 25](#_Toc165673792)

[5 Implementacja aplikacji 26](#_Toc165673793)

[5.1 Środowiska programistyczne 26](#_Toc165673794)

[5.2 Podział na pakiety i strony 26](#_Toc165673795)

[Backend 26](#_Toc165673796)

[Frontend 27](#_Toc165673797)

[5.3 Implementacja bazy danych 28](#_Toc165673798)

[5.4 Implementacja serwera 29](#_Toc165673799)

[5.5 Implementacja interfejsu użytkownika 35](#_Toc165673800)

[5.6 Problemy implementacyjne 38](#_Toc165673801)

[Błąd CORS przy wysyłaniu żądań 38](#_Toc165673802)

[Implementacja pobierania ćwiczeń do generowania planu treningowego 38](#_Toc165673803)

[Komunikacja z użyciem obiektów enum 38](#_Toc165673804)

[6 Testowanie aplikacji 40](#_Toc165673805)

[6.1 Przeprowadzone testy 40](#_Toc165673806)

[6.2 Wyniki testów manualnych 41](#_Toc165673807)

[7 Podsumowanie 43](#_Toc165673808)

[7.1 Wnioski 43](#_Toc165673809)

[7.2 Koncepcja rozwoju aplikacji 43](#_Toc165673810)

[Bibliografia 45](#_Toc165673811)

[Spis rysunków 46](#_Toc165673812)

[Spis listingów 47](#_Toc165673813)

# Wstęp

Podczas pracy jako programista aplikacji internetowych w frameworku[[1]](#footnote-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,
* Kompatybilności i wsparcia (współpraca z innymi narzędziami / bibliotekami),
* Dokumentacji – kompletność i użyteczność dokumentacji,
* Skalowalności,
* Testowania,
* Popularności,
* 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.

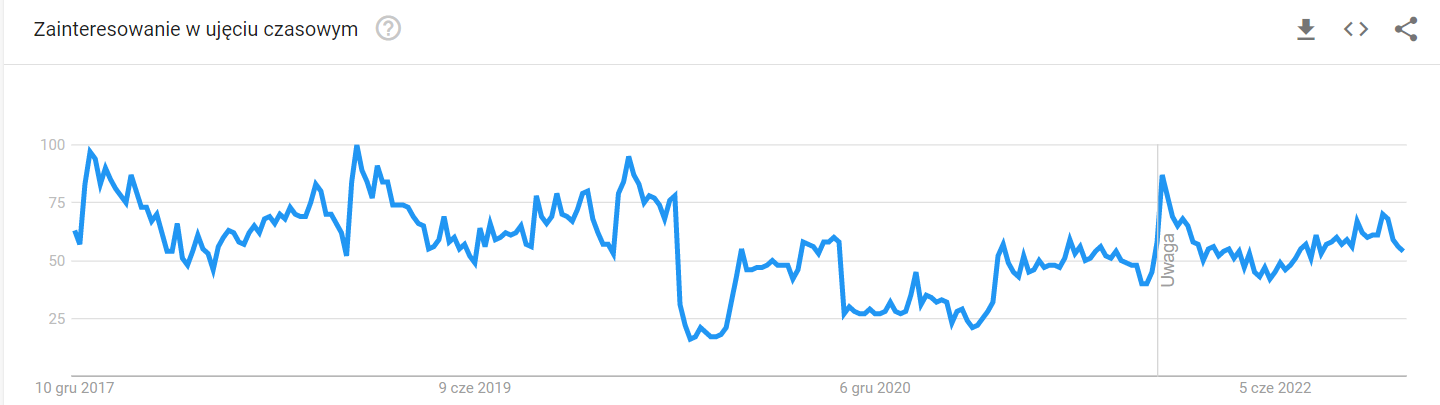
Po zbudowaniu aplikacje zostaną porównane pod względem składni i struktury za pomocą metryk Chidambera Kemerera. Wydajność zostanie przetestowana narzędziami Google Lighthouse, WebPageTest, React Developer Tools, Vue Devtools i Angular Augury. Porównaniu ulegną również rozmiary paczek dla pustego, skończonego projektu. Rozmiar zmieni się też dla paczki produkcyjnej. Kompatybilność i wsparcie obejmuje takie aspekty jak: język, którego używa dany framework, biblioteki, z których dany framework może skorzystać, czy narzędzia deweloperskie ułatwiające prace z tymi narzędziami. Porównane zostaną kompletność dokumentacji, popularność i sposób implementacji DOM. Temat skalowalności przedstawi strukturę modułową oraz wstrzykiwanie zależności (ang. „*dependency injection*”) Angulara, komponenty oraz funkcje specjalne Reacta i  jedno-plikowe komponenty Vue. W aspekcie testowania zostaną porównane odpowiednie narzędzia.

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.

# 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.**

## 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.**).

# 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.

## 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.

## 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.

## 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.

Obraz zawierający zrzut ekranu, design

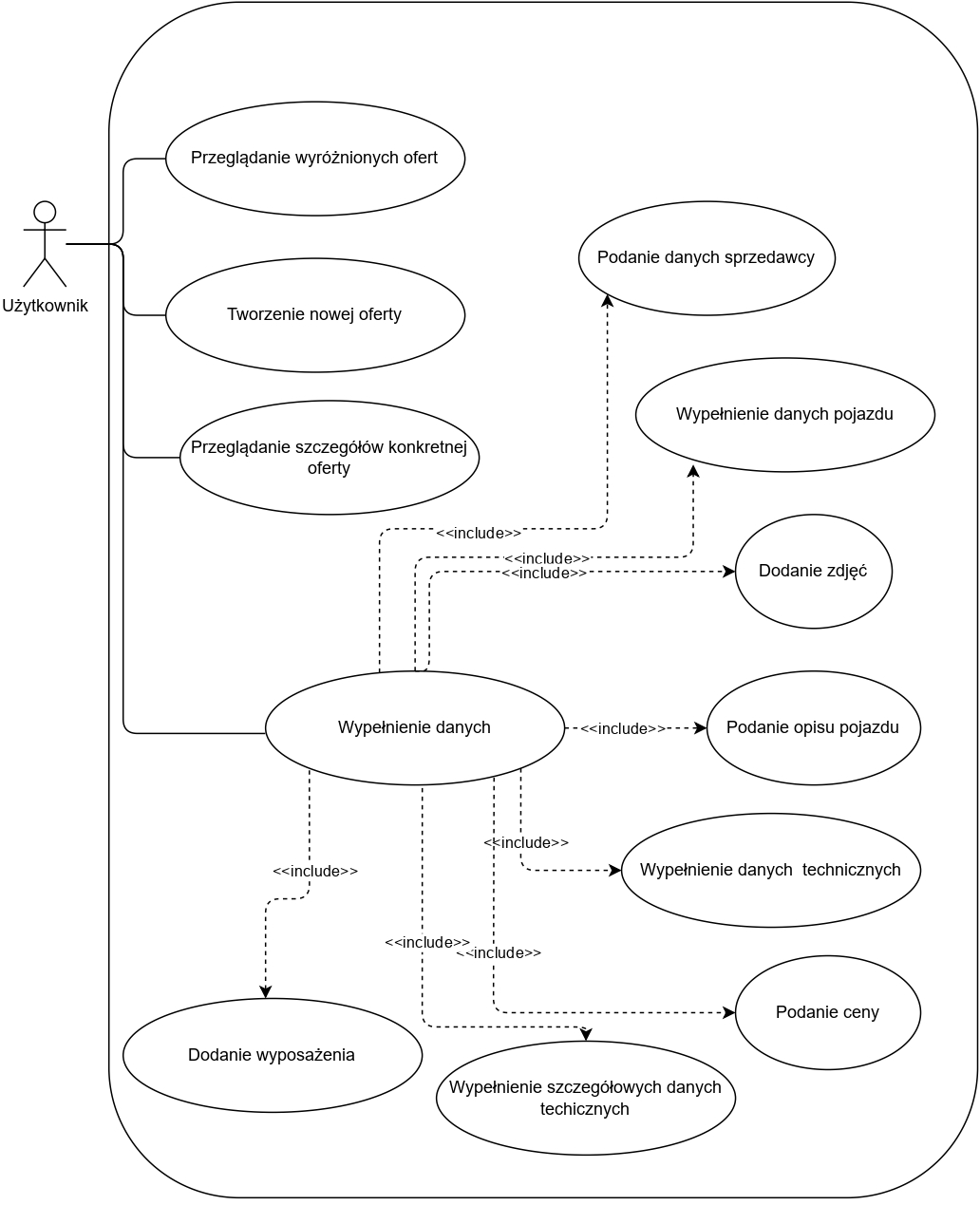
Opis wygenerowany automatycznie

Rysunek 2.1: Diagram związków encji

## 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

## 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.

# 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.

Obraz zawierający tekst, pojazd, Pojazd lądowy, Projektowanie samochodów

Opis wygenerowany automatycznie

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.

Obraz zawierający tekst, Pojazd lądowy, pojazd, samochód

Opis wygenerowany automatycznie

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).

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, Reklama internetowa

Opis wygenerowany automatycznie

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.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Rysunek 3.4: Ekran tworzenia ogłoszenia cz.1

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Rysunek 3.5: Ekran tworzenia ogłoszenia cz.2

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Rysunek 3.6: Ekran tworzenia ogłoszenia cz.3

Obraz zawierający tekst, zrzut ekranu, numer, paragon

Opis wygenerowany automatycznie

Rysunek 3.7: Ekran tworzenia ogłoszenia cz.4

Obraz zawierający tekst, zrzut ekranu, design

Opis wygenerowany automatycznie

Rysunek 3.8: Ekran tworzenia ogłoszenia cz.5

# 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.

## Wybrane języki

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

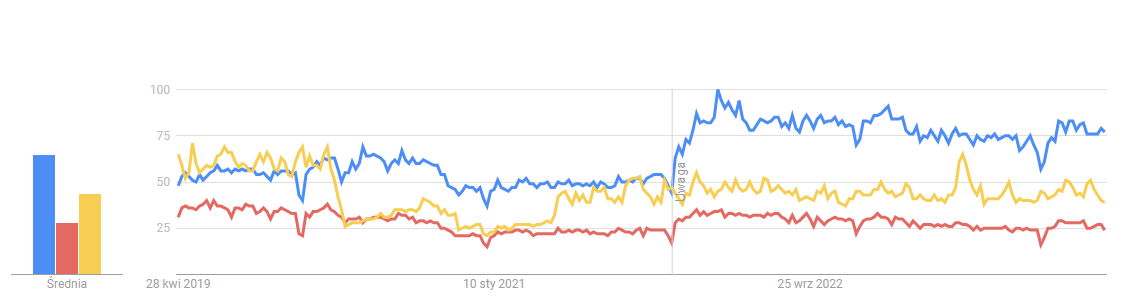
C# jest prostym, współczesnym, obiektowym językiem programowania zapewniającym bezpieczeństwo typów. Korzenie C# sięgają do rodziny języków C i z tego powodu jest podobny do języków C, C++, czy Java. Zapewnia wsparcie dla programowania opartego o komponenty. Współczesne oprogramowanie mocno zależy na 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]

## 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>>*.

## 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]

## 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]

## 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.

# 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.

## 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.

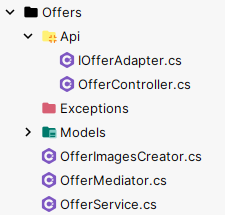
## Podział na pakiety i strony

### Backend

Struktura plików serwera została podzielona na foldery:

* *Properties* – pakiet przechowujące konfiguracje uruchomieniową,
* *DbContexts* – pakiet, w którym znajduje się klasa tworząca konteksty dla tabel wbazie 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. Klasa O*fferController* (Listing 5.1), by poprawnie działało, musi dziedziczyć po *ControllerBase* – klasie z pakietu *Microsoft.AspNetCore.Mvc*, a także posiadać adnotacje *ApiController* oraz *Route* – które ustawia ścieżkę do endpointów w tej klasie.

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

[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 5.1: Fragment klasy OfferController

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.

Folder Models zawiera klasy typu dto2, 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.

### 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.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek 5.2: Podział frontendu na foldery

## 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 5.2: 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 5.3 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 5.3: 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 5.4: Skrypt wykonujący komendy SQL w plikach znajdujących się w folderze SQL\_Scripts.

Listing 5.4 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*.

## Konteneryzacja serwera

Podobnie jak bazy danych konteneryzacja serwera wymagała zdefiniowania kontener serwera. W tym celu wymagane jest stworzenie pliku *dockerfile* (Listing 5.5), 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 5.5: Dockerfile aplikacji serwerowej.

W pliku *docker-compose.yaml* dodana została sekcja *server* (Listing 5.6) 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 5.6: Definicja kontenera serwera

## Implementacja serwera

W punkcie 5.2 został opisany sposób podziału plików w pakietach. Odnosząc się do tabeli *Exercise* (Listing 1) przedstawiony zostanie pakiet *exercise*. Klasa odpowiadająca encji (Listing 2) znajdującej się w bazie danych znajduje się w pakiecie *model.entity*. Pierwsze dwie adnotacje oznaczają, że klasa ta odnosi się do encji w bazie danych o nazwie *exercise*. Kolejne oznaczenia pochodzące z rozszerzenia *Lombok* to:

* *Getter* –generowana jest funkcja dostępu dla każdego atrybutu w klasie,
* *Setter -* generowana jest funkcja ustawiania wartości dla każdego atrybutu w klasie,
* *AllArgsConstructor* – tworzy konstruktor ze wszystkimi atrybutami w klasie,
* *NoArgsConstructor* – tworzy domyślny konstruktor klasy
* *Builder* – pomocny przy tworzeniu nowych obiektów danej klasy przy użyciu funkcji *builder()* i *build()*.

Ostatnia adnotacja zostanie opisana w punkcie 5.6.

@Entity  
@Table(name = **"exercise"**)  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@TypeDef(  
 name = **"pgsql\_enum"**,  
 typeClass = PostgreSQLEnumType.**class**)  
**public class** Exercise {  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private** Integer **id**;  
 **private** String **exerciseName**;  
 **private** String **description**;  
 **private** String **url**;  
 @Enumerated(EnumType.***STRING***)  
 @Column(columnDefinition = **"muscle\_group"**)  
 @Type(type=**"pgsql\_enum"**)  
 **private** MuscleGroup **muscleGroup**;  
 @Enumerated(EnumType.***STRING***)  
 @Column(columnDefinition = **"exercise\_type"**)  
 @Type(type=**"pgsql\_enum"**)  
 **private** ExerciseType **exerciseType**;  
 @Enumerated(EnumType.***STRING***)  
 @Column(columnDefinition = **"difficulty"**)  
 @Type(type=**"pgsql\_enum"**)  
 **private** Difficulty **difficulty**;  
 **private** Boolean **isForOldPeople**;  
 **private** Boolean **isForObesePeople**;  
 **private** Boolean **isForYoungPeople**;  
 **private** Boolean **isForWomen**;  
 @OneToMany(mappedBy = **"exercise"**)  
 **private** List<ExerciseWithData> **exercisesWithData**;  
}

Listing 5.7: Klasa Exercise

Klasa oznaczona adnotacją *@Entity* musi zawierać pole, które oznaczone jest adnotacją *@Id*, która oznaczać będzie klucz główny. Warto przy takim polu dodać jednocześnie adnotacje *@GeneratedValue*, która określi sposób generowania id. W tym przypadku podana została strategia *IDENTITY* oznaczająca, że każdy nowy obiekt tej klasy zapisany w bazie danych będzie miał id zwiększone o 1, względem poprzedniego obiektu.

Encja *Exercise* jest w relacji *jeden-do-wielu* z encją *ExerciseWithData*. By odwzorować to zachowanie po stronie serwera należało stworzyć atrybut zawierający listę obiektów tej encji. W klasie *ExerciseWithData* zdefiniowano obiekt klasy *Exercise*, który oznaczono adnotacjami:

* *@ManyToOne* – typ kaskady *Detach* oznacza, że w momencie usuwania obiektu *ExerciseWithData* obiekt *Exercise* zostanie „odłączony” i nie zostanie usunięty,
* *@JoinColumn* – oznaczające, że dane pole to klucz obcy.

W celu komunikacji się z bazą danych należało stworzyć interfejs, który *Springboot* nazywa repozytorium. Takie interfejsy oznaczone są adnotacją *@Repository* oraz rozszerzone o *JpaRepository*, której jako typy generyczne przekazujemy klasę mapującą encję w bazie danychoraz typ zmiennej, który w tej klasie oznaczony jest adnotacją *@Id.*

W tym interfejsie można tworzyć funkcje na 3 różne sposoby:

* funkcja używa słów rozpoznawalnych przez *Springboot* takich jak: *find*, *by*, *id*, *and* przez co można zbudować funkcję np. *findExerciseById*, która zgodnie z nazwą znajdzie w bazie danych obiekt *Exercise* o id podanym w parametrze funkcji,
* funkcja z dodaną adnotacją *@Query*, gdzie w adnotacji za pomocą języka *JPQL* bezpośrednio przekazuje się funkcji, jakie dane mają zostać pobrane z bazy danych,
* funkcja z dodaną adnotacją *@Query*, gdzie w adnotacji zostaje ustawiony parametr *nativeQuery*, a następnie za pomocą języka *SQL* określone zostają dane do zwrócenia.

By aplikacja mogła komunikować się z serwerem należało zdefiniować w nim klasę odpowiadającą za API (Listing 4). Adnotacje tej klasy oznaczają:

* *@CrossOrigin* – koncept bezpieczeństwa zabraniający pobierania żądań od źródeł innych niż podane,
* @RestController – oznaczająca, że klasa jest wystawia API,
* *@RequestMapping* – oznaczające url, w które aplikacja webowa powinna uderzać,
* *@RequiredArgsConstruction* – oznacza, że Lombok wygeneruje konstruktor z wymaganymi polami (w tym przypadku pole *adapter*).

Metody zawarte w tej klasie oznaczone są adnotacjami @*GetMapping* oraz *@PostMapping* oznaczająco odpowiednio metodę API *GET* oraz *POST*. W adnotacjach tych podano też dalszy ciąg url.

Przed parametrami funkcji znajdują się adnotacje *@PathVariable* i *@RequestBody*. Oznaczają one, że wartość zmiennej będzie znajdować się odpowiednio w linku url w nawiasach klamrowych lub ciele otrzymanego żądania.

Każdy z pakietów zawiera też interfejs *translator* (Listing 6) używany przez narzędzie *Mapstruct* – jest to oznaczone przez adnotację *@Mapper* znajdującą się nad definicją interfejsu. Teraz należy już tylko zdefiniować funkcję z klasami, które chcemy zmapować.

@Repository  
**public interface** ExerciseRepository

**extends** JpaRepository<Exercise, Integer> {

Optional<Exercise> findExerciseById(**int** id);  
 List<Exercise> findAll(Specification<Exercise> toCriteria);  
 Optional<Exercise> getExerciseById(Integer id);  
 List<Exercise> getExercisesByDifficultyAndExerciseType(  
 Difficulty difficulty, ExerciseType exerciseType  
 );  
 @Query(  
 nativeQuery=**true**,  
 value=

**"SELECT \* FROM exercise e ORDER BY random() LIMIT ?1"** )  
 List<Exercise> fetchRandomExercises(  
 Integer numberOfExercisesToPick  
 );  
 @Query(**"SELECT e FROM Exercise e"** +  
 **" WHERE e.difficulty in ?1 and e.exerciseType = ?2 "** + **" and e.isForWomen = true"**)  
 List<Exercise> getExercisesByDifficultyAndExerciseTypeAndIsFemale(  
 List<Difficulty> difficulty, ExerciseType exerciseType  
 );

Listing 5.8: Repozytorium Exercise

@CrossOrigin(origins = **"http://localhost:3000"**)  
@RestController  
@RequestMapping(value =**"/api/exercise"**)  
@RequiredArgsConstructor  
**public class** ExerciseController {  
 **private final** ExerciseAdapter **adapter**;  
 @GetMapping(**"/details/{id}"**)  
 **public** ResponseEntity<ExerciseDetails> getExerciseDetails(  
 @PathVariable Integer id  
 ) {  
 ExerciseDetails exerciseDetails = **adapter** .getExerciseDetails(id);  
 **return** ResponseEntity.*ok*(exerciseDetails);  
 }  
 @PostMapping(value = **"/get"**)  
 @ResponseBody  
 **public** ResponseEntity<GetFilteredExerciseResp> getFilteredExercises(  
 @RequestBody GetFilteredExercisesReq request  
 ) {  
 GetFilteredExerciseResp resp = **adapter** .getFilteredExercises(request);  
 **return** ResponseEntity.*ok*(resp);  
 }  
}

Listing 5.9: Klasa ExerciseController

@Component  
@RequiredArgsConstructor  
**public class** ExerciseMediator **implements** ExerciseAdapter {  
 **private final** ExerciseRepository **repository**;  
 **private final** ExerciseTranslator **translator**;  
 @Override  
 **public** ExerciseDetails getExerciseDetails(Integer id) {  
 Optional<Exercise> maybeExercise = **repository** .getExerciseById(id);  
 **if** (maybeExercise.isEmpty()) {  
 **throw new** ExerciseNotFoundException(id);  
 }  
 **return translator**.toExerciseDetail(maybeExercise.get());  
 }

…

Listing 5.10: Klasa ExerciseMediator

@Mapper(componentModel = **"spring"**)  
**public interface** ExerciseTranslator {  
 ExerciseTranslator ***INSTANCE*** = Mappers.*getMapper*(  
 ExerciseTranslator.**class** );  
 ExerciseTableData toTableData(Exercise exercise);  
 List<Difficulty> toDifficultyEntityModels(  
 List<pl.edu.pwr.common.enums.Difficulty> difficulties  
 );  
 List<MuscleGroup> toMuscleGroupEntityModels(  
 List<pl.edu.pwr.common.enums.MuscleGroup>   
 muscleGroups  
 );  
 List<ExerciseType> toExerciseTypeEntityModels(  
 List<pl.edu.pwr.common.enums.ExerciseType>  
 exerciseTypes  
 );  
 ExerciseDetails toExerciseDetail(Exercise exercise);  
}

Listing 5.11: Translator klasy Exercise

Pozostała jeszcze jedna główna klasa pakietu e*xercise - Mediator* (Listing 5) implementująca interfejs *Adapter*. Odpowiada ona za logikę aplikacji, walidowanie danych i zwracanie ich do kontrolera.

W przypadku bardziej skomplikowanych operacji, tworzone zostawały dodatkowe klasy. Taka sytuacja miała miejsce przy generowaniu planu dla użytkownika w pakiecie *trainingPlan*, gdzie funkcje generowania planu były za duże do przechowania w jednej klasie.

Generowanie planu treningowego rozpoczyna się od sprawdzenia, czy obiekt wysłany w żądaniu nie jest pusty, a użytkownik, dla którego zostanie stworzony plan istnieje w bazie danych. Jeśli te dane są poprawne, następuje przypisanie atrybutów żądania do zmiennych zdefiniowanych w ciele funkcji.

Kolejnym etapem generowania planu jest określenie, jaki stopień zaawansowania określił użytkownik. Wykonuje się to w operacji *switch(),* która jako parametr przyjmuje wartość *difficulty*. Omówiona zostanie ścieżka generowania łatwego planu treningowego.

W przypadku *EASY* obliczana zostaje ilość wszystkich ćwiczeń do wygenerowania. Wartość ta wynika z pomnożenia liczby dni treningowych w jednym planie treningowym, liczby planów treningowych do wygenerowania oraz liczby ćwiczeń w jednym dniu treningowym. Następnie wywoływana jest funkcja z klasy *Service generate().*

W tej funkcji za pomocą funkcji *getExercisesToGenerateTrainingPlan()* oraz *generate()* następuje generowanie całego planu treningowego na kolejne tygodnie.

Funkcja dobierająca ćwiczenia do generowania planu jest najważniejszym elementem tej operacji. Bierze ona jako parametry: liczbę ćwiczeń do wygenerowania, typ tych ćwiczeń, wartość prawda/fałsz dotyczącą tego, czy użytkownik jest otyły, w podeszłym wieku lub młody, a także jego płeć (wartość płeć przyjmuje wartość prawda/fałsz w zależności tego, czy użytkownik jest kobietą).

Następnie ustawiana jest flaga, która zostanie opisana w punkcie 5.6 oraz wywoływana jest funkcja *getExercisesBasedOnFlag()*, która pobiera z bazy danych dokładnie te ćwiczenia, które w 100% spełniają kryteria użytkownika.

Funkcja *getExercisesToGenerateTrainingPlan()* kończy się przez wywołanie metody *getExercisesToCreatePlan()*.Metoda ta filtruje pobrane do tej pory ćwiczenia i wyrzuca z nich te, które nie będą trenowane dla tego poziomu trudności. Następnie, jeśli nie pobrano wystarczająco ćwiczeń z bazy danych, funkcja wchodzi do pętli *while()*, która wykonuje się tak długo, aż liczba ćwiczeń pobranych nie równa się liczbie ćwiczeń wymaganych lub liczba iteracji pętli nie przekroczy 50.

W środku pętli pobierane zostają losowe ćwiczenia. Rozmiar pobranej listy zależy od ilości brakujących ćwiczeń. Następnie przechodząc przez wszystkie pobrane ćwiczenia sprawdzamy, czy spełniają one warunki:

* ćwiczenie nie może być trudne oraz
  + typ ćwiczenia musi się zgadzać z typem planu treningowego lub
  + musi zgadzać się warunek zależny od flagi lub
  + zmienna losowa będzie większa od zadanej wartości.

Gdy ćwiczenie spełni warunek następują ostatnie dwa sprawdzenia. Pierwsze z nich sprawdza, czy rozmiar listy ćwiczeń do wygenerowania planu nie jest większy od liczby ćwiczeń, które należało pobrać. Natomiast drugie sprawdza, czy lista ćwiczeń nie zawiera już sprawdzanego ćwiczenia – jeśli tak, to pomija dodawanie go.

Program przechodząc przez te wszystkie kroki, w wyjątkowych sytuacjach, nadal może nie mieć wystarczającej liczby ćwiczeń do wygenerowania planu, dlatego wywoływana jest ostatnia funkcja *getMissingExercisesIfNecessaary()*. Jej zadaniem jest sprawdzenie, czy liczba ćwiczeń jest wystarczająca. Jeżeli odpowiedź jest przecząca, następuje pobranie brakującej ilości ćwiczeń. Nie jest wówczas przeprowadzana żadna walidacja ćwiczeń.

Po pobraniu wszystkich ćwiczeń algorytm przechodzi do tworzenia planu treningowego funkcją *generateTrainingPlan()*. Następuje iteracja przez liczbę planów treningowych do wygenerowania. W każdej iteracji zachodzi:

* zapisanie nowego planu treningowego w bazie danych w celu wygenerowania id,
* wywołanie funkcji *generateTrainingDays()*,
* przypisanie użytkownika i dni treningowych do planu.

Logika funkcji *generateTrainingDays()* jest podobna do funkcji tworzenia planu treningowego. Funkcja oblicza liczbę ćwiczeń dla jednego dnia, a następnie w pętli tworzy nowe dni treningowe i ustawia ich dane, w tym ćwiczenia z danymi przy użyciu funkcji *generateExercisesWithData()*. Pętla kończy się, gdy liczba wygenerowanych dni treningowych będzie odpowiadała ich wymaganej liczbie.

Funkcja *generateExercisesWithData()* iteruje od 0 do liczby dni na jeden dzień treningowy, tworząc listę ćwiczeń z danymi poprzez ustawianie liczby powtórzeń, liczby serii, losowo wybranych ćwiczeń z wcześniej stworzonej listy ćwiczeń (która będzie zmniejszana z każdą iteracją pętli), a następnie zwraca tę listę, by przypisać ją do dnia treningowego.

## Implementacja interfejsu użytkownika

Podział folderów w projekcie strony internetowej został opisany w punkcie 5.2. Na przykładzie *main-page* opisana zostanie implementacja logiki tworzącej aplikację internetową.

Głównym plikiem, w którym znajduje się szkielet strony oraz logika obsługująca zachowanie strony głównej jest *MainPageContainer.js.* Biblioteka *React* jest dość specyficzna. Strona zbudowana jest z funkcji, które zwracają kod HTML w pliku o rozszerzeniu *Javascript*. Takie jest przynajmniej pierwsze wrażenie, gdy ktoś niezaznajomiony spojrzy na kod aplikacji. Tak naprawdę, funkcje te zwracają kod JSX, które jest rozszerzeniem składni *Javascript*, wprowadzonym przez *Reacta*.

Główną funkcją w pliku jest *MainPageContainer()*, której pierwszym zadaniem jest zweryfikowanie, czy użytkownik jest zalogowany. Używa do tego *hooka[[2]](#footnote-2)* *useEffect()*, który uruchamia funkcję lambda zawsze podczas pierwszej inicjalizacji oraz w zależności od tego, jakie parametry podaliśmy jako drugi parametr. Pusta lista oznacza, że *hook* uruchomi się tylko raz.

Wywołana zostaje funkcja wyeksportowana z pliku *CheckIfUserIsLoggedIn.js.* Parametry tej funkcji to:

* drugi parametr *hooka* *useState()*, gdzie *hook* ten możemy porównać do zdefiniowania zmiennej *x* oraz funkcji ustawiającej wartość tej zmiennej w jednej linijce kodu,
* zmienna *navigate* – będąca *hookiem* *useNavigate()*, służącego do nawigowania się po różnych stronach za pomocą kodu.

Metoda *isUserLoggedIn()* sprawdza, czy w pamięci przeglądarki zapisany jest obiekt *user*. Jeśli nie ma takiego obiektu to użytkownik zostaje przekierowany na stronę logowania. Jeśli obiekt taki istnieje i jego parametr *isUserLoggedIn* ma wartość *true* to na taką wartość ustawiona zostaje też zmienna *userId*.

export function isUserLoggedIn(setUserId, navigator) {

    let userFromMem = JSON.parse(window.localStorage.getItem('user'));

    if(!userFromMem || (userFromMem && (!userFromMem.userId || !userFromMem.isUserLoggedIn))) {

        navigateToLoginPage(navigator);

    }

    if(userFromMem &&  userFromMem.isUserLoggedIn) {

        setUserId(userFromMem.userId);

    }

}

Listing 5.12: Funkcja sprawdzająca, czy użytkownik jest zalogowany

Przy pierwszej inicjalizacji widoku strony głównej uruchamiane zostają jeszcze dwa dodatkowe *hooki* *useEffect()* (Listing 8). Ich zadaniem jest pobieranie listy planów treningowych do wyświetlenia w sekcji *Twoje plany treningowe*.

useEffect(() => {

        if (userId) {

            fetchTrainingPlans(

                userId, setIsLoading, handleSetTrainingPlans, showErrorMsg

            )

            setFetchAfterGenerated(false);

        }

    }, [userId, fetchAfterGenerated]);

useEffect(() => {

        if(userId && fetchAfterDelete) {

            fetchTrainingPlans(

                userId, setIsLoading, handleSetTrainingPlans, showErrorMsg

            );

        }

    }, [fetchAfterDelete]);

Listing 5.13: Hooki pobierające plany treningowe

Uruchamiane są w momencie pierwszej inicjalizacji oraz za każdym razem, gdy któraś ze zmiennych (*userId, fetchAfterGenerated, fetchAfterDelete*) zostanie zmieniona. By zapobiec próbie pobrania planów, nawet gdy użytkownik nie jest zalogowany, w środku funkcji anonimowej znajduje się warunek sprawdzający, czy użytkownik jest zalogowany i tylko w tym przypadku pobierane są plany treningowe. Pobieranie planów jest zadaniem metody *fetchTrainingPlans()* (Listing 9)

Funkcja ta definiuje na początku kontroler mający na celu zaprzestanie wysyłania i odbierania jakichkolwiek danych w momencie, gdy użytkownik przejdzie na inną stronę.

Metoda *fetch()* jako parametry otrzymuje url oraz obiekt ustawiający specyfikacje żądania:

* *method* – oznacza jaki typ zapytania REST będzie wysyłany,
* *headers* – *content-type* wskazuje jaki typ danych będzie zwracany ze źródła,
* *signal* – jako wartość otrzymuje sygnał kontrolera zdefiniowanego na początku funkcji.

W tym momencie kontroler serwera wystawiający api na podanym url zaczyna swoją pracę i zwraca odpowiednie plany treningowe.

const fetchTrainingPlans = (id, setIsLoading, handleSetTrainingPlans, showErrorMsg) => {

    const controller = new AbortController();

    setIsLoading(true);

    fetch("http://localhost:8080/api/trainingPlan/" + id, {

        method: 'GET',

        headers: {

            'Content-Type': 'application/json'

        },

        signal: controller.signal,

        })

        .then(resp => {

            if (resp.ok) {

                return resp.json();

            }

            throw resp;

        })

        .then(data => {

            if(data){

                handleSetTrainingPlans(data.trainingPlanTableData, showErrorMsg);

            }

        })

        .catch(error => {

            showErrorMsg();

            throw error;

        })

        .finally(() => {

            setIsLoading(false)

        });

        return () => {

            controller.abort();

        }

}

Listing 5.14: Ciało funkcji pobierającej plany treningowe

Gdy serwer zwróci dane wykonuje się pierwsza metoda *then()*, która sprawdza, czy odpowiedź jest prawidłowa i jeśli tak, wykonuje się druga metoda o tej samej nazwie, w której możemy operować na otrzymanym z serwera obiekcie. W tym przypadku wykonuje się funkcja *handleSetTrainingPlans()*, która mapuje otrzymany obiekt i zapisuje te obiekty w zmiennej *userTrainingPlans*. Po wykonaniu tej operacji wykonuje się funkcja *finally()*.

Można zauważyć, że od funkcji *fetch()* odchodzi jeszcze jedna metoda *catch*(), której zadaniem jest obsługa wyjątków zwróconych z serwera.

W międzyczasie funkcja *MainPageContainer()* zwraca kod JSX, w którym znajdują się odwołania *UserTrainingPlans* oraz *UserDataToGeneratePlan*. Odnoszą się one do funkcji znajdujących się w plikach *UserTrainingPlans.js* oraz *UserDataToGeneratePlan.js*.

Do funkcji *UserTrainingPlans()* jako parametry przekazywane są plany treningowe, flaga mówiąca o tym, czy w danym momencie są pobierane dane planów treningowych oraz funkcja usuwająca plan z listy. Metoda ta służy do wyświetlania tabeli z danymi dotyczącymi planów. Obsługuje też usuwanie planu treningowego z bazy danych. Wysyła żądanie podobne do tego z Listing 9, różnicą jest to, że metoda ustawiona jest teraz na *DELETE* oraz uderza w inny *endpoint*. Po otrzymaniu prawidłowej odpowiedzi z serwera wywoływana jest z funkcji *UserTrainingPlans()* metoda *deletePlanById()* znajdująca się w *MainPageContainer*.

## Problemy implementacyjne

Naturalnym jest napotykanie problemów podczas implementacji aplikacji lub jakiegoś rozwiązania. Nie inaczej było w przypadku tej strony internetowej. Napotkano wiele mniejszych i kilka większych problemów:

* wysyłanie obiektów typu enum między *backendem* a bazą danych,
* błąd *cross-origin resource sharing* przy wysyłaniu żądań z *frontendu* do *backendu*,
* prosta implementacja pobierania ćwiczeń do generowania planu treningowego, które spełniać będą wszystkie kryteria.

### Błąd CORS przy wysyłaniu żądań

W momencie, gdy wysłane zostało żądanie z aplikacji internetowej, zwracany był błąd, który sprawiał, że nie można było współdzielić zasobów z serwerem. Rozwiązaniem było dodanie nowej klasy *WebConfig* z dwoma adnotacjami: *@Configuration* oraz *@EnableWebMvc*. Klasa ta dodatkowo została rozszerzona o klasę abstrakcyjną *WebMvcConfigurerAdapter*. Dzięki temu można było nadpisać funkcję *addCorsMappings()*, w której do rejestru dodano źródło <http://localhost:3000> - port zajęty przez aplikację.

Dodatkowo do każdego kontrolera serwera dodano adnotację *@CrossOrigin* przedstawioną w punkcie 5.4.

### Implementacja pobierania ćwiczeń do generowania planu treningowego

Encja ćwiczeń w bazie danych ma pola trudność, typ oraz 4 pola definiujące, czy osoba ćwicząca jest otyła, w podeszłym wieku, młoda lub czy jest kobietą. Istnieje 12 kombinacji parametrów osoby. By uniknąć pisania funkcji zawierającej w sobie 12 skomplikowanych wyrażeń *if*-*else* zadeklarowano zmienną *flag*. Jej wartość zwracana jest przez funkcję *getFlagValue()* zdefiniowaną w klasie *TrainingPlanUtils*.

Rozwiązanie to miało być szybką „implementacją” 4-bitowej flagi, gdzie wartość „bitu” jest ustawiana, jeśli odpowiadający mu atrybut ćwiczenia jest równy *true*.

Ustawienie flagi umożliwia prosty sposób znalezienia odpowiedniej funkcji z *ExerciseRepository*. Funkcja *getExercisesBasedOnFlag()* zbudowana jest z wyrażenia *switch*, które w zależności od wartości flagi wybiera prawidłową metodę do wywołania, która zwróci listę ćwiczeń spełniających odpowiednie kryteria.

### Komunikacja z użyciem obiektów enum

Tworzenie typów *enum* w bazie danych oraz po stronie *backendu* stworzył duży problem z mapowaniem obiektów podczas komunikacji *serwer-baza*\_*danych*.

Rozwiązanie tego problemu przedstawił użytkownik Vlad Mihalcea **Błąd! Nie można odnaleźć źródła odwołania.**, który jest autorem biblioteki *hybernate-types* **Błąd! Nie można odnaleźć źródła odwołania.**.

W aplikacji na początku zaimplementowane były cztery klasy *enum* odpowiadające tym z bazy danych. Należało jednak stworzyć kolejne cztery klasy *enum* rozszerzone o konstruktor i dwie metody:

* *toString()* – z adnotacją *@JsonValue*,
* *fromValue()* – z adnotacją *@JsonCreator*.

Następnie należało dodać klasę *PostgreSQLEnumType*, która rozszerzała klasę *org.hibernate.type.EnumType*. Znajduje się w niej jedna funkcja *nullSafeSet()*. Teraz, gdy klasa odpowiadająca encji bazy danych używa zmiennych o typie *enum*,należy dodać nad taką klasą adnotację *@TypeDef*, w której ustawiamy pola *name* na *psql\_enum* oraz *typeClass* na klasę *PostgreSQLEnumType.class*. Pola *enum* takiej klasy powinny być dodatkowo oznaczone adnotacją *@Type,* gdzie wartość *type* jest równa wartości podanej w polu *name*.

Po wykonaniu tych operacji komunikacja i mapowanie *enumów* znajdujących się na serwerze i bazie danych powinna przebiegać pomyślnie.

# Testowanie aplikacji

Nieważne jak dobrze napisany jest kod, jeśli nie jest przetestowany, to nie jest on wiarygodny. Aplikacja została przetestowana na dwa sposoby. Pierwszy z nich to testy manualne po stronie *frontendu*, a drugie to testy *backendu* przy użyciu narzędzia *JUnit*

## Przeprowadzone testy

Testy *backendu* obejmują działanie klas *Mediator* obsługujących główną logikę aplikacji. Sprawdzały one poprawność działania aplikacji podczas przekazywania do niej danych prawidłowych oraz danych, które powodowały wyrzucanie wyjątku. Główna logika tych testów była podzielona na 3 części:

* *given* – w której deklarowane są zmienne testowe oraz określane jest zachowanie metod zamockowanych klas lub interfejsów,
* *when* – w tej części testu wywoływana jest metoda do testowania,
* *then* – w której znajdują się sprawdzenia, czy zwrócone dane mają wartości zgodne z oczekiwaniami.

Przykładem testu może być logowanie się użytkownika po stronie serwera. W sekcji *given* definiowane są zmienne, a następnie przy użyciu funkcji *when()* pochodzącej z narzędzia *Mockito* definiowane jest zachowanie funkcji *findGymUserByLogin()*, która implementowana jest przez repozytorium klasy *GymUser.*

@DisplayName(**"Logged In - Correct Data"**)  
@Test  
**public void** LogInCorrectly() {  
 *// given* Integer id = 1;  
 String login = **"login"**;  
 String password = **"password"**;  
 GymUser user = **new** GymUser();  
 user.setId(id);  
 user.setLogin(login);  
 user.setPassword(password);  
 Optional<GymUser> maybeUser = Optional.*of*(user);  
 *when*(**repository**.findGymUserByLogin(login)).thenReturn(maybeUser);  
 LoginUserReq req = **new** LoginUserReq();  
 req.setLogin(login);  
 req.setPassword(password);  
 *// when* LoginUserResp resp = **mediator**.loginUser(req);  
 *// then  
 assertEquals*(id, resp.getUserId());  
 *assertTrue*(resp.getIsUserLoggedIn());

}

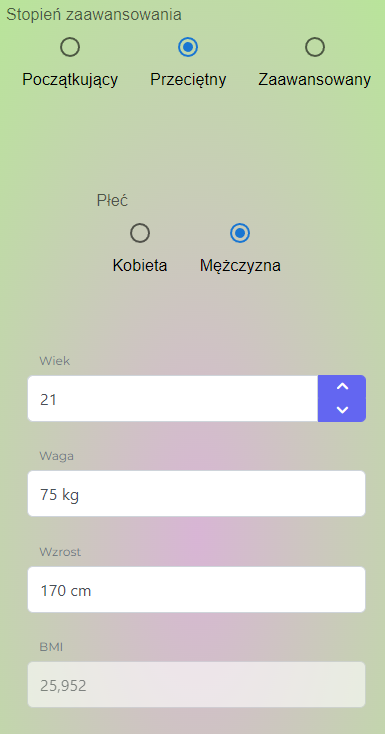
Listing 6.1: Test Logowania się użytkownika

W przypadku tego testu zwracany obiekt będzie zawierał dane, które należy sprawdzić przy pomocy funkcji *assertEquals()* oraz *assertTrue()*, które w przypadku niezgodności oznaczą test jako wykonany z niepowodzeniem.

Strona internetowa była przetestowana w sposób manualny. Na każdej podstronie i sekcji wprowadzane były dane, które mogły spowodować popsucie działania aplikacji. Testy wypadły dobrze, aplikacja działała prawidłowo. Funkcjonalności zostały zaimplementowane poprawnie, a wiele pól jest objęte walidacją danych, co ogranicza ryzyko błędów. Komunikaty wyświetlają się użytkownikowi podczas korzystania z aplikacji. Podczas poprawnie wykonanej operacji lub operacji przerwanej przez błąd, użytkownik otrzyma odpowiedni komunikat.

## Wyniki testów manualnych

Najważniejszą funkcjonalnością aplikacji jest generowanie planów. W systemie istnieją trzy rodzaje ćwiczeń – siłowe, kardio oraz mobilności. Dla treningu siłowego generowane zostają cztery tygodnie treningowe, natomiast dla pozostałych - jeden. Przeprowadzono test dla przeciętnego użytkownika wybierającego trening siłowy z czterema dniami treningowymi w tygodniu. Sekcja *Wygeneruj swój plan* została przedstawiona na Rysunek 6.1.



Rysunek 6.1: Sekcja "Wygeneruj swój plan"

Po wciśnięciu przycisku *Wygeneruj plan!* należy oczekiwać pojawienia się w sekcji *Twoje plany treningowe* czterech planów treningowych, gdzie każdy z tych planów będzie zawierał cztery dni treningowe z ćwiczeniami, których dominująca część jest przeznaczona do poprawy siły.

Wynikiem operacji są plany treningowe przedstawione na Rysunek 6.2. Dominującym typem każdego z planów zgodnie z oczekiwaniami jest typ *Siła*. Daty dni treningowych nie nachodzą na siebie – żaden z nich nie trwa dłużej niż 7 dni, a to oznacza, że wygenerowane plany spełniają wymagania funkcjonalne podane w punkcie 2.2.

Przerwa pomiędzy czterema dniami treningowymi w planie wynosi jeden dzień. Dzień treningowy składa się z 6 ćwiczeń z zaplanowaną liczbą powtórzeń i serii dla każdego z nich.

Obraz zawierający stół

Opis wygenerowany automatycznie

Rysunek 6.2: Wygenerowane plany treningowe.

Ręczne tworzenie planów treningowych jest drugą najważniejszą funkcjonalnością aplikacji. Użytkownik samodzielnie wybierając ćwiczenia jest w stanie tworzyć dni treningowe. Z dni składać się będzie plan treningowy. Podczas wybierania daty dnia treningowego ważna jest jej unikalność. W przypadku powtórzenia dat użytkownik otrzyma komunikat informujący go o tym, że dni w planie muszą się różnić.

Wynik tworzenia planu możemy zobaczyć na stronie głównej aplikacji. Oprócz planów, które są aktualnie przypisane do użytkownika, pojawi się tam plan, który użytkownik ręcznie stworzył.

# Podsumowanie

Implementacja aplikacji internetowej pozwoliła na poznanie nowych technologii oraz rozwinięcie umiejętności pisania dobrego, czytelnego kodu oraz jego testowania. Doświadczenie zdobyte podczas studiów pozwoliło na zaprojektowanie prostej, ale wystarczającej do potrzeb aplikacji, bazy danych, której tabele i relacje można w przyszłości użyć w innych projektach o podobnej tematyce.

## Wnioski

Celem pracy było stworzenie aplikacji internetowej, która generuje plany treningowe dla użytkownika w zależności od jego preferencji. Efekt końcowy pracy pozwala na nieco więcej niż to. Aplikacja pozwala na generowanie planu, ale i ręczne stworzenie planu poprzez samodzielne wybieranie ćwiczeń. Plany można przeglądać, edytować i usuwać. Ćwiczenia znajdujące się w bazie danych są zrozumiale opisane. Zawierają film instruktażowy pokazujący jak prawidłowo wykonywać ćwiczenie, którym można posłużyć się do jego nauki. Gdy osoba korzystająca z aplikacji nie jest zalogowana, nie zostanie wyświetlona zawartość strony. Użytkownik może logować się i rejestrować w aplikacji. Dane użytkownika tj. waga, wzrost i wiek są aktualizowane za każdym razem, gdy ten wygeneruje swój plan treningowy, Użytkownik ma możliwość zmiany swojego loginu,   
e-mailu i hasła. Po stronie serwera i strony internetowej przed wykonaniem jakiejkolwiek operacji sprawdzana jest najpierw spójność danych. Na *backendzie* zdefiniowano klasy rozszerzające klasę *RuntimeException* w celu wyświetlenia dokładniejszych informacji dotyczących błędów pojawiających się w trakcie działania aplikacji. Baza danych postawiona jest za pomocą narzędzia *Docker*, które jest jednym z najpopularniejszych narzędzi do konteneryzacji aplikacji w małych, lekkich maszynach wirtualnych. Narzędzie *Flyway* pozwala na uzupełnianie bazy danymi za każdym razem, gdy uruchomimy serwer.

## Koncepcja rozwoju aplikacji

Aplikacja ma wiele ciekawych sposobów, w jakie można ją rozwinąć. Istnieje bowiem wiele funkcjonalności, które można dodać do implementacji. Mogą to być:

* wprowadzenie bezpieczeństwa przy pomocy narzędzi takich jak *Keycloak* lub *SpringSecurity*,
* podział roli użytkowników,
* nowe role mogą skutkować powstaniem użytkownika, który jest trenerem personalnym,
* implementowanie czatu między użytkownikami tj. trener personalny i osoba, która zaczyna ćwiczyć na siłowni,
* dodanie możliwości dodawania, edycji i usuwania ćwiczeń z poziomu aplikacji dla roli *Admin*,
* rozszerzenie aplikacji o układanie diety, która jest ważną częścią treningów,
* dodanie funkcjonalności oceniania wygenerowanych planów i ćwiczeń,
* implementacja generowania planu treningowego używając nauczania maszynowego,
* poprawienie ogólnego wyglądu aplikacji.

Najciekawszym z punktów wydaje się być generowanie planu używając nauczania maszynowego. Wadą tego rozwiązania byłby fakt, że duża liczba użytkowników musiałaby generować i oceniać swoje plany, by model zaczął generować je poprawniej i były dopasowane do użytkownika.

Jeśli aplikacja miałaby kiedykolwiek zostać wdrożona i przekazana do publicznego użytku, wymaganym byłoby zaimplementowanie systemów bezpieczeństwa tj. zabezpieczenie poufnych danych użytkowników i zablokowanie dostępu do tych podstron, do których użytkownik o danej roli nie ma dostępu.

# 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.

# Spis rysunków

[Rysunek 1: Popularność siłowni w Polsce na przestrzeni lat 2017-2022 7](#_Toc121343141)

[Rysunek 2. Strona główna fabrykasily.pl 9](#_Toc121343142)

[Rysunek 3: Koszty pakietów treningowych na stronie budujmase.pl **Błąd! Nie zdefiniowano zakładki.**](#_Toc121343143)

[Rysunek 4: Diagram związków encji 11](#_Toc121343144)

[Rysunek 5: Diagram przypadków użycia **Błąd! Nie zdefiniowano zakładki.**](#_Toc121343145)

[Rysunek 6: Ekran menu głównego 15](#_Toc121343146)

[Rysunek 7: Ekran listy dni treningowych 15](#_Toc121343147)

[Rysunek 8: Ekran edycji dnia treningowego 16](#_Toc121343148)

[Rysunek 9: Ekran listy ćwiczeń z danego dnia treningowego 16](#_Toc121343149)

[Rysunek 10: Ekran tworzenia planu treningowego 17](#_Toc121343150)

[Rysunek 11: Ekran listy wszystkich ćwiczeń 18](#_Toc121343151)

[Rysunek 12: Ekran szczegółów ćwiczenia 19](#_Toc121343152)

[Rysunek 17: Porównanie zainteresowania Angularem i ReactemJS 21](#_Toc121343153)

[Rysunek 14: Struktura pakietów na przykładzie TrainingPlan 27](#_Toc121343154)

[Rysunek 15: Sekcja "Wygeneruj swój plan" 41](#_Toc121343155)

[Rysunek 16: Wygenerowane plany treningowe. 42](#_Toc121343156)

# Spis listingów

[Listing 1: Skrypt do generowania tabeli Exercise 29](#_Toc121342989)

[Listing 2: Klasa Exercise 30](#_Toc121342990)

[Listing 3: Repozytorium Exercise 32](#_Toc121342991)

[Listing 4: Klasa ExerciseController 32](#_Toc121342992)

[Listing 5: Klasa ExerciseMediator 33](#_Toc121342993)

[Listing 6: Translator klasy Exercise 33](#_Toc121342994)

[Listing 7: Funkcja sprawdzająca, czy użytkownik jest zalogowany 36](#_Toc121342995)

[Listing 8: Hooki pobierające plany treningowe 36](#_Toc121342996)

[Listing 9: Ciało funkcji pobierającej plany treningowe 37](#_Toc121342997)

[Listing 10: Test Logowania się użytkownika 40](#_Toc121342998)

1. „Frameworki określają strukturę aplikacji, jej mechanizm działania oraz dostarczają biblioteki i komponenty przydatne podczas tworzenia programów.” [1] [↑](#footnote-ref-1)
2. „Hooki są to funkcje, które pozwalają „zahaczyć się” w mechanizmy stanu i cyklu życia Reacta, z wewnątrz komponentów funkcyjnych.” **Błąd! Nie można odnaleźć źródła odwołania.** [↑](#footnote-ref-2)