



**Politechnika  
Śląska**

## **PROJEKT INŻYNIERSKI**

System do monitorowania położenia geoprzestrzennego obiektów

**Jakub SIBIK**

**Nr albumu: 300420**

**Kierunek:** Informatyka

**Specjalność:** Informatyczne Systemy Mobilne i Przemysłowe

**PROWADZĄCY PRACĘ**

**dr inż. Łukasz Wycislik**

**KATEDRA Informatyki Stosowanej**

**Wydział Automatyki, Elektroniki i Informatyki**

**Gliwice 2025**



## **Tytuł pracy**

System do monitorowania położenia geoprzestrzennego obiektów

## **Streszczenie**

Niniejsza praca inżynierska zajmuje się lokalizacją, opartą o system GPS. Jej celem jest śledzenie obiektów. Aby go zrealizować, stworzona została aplikacja, która ma możliwość wyświetlania tras przebytych przez konkretne obiekty, takie jak lokalizator, kierowca lub pojazd. Praca opisuje proces budowania programu oraz zawiera niezbędne informacje, które powinna zawierać dokumentacja tego typu projektu. W pierwszej kolejności temat został przeanalizowany. Następnie wybrano odpowiednie narzędzia, stos technologiczny oraz sformułowano wymagania funkcjonalne i niefunkcjonalne. Kolejnym krokiem było sporządzenie specyfikacji, zarówno zewnętrznej, opisującej wygląd aplikacji, sposób obsługi oraz wymagania systemowe, jak i wewnętrznej, składającej się z opisu architektury i struktur danych, popartych licznymi schematami i diagramami. W tej części znajduje się również wytłumaczenie działania niektórych fragmentów kodu. Przeprowadzono testy aplikacji, aby wyeliminować powstałe w trakcie tworzenia kodu błędy. Na końcu sformułowano wnioski i porównano efekty do wyznaczonych wcześniej celów.

## **Słowa kluczowe**

Lokalizator, pojazd, kierowca, aplikacja, program

## **Thesis title**

System for geospatial monitoring of objects

## **Abstract**

This thesis deals with location, based on a GPS system. Its aim is to track objects. In order to make it happen, an application that has the ability to display routes traveled by specific objects, such as tracker, driver or vehicle was created. Thesis describes the process behind building the program and it contains necessary information that should be included in the documentation of this type of project. First, the topic was analysed. Next, the right tools were chosen, as well as technology stack and functional and non-functional requirements were written down. The next step was to prepare specifications, both external, which describes user interface, manual and system requirements, and internal, which consists of a description of the architecture and data structures, surrounded with numerous diagrams. There are also some explanations for fragments of code in this section. Tests were carried out to eliminate errors in application, which could have ap-

peared during coding. At the end, conclusions were made and the effects were compared to the previously set goals.

### **Key words**

Tracker, vehicle, driver, application, program

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>1</b>
1.1	Wprowadzenie w problem	1
1.2	Cel pracy	2
1.3	Zakres pracy	2
1.4	Charakterystyka rodzin	2
<b>2</b>	<b>Analiza tematu</b>	<b>3</b>
2.1	Sformułowanie problemu	3
2.2	Osadzenie tematu w kontekście aktualnego stanu wiedzy	3
2.3	Opis znanych rozwiązań	4
<b>3</b>	<b>Wymagania i narzędzia</b>	<b>7</b>
3.1	Wymagania funkcjonalne	7
3.2	Wymagania niefunkcjonalne	8
3.3	Stos technologiczny	8
3.3.1	Java	10
3.3.2	Spring Boot	10
3.3.3	JavaScript	10
3.3.4	TypeScript	10
3.3.5	React	11
3.3.6	PostgreSQL	11
3.3.7	PostGIS	11
3.3.8	Hibernate	11
3.3.9	Git	12
3.4	Oprogramowanie do edycji kodu	12
3.4.1	IntelliJ Idea	12
3.4.2	Visual Studio Code	12
3.4.3	PgAdmin	12
3.4.4	GitHub	13
3.5	Metodyka pracy	13

<b>4 Specyfikacja zewnętrzna</b>	<b>15</b>
4.1 Wymagania sprzętowe i programowe	15
4.2 Kategorie użytkowników	15
4.3 Sposób obsługi	16
<b>5 Specyfikacja wewnętrzna</b>	<b>29</b>
5.1 Przedstawienie architektury systemu	29
5.2 Opis struktur danych	31
5.3 Model danych	34
5.4 Szczegóły implementacji wybranych fragmentów	34
5.4.1 Komponenty React	34
5.4.2 Proces generowania i obsługi żądań	36
<b>6 Weryfikacja i walidacja</b>	<b>41</b>
6.1 Sposób testowania	41
6.2 Wykryte i usunięte błędy	42
<b>7 Podsumowanie i wnioski</b>	<b>43</b>
7.1 Uzyskane wyniki w świetle postawionych celów i zdefiniowanych wymagań	43
7.2 Kierunki dalszego rozwoju systemu	43
<b>Bibliografia</b>	<b>45</b>
<b>Lista dodatkowych plików, uzupełniających tekst pracy</b>	<b>49</b>
<b>Spis rysunków</b>	<b>52</b>

# Rozdział 1

## Wstęp

Ten rozdział ma na celu przybliżenie czytelnikowi zagadnienia systemów lokalizacji, będących ściśle związanych z tematem pracy, jak również przedstawienie zakresu, celu pracy oraz krótki opis kolejnych rozdziałów, które niniejsza praca inżynierska zawiera.

### 1.1 Wprowadzenie w problem

Początki technologii do określania pozycji sięgają lat 60. XX wieku. Powstał wtedy system NAVSAT (ang. Navigation Satellite System) - będący pierwszym satelitarnym systemem nawigacyjnym. Został on opracowany przez Stany Zjednoczone oraz wykorzystywany był przez tamtejszą marynarkę wojenną. W latach 70. XX wieku postanowiono wprowadzić międzynarodowy standard, dzięki czemu powstał system GPS (ang. Global Positioning System), który jest używany po dziś dzień. Więcej na temat jego historii znajduje się w książce Jeffa Hurna "GPS: A Guide to the Next Utility" [6].

Przez lata system ten był rozwijany, co zaowocowało jego dostępnością dla przeciętnych użytkowników. W efekcie tego, GPS jest wsparciem dla ludzi w wielu dziedzinach. W obecnych czasach ponad połowa światowej populacji posiada smartfony, które to mają wbudowane systemy GPS. Pozwala to przede wszystkim na sprawną nawigację do celu czy dokładne ustalenie pozycji danej osoby. Samochody również są w posiadaniu znacznej części populacji, a ponadto stanowią dosyć znaczną część budżetu domowego. Z tego względu ludzie zaopatrują się w lokalizatory samochodowe. Podczas kupna takiego urządzenia, klient otrzymuje zazwyczaj dostęp do strony internetowej, na której jest w stanie sprawdzać położenie swojego samochodu, w którym został umieszczony lokalizator. Takie rozwiązania są dosyć proste, niewystarczające dla wielu użytkowników, wygląd interfejsu również pozostawia wiele do życzenia. Wraz z rozwojem komputerów oraz smartfonów, zwiększa się możliwości do stworzenia aplikacji do zarządzania lokalizatorami w pojazdach, która oferowałaby większą ilość funkcjonalności, oraz która byłaby atrakcyjniejsza niż podstawowe odpowiedniki, będące obecnie na rynku.

## 1.2 Cel pracy

Celem niniejszej pracy inżynierskiej jest śledzenie lokalizacji obiektów. Umożliwi to aplikacja, pozwalająca na zarządzanie lokalizatorami, pojazdami i kierowcami, która usprawniłaby obługę tego typu urządzeń w obrębie rodziny lub firmy posiadającej flotę samochodów i ułatwiałaby przeglądanie tras przebytych przez poszczególne osoby, pojazdy czy lokalizatory.

## 1.3 Zakres pracy

Praca obejmuje proces i sposób tworzenia oprogramowania, specyfikacje: zewnętrzną i wewnętrzną, testowanie, jak również efekty i wnioski.

## 1.4 Charakterystyka rozdziałów

Następnym rozdziałem jest rozdział drugi, będący zatytuowanym "Analiza tematu". Jego treść ukazuje proces rozeznania autora w temacie lokalizatorów, jak również porównanie planowanego projektu do istniejących rozwiązań. Znajduje się tam też proces wyboru stosu technologicznego. Kolejny rozdział - trzeci - nosi nazwę "Wymagania i narzędzia". Pierwszą jego część stanowią wymagania, które dzielą się na funkcjonalne oraz niefunkcjonalne. Obydwa ich rodzaje są wymienione i opisane. Drugą częścią rozdziału są narzędzia. Autor przedstawia w niej niezbędne urządzenia, programy oraz wybrany stos technologiczny, niezbędny do zbudowania projektu. "Specyfikacja zewnętrzna" jest rozdziałem czwartym. Znajduje się tam przede wszystkim opis widoku systemu połączony z instrukcją obsługi. W zrozumieniu tekstu pomagają umieszczone na rysunkach zrzuty ekranu, przedstawiające wygląd konkretnych fragmentów programu. W rozdziale czwartym sformułowane są także wymagania sprzętowe i programowe, stanowiące o urządzeniach, na których aplikacja będzie w stanie zostać uruchomiona, a także przedstawione są rodzaje użytkowników systemu oraz funkcje, do których mają dostęp. Piąty rozdział ma tytuł "Specyfikacja wewnętrzna". W tej części pracy opisane zostały struktury i architektura danych, które zostały wykorzystane w projekcie, a zatem również przedstawiona została baza danych, poparta schematem. Rozdział ten posiada też opis fragmentów kodu źródłowego projektu. Następnym rozdziałem - szóstym - jest "Weryfikacja i validacja". Dotyczy on przeprowadzania testów i sprawdzeń pod kątem błędów, które mogły się pojawić podczas tworzenia programu. Ostatni, siódmy rozdział zatytuowany jest "Podsumowanie i wnioski". W nim zostały przedstawione zostały efekty uzyskane w projekcie w porównaniu do wstępnych założeń, jak również przemyślenia autora, które wytworzyły się podczas prac nad aplikacją oraz niniejszą pracą inżynierską.

# Rozdział 2

## Analiza tematu

Aby rozpocząć działania nad projektem, w pierwszej kolejności należało przeanalizować problem. Ponadto, znaleziono podobne rozwiązania, które mogą być wzorem lub pomocą przy dążeniu do innowacyjnego projektu.

### 2.1 Sformułowanie problemu

Aplikacja do śledzenia lokalizatorów może zostać stworzona w różnych językach programowania. Może być również dostępna na różnego rodzaju urządzeniach, od urządzeń mobilnych, po komputery stacjonarne. Ważna jest również kompatybilność programu z konkretnym lokalizatorem. Niezbędne jest zastanowienie się nad tymi kwestiami, aby móc przejść do kolejnych kroków tworzenia aplikacji.

### 2.2 Osadzenie tematu w kontekście aktualnego stanu wiedzy

Wśród lokalizatorów dostępnych na rynku można znaleźć wiele rozwiązań. W zależności od ceny, możemy otrzymać dodatkowe funkcjonalności, mniej lub bardziej znaczące, są to między innymi czujnik wstrząsu, monitorowanie prędkości czy podsłuch. Producenci zazwyczaj posiadają własne strony internetowe, do których klient otrzymuje bezpłatny dostęp po zakupie produktu. Pozwalają one zwykle powiązać lokalizator z kontem użytkownika i śledzić na bieżąco jego lokalizację.

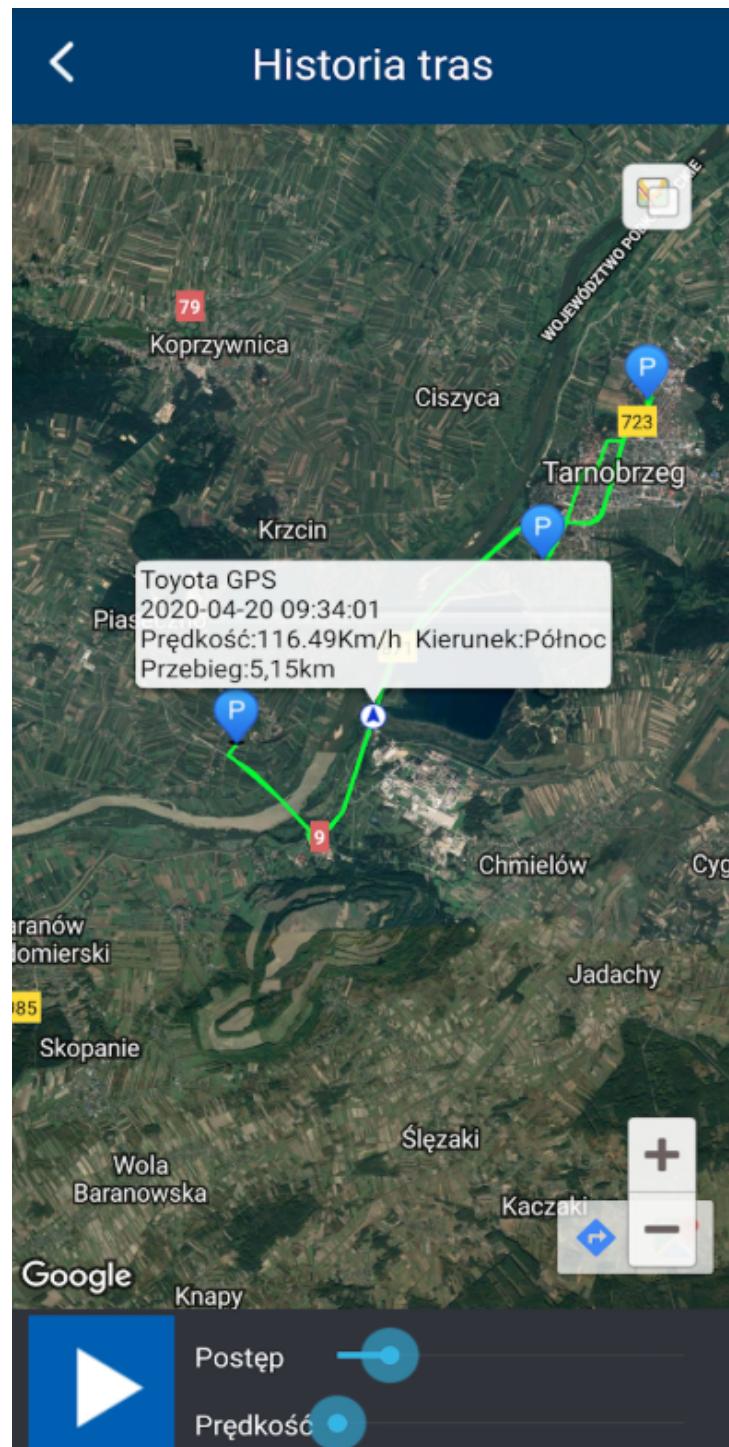
W celu stworzenia aplikacji do monitorowania lokalizatorów dla jak największej liczby ludzi, korzystających z tego typu urządzeń, warto przeanalizować kilka kwestii. Niezbędna funkcją lokalizatora jest możliwość jego konfiguracji, aby wysyłał dane na konkretny adres IP oraz port. Umożliwi to dostęp aplikacji do lokalizacji urządzenia. Szukane więc będą lokalizatory, do których można włożyć kartę SIM. Z tego powodu odrzucono produkty o

najniższej cenie na rynku, gdyż nie oferują one wymaganej do działania programu funkcji. Przykładem jest Samsung SmartTag2, który nie umożliwia wysyłania danych lokalizacyjnych na konkretny adres. Kolejnym znaczącym aspektem wyboru lokalizatora jest jego cena w kontekście klienta. Należy wykluczyć najdroższe opcje, aby nie ograniczyć ilości potencjalnych użytkowników aplikacji. Model GP800+, którego cena sięga powyżej sześciuset złotych, może być nieosiągalny dla wielu osób. Przykładami, które odpowiadają wyżej opisanym wymaganiom, są Coban TK108 lub Mking MK07A.

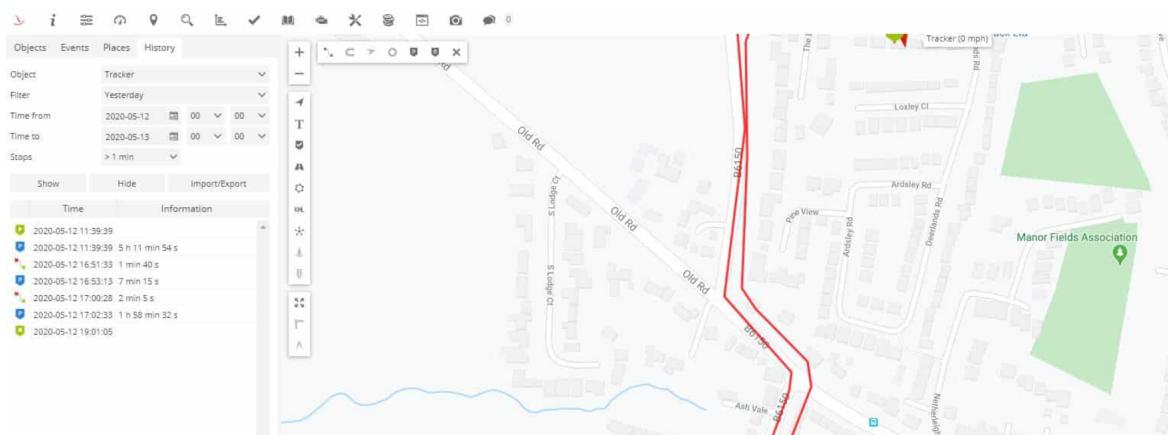
Podczas przeprowadzania analizy tematu, należało również wybrać stos technologiczny, w którym aplikacja będzie tworzona. Możliwe jest stworzenie aplikacji mobilnej, dostępnej przede wszystkim na smartfony, co wiąże się z wykorzystaniem języków programowania takich jak Flutter, React Native, Kotlin czy Swift. Kolejną opcją jest stworzenie aplikacji przeglądarkowej. Ten typ programów najczęściej pisany jest przy użyciu języków: Java, Python, JavaScript.

## 2.3 Opis znanych rozwiązań

MKING GPS to aplikacja stworzona przez firmę Mking, posiadającą w swojej ofercie wiele modeli lokalizatorów GPS. Umożliwia zarządzanie urządzeniami i jest dostępna jest wyłącznie na smartfony. Widok mapy w MKING GPS znajduje się na rys. 2.1 Chcąc utworzyć bardziej uniwersalny program, musiałby on być dostępny również na komputerach stacjonarnych czy też laptopach. Kolejnym przykładem jest platforma iTrackLive, będąca rozbudowaną aplikacją przeglądarkową - rys. 2.2. Niemniej jednak, liczba ikon i przycisków może przytłaczać zwykłego użytkownika. Aby ułatwić klientom użytkowanie aplikacji, będzie ona prostsza w obsłudze, z mniejszą ilością funkcji, lecz ze schudniejszym interfejsem.



Rysunek 2.1: Zrzut ekranu przedstawiający wygląd aplikacji MKING GPS.



Rysunek 2.2: Zrzut ekranu przedstawiający wygląd aplikacji iTrackLive.

# Rozdział 3

## Wymagania i narzędzia

Po analizie tematu zdecydowano, że zostanie stworzona aplikacja przeglądarkowa. Będzie się ona składała z trzech warstw: backend, frontend i baza danych. Backend to część funkcjonalna programu, w niej odbywają się obliczenia i logika programu. Frontend odpowiada za interfejs - obsługę formularzy, przycisków, zakładek. Trzecią częścią jest baza danych, która przechowuje dane użytkowników i stworzonych przez nich obiektów. Lokalizatorem, który został wybrany do współpracy z programem, jest model Mking MK07A. Spełnia on wymagania urządzenia, które zostały opisane podczas analizy. Ponadto, jest dostępny do kupienia na wielu stronach internetowych. Dodatkowym atutem jest jego akumulator, którego pojemność wynosi 10000 mAh, dzięki czemu nie wymaga częstego ładowania. Został on zakupiony, wraz z kartą SIM, aby możliwe było uruchomienie i skonfigurowanie urządzenia.

Tworzona aplikacja ma ułatwić użytkownikowi zarządzanie lokalizatorami, kierowcami i pojazdami jednocześnie oraz pozwalać na wyświetlanie tras poszczególnych kierowców i pojazdów. Wymagania aplikacji należy podzielić na funkcjonalne i niefunkcjonalne. Przyjmyjmy się w pierwszej kolejności wymaganiom funkcjonalnym. Zdecydowano się na model przypadków użycia, które zostały poniżej przedstawione.

### 3.1 Wymagania funkcjonalne

Dla niezalogowanego użytkownika przewidziano dwa przypadki użycia (przedstawia je diagram widoczny na rys 3.1). Pierwszy to rejestracja - gdy użytkownik wchodzi na stronę aplikacji, może się zarejestrować, czyli stworzyć bezpłatne konto, które będzie przypisane do podanego przez niego w formularzu rejestracji adresu email. Drugą opcją jest logowanie. W przypadku uprzedniej rejestracji w systemie, użytkownik może zalogować się na istniejące konto.

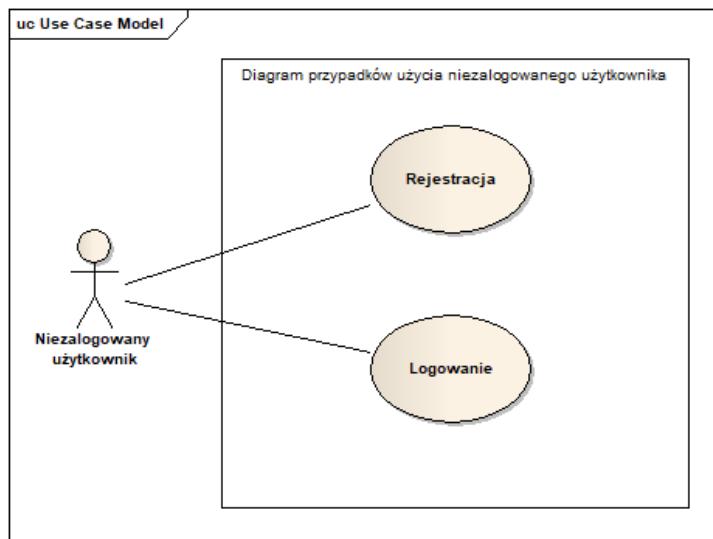
Przypadków dla zalogowanego użytkownika jest nieco więcej (przedstawia je diagram widoczny na rys 3.2) . Dodanie obiektu składa się z trzech przypadków użycia. Jednym z nich jest dodanie kierowcy, w którym użytkownik ma możliwość dodania kierowcy do swojego konta, podając jego imię oraz nazwisko. Kolejny przypadek to dodanie pojazdu. Pozwala zapisanie w systemie pojazdu poprzez podanie jego marki, modelu, numeru rejestracyjnego oraz numeru VIN. Dodanie lokalizatora to ostatni z przypadków wchodzących w skład dodania obiektu. W tym wypadku użytkownik może dodać posiadane lokalizatory do swojego konta, podając ich nazwę, numer seryjny oraz typ. Poza obiektem, użytkownik może również dodać powiązanie. Jednym z nich jest kierowca pojazdu - ten przypadek umożliwia powiązanie kierowcy z pojazdem w podanym przez użytkownika okresie czasu. Drugie powiązanie to lokalizator w pojeździe. Działa na podobnej zasadzie, umożliwia powiązanie lokalizatora z pojazdem w danym okresie czasu. Użytkownik będzie w stanie również wyświetlać trasę. Do wyboru jest trasa przebyta przez konkrentego kierowcę, pojazd lub lokalizator. Każda z trzech opcji umożliwia zdefiniowanie okresu czasu, z którego aplikacja ma wyświetlić trasę. Kolejnym przypadkiem jest usunięcie obiektu. Dostępna jest opcja usunięcia kierowcy, pojazdu lub lokalizatora - każda z nich kasuje dany obiekt z systemu. Powiązania również można usunąć, kierowcę pojazdu lub lokalizator w pojeździe, czego następstwem jest brak istnienia tego konkretnego powiązania w aplikacji.

## 3.2 Wymagania niefunkcjonalne

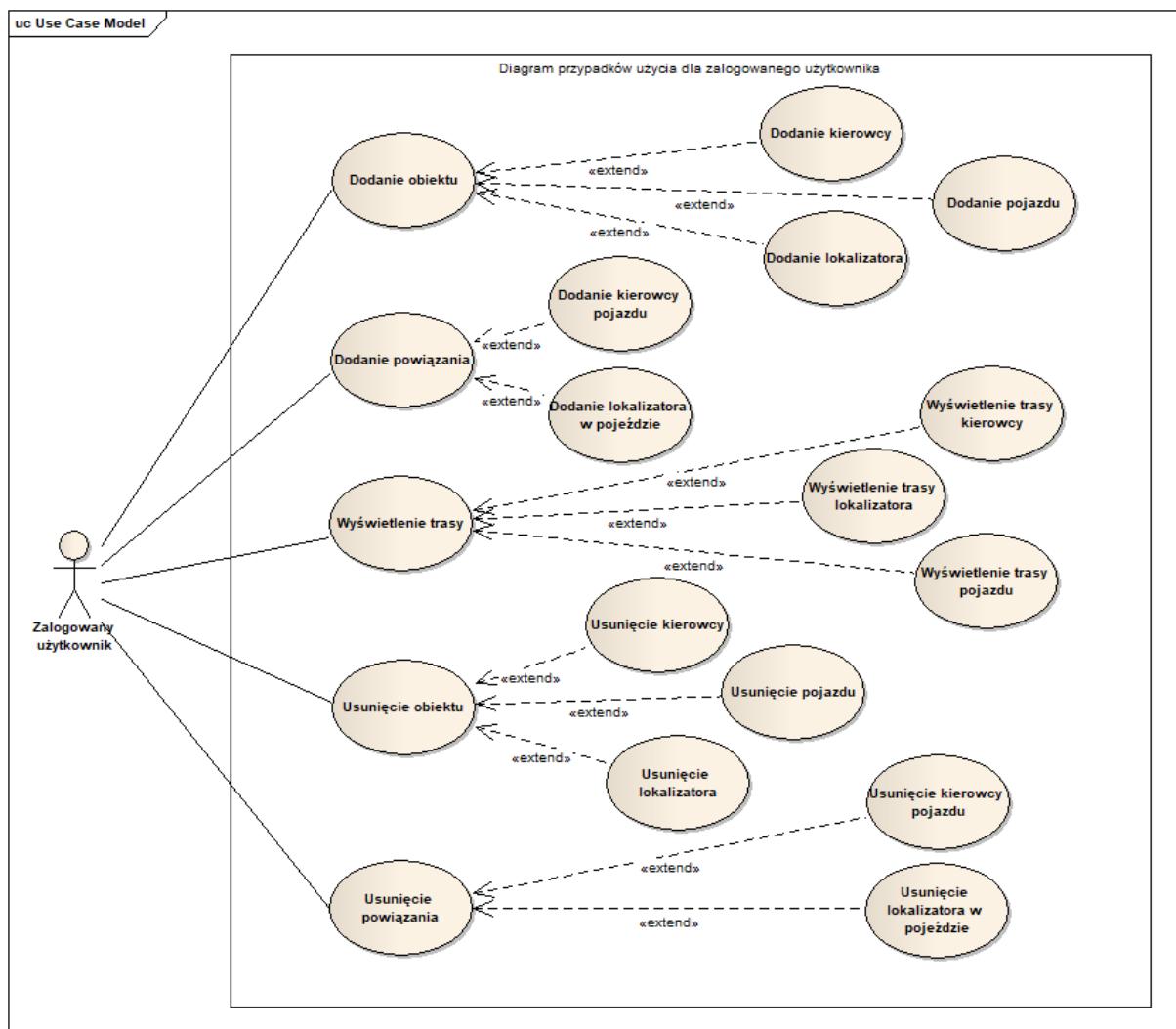
Wymagania niefunkcjonalne definiują w jaki sposób aplikacja powinna działać. Dążąc do jak największej liczby użytkowników, ustalono, iż jednym z tego typu wymagań jest prosta obsługa. Oznacza to przede wszystkim utworzenie intuicyjnego interfejsu. Dostęp do każdej funkcji programu powinien być możliwy przy wykonaniu minimalnej ilości kliknięć myszy (lub ekranu w przypadku urządzeń dotykowych), nieprzekraczającej trzech. Kolejnym założonym wymaganiem są niskie wymagania systemowe - aplikacja powinna być dostępna dla urządzeń posiadających powszechnie używane przeglądarki internetowe, takie jak Chrome, Firefox, Opera czy Safari. Ostatnim wymaganiem niefunkcjonalnym jest skalowalność interfejsu użytkownika. Wymagane jest, aby program był dostosowany do różnej wielkości oraz rozdzielczości ekranów. Użytkowanie aplikacji powinno być możliwe na najpopularniejszych smartfonach, jak na przykład Samsung S24 lub Iphone 15 od Apple.

## 3.3 Stos technologiczny

Przed rozpoczęciem tworzenia programu, wybrano odpowiednie języki programowania, które umożliwiają budowę aplikacji przeglądarkowej. Zdecydowano się także na biblioteki



Rysunek 3.1: Diagram przypadków użycia niezalogowanego użytkownika.



Rysunek 3.2: Diagram przypadków użycia zalogowanego użytkownika.

i platformy programistyczne, które ułatwiają pisanie kodu i komunikacje między różnymi częściami aplikacji.

### 3.3.1 Java

Ze względu na liczne biblioteki usprawniające proces rozwijania oprogramowania, część funkcjonalną programu postanowiono napisać w języku Java. Jest to jeden z najpopularniejszych wysokopoziomowych języków programowania. Stworzony w 1995 roku, jest nadal rozwijany i powszechnie używany. Jego cechami są przede wszystkim: obiektowość, dobrze rozwinięta biblioteka standardowa, niezliczona ilość dodatkowych bibliotek. Posiada on również Garbage Collector, odpowiadający za zarządzanie pamięcią, co zwiększa bezpieczeństwo języka. Więcej informacji na ten temat można znaleźć w książce "Java. Efektywne programowanie" autora Joshua Bloch [1]. Jest to częsty wybór w przypadku tworzenia aplikacji przeglądarkowych, ze względu na istniejące platformy programistyczne oraz biblioteki, będące idealnym rozwiązaniami do tego typu programów.

### 3.3.2 Spring Boot

W celu konfiguracji aplikacji przeglądarkowej została wybrana platforma programistyczna, jaką jest Spring Boot. Jest popularnym wyborem dla języka Java, gdyż w znaczny sposób ułatwia konfigurację tego typu aplikacji, jak również późniejsze jej rozwijanie pod względem tworzenia REST API czy połączenie programu z bazą danych. Konfiguracja odbywa się w dużej mierze automatycznie, wkład programisty opiera się przede wszystkim na dodaniu odpowiednich zależności. Charakterystyczne dla Spring Boot są adnotacje, które są poprzedzone znakiem "@". To dzięki nim Spring Boot potrafi interpretować napisany kod, aby następnie wykonywać pewne operacje automatycznie. Więcej informacji na ten temat znajduje się na oficjalnej stronie producenta [2].

### 3.3.3 JavaScript

To również bardzo powszechny język programowania, natomiast w odróżnieniu do Javy, odpowiada on przede wszystkim za interfejs użytkownika. Ważną jego cechą, wykorzystywaną w aplikacjach przeglądarkowych, jest obsługa asynchroniczności. Jest to spowodowane wysyłaniem zapytań HTTP, które trafiają do Javy, a odpowiedzi nie zawsze są natychmiastowe. Więcej opisuje David Flanagan w swojej książce "JavaScript. The Definitive Guide" [3].

### 3.3.4 TypeScript

Nie jest uznawany za osobny język, a za nadzbiór JavaScriptu. Rozszerza jego możliwości, przede wszystkim dzięki typowaniu, które polega na tym, iż zmiennym można przy-

oisywać konkretny, zdefiniowany typ. Dotyczy to również danych zwracane przez funkcje. Znacznie zwiększa to czytelność kodu oraz pozwala uniknąć przypisywania niewłaściwych typów do zmiennych. Informacje na temat tego języka można znaleźć w oficjalnej dokumentacji TypeScript [10].

### 3.3.5 React

React jest biblioteką do tworzenia interfejsu użytkownika. Działa z językiem JavaScript, jak również TypeScript. Został on stworzony, aby budowane aplikacje przeglądarkowe były dynamiczne - to znaczy aby nie potrzebowały odświeżania strony, aby aktualizować jej widok. React polega na tworzeniu komponentów, które odpowiadają za renderowanie określonej części interfejsu. Charakterystycznymi elementami tej biblioteki są Hooki, czyli funkcje umożliwiające zarządzanie stanem komponentów. Aby dowiedzieć się więcej, warto wejść na oficjalną stronę internetową React [9].

### 3.3.6 PostgreSQL

Zaawansowany system do zarządzania bazą danych, zgodny ze standardem SQL, pozwalający również na dostosowanie do indywidualnych potrzeb, gdyż posiada na przykład możliwość tworzenia własnych typów danych. Często wykorzystywany w aplikacjach przeglądarkowych, ze względu na jego wysoką wydajność oraz skalowalność. Dokumentację do PostgreSQL można pobrać z oficjalnej strony [8].

### 3.3.7 PostGIS

Jest to rozszerzenie do PostgreSQL, które wspiera obsługę danych geograficznych i przestrzennych. Daje możliwość przechowywania danych przede wszystkim o punktach, ale również o liniach, wielokątach. Rozszerzenie to jest używane w przypadku programów związanych z obsługą GPS oraz map, także idealnie pasuje do niniejszej pracy. Dokumentacja dostępna na oficjalnej stronie PostGIS [7].

### 3.3.8 Hibernate

Bardzo często wybierana platforma programistyczna w celu automatyzacji mapowania między obiektami w języku Java, a tabelami w bazie danych. Umożliwia to pracę nad obiektami, bez konieczności wykonywania skomplikowanych zapytań SQL. Więcej informacji znajduje się na oficjalnej stronie internetowej Hibernate [5].

### 3.3.9 Git

Ostatnim elementem jest system kontroli wersji. Dzięki temu postępy w pracy będą zabezpieczone. Niemniej jednak, to nie jedyny atut. W przypadku błędów aplikacji spowodowanych zmianami w kodzie źródłowym, zlokalizowanie ich przyczyny będzie znacznie ułatwione. Wybrano system Git, gdyż jest on najpopularniejszą opcją, a co za tym idzie, jest obsługiwany przez niemal każde oprogramowanie programistyczne. Dokumentacja do tego systemu widnieje na oficjalnej stronie Git [4].

## 3.4 Oprogramowanie do edycji kodu

Aby przystąpić do pisania kodu, należało wybrać i zainstalować odpowiednie programy, które to wspierają odpowiednie języki, biblioteki czy platformy programistyczne.

### 3.4.1 IntelliJ Idea

Do części funkcjonalnej został wykorzystany IntelliJ Idea wydawcy JetBrains. Ten program jest zaawansowanym, wieloplatformowym środowiskiem programistycznym, posiadającym dużą ilość wbudowanych narzędzi, jak również obsługującym wiele bibliotek i platform programistycznych, takich jak Spring Boot. Hibernate oraz JPA również są wspierane przez IntelliJ. Są to narzędzia, które potrafią powiązać kod aplikacji z bazą danych, co znacznie uprości rozwiązywanie programu.

### 3.4.2 Visual Studio Code

Przechodząc do interfejsu użytkownika, a zatem do kodu w języku TypeScript z zastosowaniem biblioteki React, wybrano Microsoft Visual Studio Code, który jest powszechnie używany przez programistów w tym celu, ze względu na odpowiednie wsparcie języków i bibliotek typowych do tworzenia wizualnej części aplikacji.

### 3.4.3 PgAdmin

W kontekście bazy danych, potrzebne było oprogramowanie, które będzie w stanie obsługiwać zapytania kierowane do bazy danych - na początku do stworzenia bazy, poszczególnych tabel, a w przyszłości do eliminowania ewentualnych błędów oraz do testowania aplikacji. PgAdmin umożliwia te operacje, jak również pozwala na stworzenie diagramu bazy danych.

### 3.4.4 GitHub

Należało również dokonać wyboru platformy hostingowej, która będzie przechowywała repozytorium Git. W tym celu skorzystano z GitHuba, będącego jedną z najbardziej powiększych opcji. Jest on także wspierany przez większość środowisk programistycznych.

## 3.5 Metodyka pracy

Do usystematyzowania i organizacji prac nad projektem wybrano model kaskadowy (ang. waterfall). Polega on na sekwencyjnym działaniu nad kolejnymi etapami. Zdecydowano się na niego, ponieważ w opinii autora, w przypadku jednoosobowego projektu, umożliwia on zoptymalizowanie wykorzystywanej czasu. Zrezygnowano z innych opcji metodyki pracy, wśród których przykładem jest Scrum. Sprawdza się on przede wszystkim w przypadku pracy w zespole, zatem uznano, że nie nadaje się do tego projektu.



# Rozdział 4

## Specyfikacja zewnętrzna

Specyfikacja zewnętrzna opisuje interakcje użytkownika z systemem, jak również jego wygląd i sposób obsługi. Niniejszy rozdział zawiera także informacje na temat wymagań sprzętowych, które należy spełnić, aby aplikacja działała poprawnie.

### 4.1 Wymagania sprzętowe i programowe

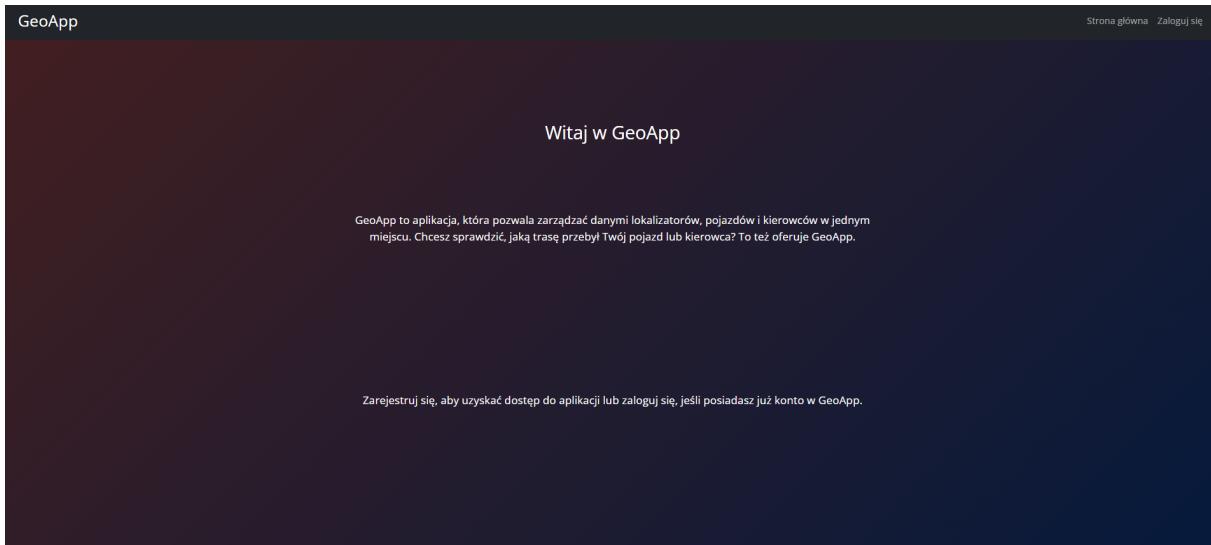
Aplikacja do działania potrzebuje serwera - komputera, na którym skompilowany z kodu źródłowego program, jak również baza danych, będą uruchomione. Serwer musi posiadać dostęp do internetu oraz jego adres musi być publicznie dostępny. Dzięki temu ma on możliwość obsługiwania żądań HTTP wysyłanych przez użytkowników.

W przypadku działającego serwera, można do użytkowania aplikacji. W celu uzyskania do niej dostępu, użytkownik musi mieć zainstalowaną przeglądarkę na urządzeniu (np. komputerze stacjonarnym, laptopie, tablecie czy smartfonie) oraz połączenia z internetem. Po spełnieniu powyższych wymagań, w pasek adresu URL w przeglądarce należy wpisać adres serwera oraz port, na którym uruchomiona jest aplikacja. Można również uprosić użytkownikom uruchomienie aplikacji poprzez uruchomienie serwera na konkretnej domenie DNS, którą wystarczy wpisać w przeglądarce, natomiast nie zostało to zaimplementowane w trakcie procesu tworzenia systemu.

### 4.2 Kategorie użytkowników

Wyróżniamy dwa rodzaje użytkowników:

- niezalogowany użytkownik
- zalogowany użytkownik



Rysunek 4.1: Zrzut ekranu przedstawiający stronę główną.

Z niezalogowanym użytkownikiem mamy do czynienia wówczas, gdy nastąpi włączenie aplikacji lub po wylogowaniu się ze swojego konta podczas użytkowania. Funkcje, które ma on do dyspozycji, są mocno ograniczone. Użytkownik ma wtedy dostęp do:

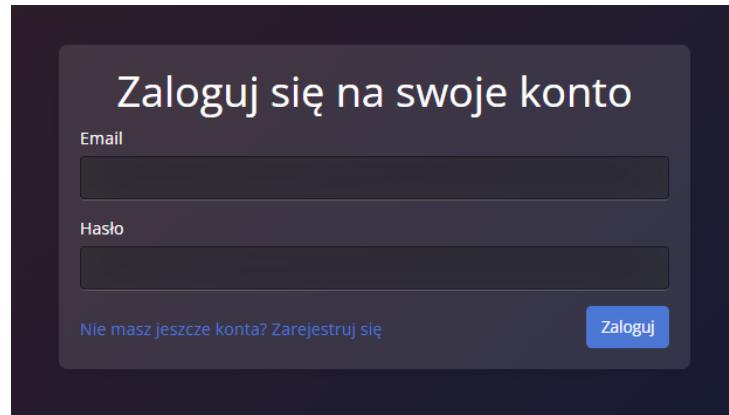
- strony głównej - tylko wyświetla tekst
- formularza rejestracji - pozwala utworzyć konto użytkownika
- formularza logowania - umożliwia zalogowanie się na istniejące w systemie konto

Zalogowany użytkownik uzyskuje dostęp do właściwych funkcji systemu, dzięki którym może zarządzać lokalizatorami, jak również kierowcami i pojazdami.

### 4.3 Sposób obsługi

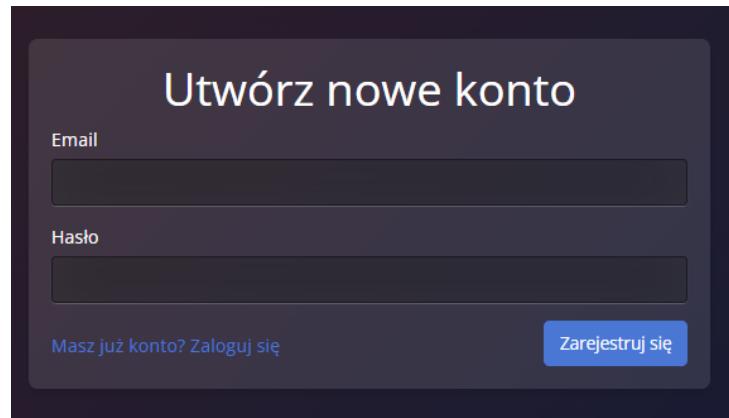
Po włączeniu aplikacji ukazuje się strona główna. Widoczny na niej tekst zachęca do skorzystania z wszystkich funkcji systemu. Jej wygląd jest widoczny na rys. 4.1. Na górze strony znajduje się pasek nawigacji, który towarzyszy użytkownikowi przez cały czas użytkowania programu, lecz różni się w zależności od tego, czy użytkownik jest niezalogowany (pasek wtedy przybiera formę jak na rys. 4.1, czy zalogowany (w tym przypadku pasek wygląda jak na rys. 4.4). Po kliknięciu w poszczególną opcję na pasku, odpowiadającą jej zakładka jest otwierana. Aby przejść do formularzy logowania i rejestracji, należy wybrać opcję "Zaloguj się".

W pierwszej kolejności ukazuje się formularz logowania (rys. 4.2). Jest to celowy zabieg, ponieważ użytkownik, po założeniu konta, z każdym kolejnym uruchomieniem aplikacji będzie się logował. Wynika z tego, iż liczba logowań będzie wyższa niż liczba rejestracji,



The screenshot shows a dark-themed login form. At the top, the title 'Zaloguj się na swoje konto' is displayed in white. Below the title are two input fields: 'Email' and 'Hasło', each with a placeholder text 'Email' and 'Hasło' respectively. At the bottom left of the form is a blue link 'Nie masz jeszcze konta? Zarejestruj się'. On the right side, there is a blue button labeled 'Zaloguj'.

Rysunek 4.2: Zrzut ekranu przedstawiający formularz logowania.

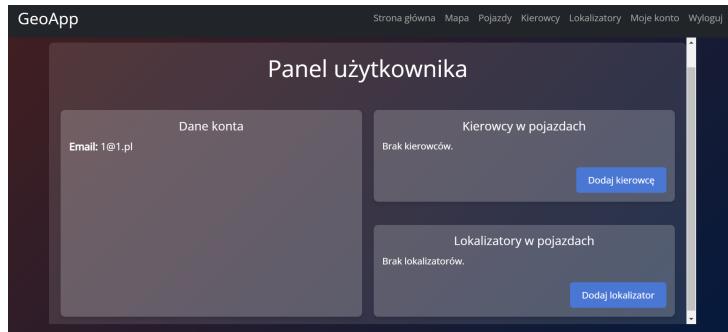


The screenshot shows a dark-themed registration form. At the top, the title 'Utwórz nowe konto' is displayed in white. Below the title are two input fields: 'Email' and 'Hasło', each with a placeholder text 'Email' and 'Hasło' respectively. At the bottom left of the form is a blue link 'Masz już konto? Zaloguj się'. On the right side, there is a blue button labeled 'Zarejestruj się'.

Rysunek 4.3: Zrzut ekranu przedstawiający formularz rejestracji.

a zatem będzie to częstsza operacja wykonywana przez użytkowników. W celu otwarcia formularza rejestracji, należy kliknąć w tekst w lewym dolnym rogu formularza logowania. Jego treść brzmi: „Nie masz jeszcze konta? Zarejestruj się” - rys. 4.2. Jest to link, który umożliwia zmianę formularza z opcji do zalogowania się, na opcję do utworzenia konta. Aby stworzyć konto użytkownika, należy wpisać adres email, z którym będzie owe konto powiązane. Kolejnym etapem jest ustalenie hasła, którym następnie wypełnia się drugie pole. W celach bezpieczeństwa, hasło powinno być trudne do rozszyfrowania przez innych ludzi. Niemniej jednak, nie ma możliwości odzyskania zapomnianego hasła, o czym warto pamiętać podczas korzystania z aplikacji. Aby dokonać procesu tworzenia konta, należy nacisnąć niebieski przycisk w prawym dolnym rogu. Widok formularza rejestracji znajduje się na rys. 4.3.

Do formularza logowania można dostać się na dwa sposoby. Jeden został wymieniony powyżej, drugim jest kliknięcie w tekst znajdujący się w lewym dolnym narożniku formularza rejestracji, brzmiący: „Masz już konto? Zaloguj się”. Dzięki temu w prosty sposób można przejść do zalogowania się, bezpośrednio po stworzeniu konta. Formularz logowania zawiera dwa pola, jedno służy do wpisania adresu email, z którym powiązane jest konto.



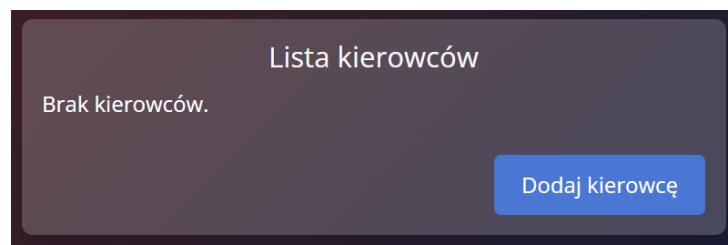
Rysunek 4.4: Zrzut ekranu przedstawiający zakładkę Moje konto.

W drugie pole należy wpisać hasło, które zostało ustalone podczas rejestracji. Zamiast znaków wpisywanych w to pole, pojawiają się kropki. Jest to mechanizm służący podniesieniu bezpieczeństwa, aby żadna dodatkowa osoba będąca w bliskiej okolicy użytkownika nie była w stanie przeczytać hasła, które jest wprowadzane. Po wypełnieniu obydwu pól wystarczy kliknąć niebieski przycisk pod formularzem.

W przypadku poprawnego wprowadzenia emailu oraz hasła, na ekranie ukazuje się panel użytkownika (rys.4.4). Służy on przede wszystkim do dodawania powiązań między kierowcami, pojazdami i lokalizatorami, natomiast w pierwszej kolejności trzeba dodać poszczególne obiekty, jakimi są kierowca, pojazd czy lokalizator. Niemniej jednak, założono, że w trakcie używania aplikacji, użytkownicy nie często będą dodawać np. nowe pojazdy, stąd pierwszą zakładką, jaka pokazuje się po zalogowaniu, jest panel użytkownika.

Aby dodać kierowcę, w pierwszej kolejności należy przejść do zakładki nazwanej „Kierowcy”. Podstawowym elementem tej części aplikacji jest lista kierowców. W przypadku gdy lista jest pusta, wyświetlany jest tekst „Brak kierowców.”, widoczny na rys. 4.5. Pod listą lub napisem świadczącym o pustej liście znajduje się przycisk służący do dodania nowego kierowcy do listy. Po naciśnięciu przycisku otworzy się formularz - jego wygląd jest dostępny na rys. 4.6. Należy wypełnić jego pola, którymi są imię oraz nazwisko kierowcy. Przycisk oznaczony napisem „Dodaj kierowcę” pozwala dokończyć proces i zapisać wprowadzone dane w systemie. Formularz zostanie zamknięty, a na liście pojawi się nowa pozycja - przykład widoku z przykładowym kierowcą przedstawia rys. 4.7. W przypadku kliknięcia czerwonego przycisku „Anuluj”, proces dodawania kierowcy zostanie wycofany, a aplikacja wróci do widoku zakładki z listą kierowców. Będąc tej w zakładce, użytkownik ma również możliwość usunięcia wybranego kierowcy. Służy do tego przycisk „Usuń”, znajdujący się po prawej stronie od każdej pozycji na liście.

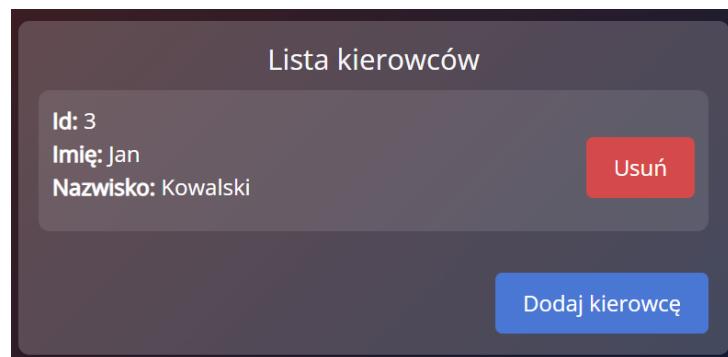
Zakładką o analogicznym działaniu jest zakładka „Pojazdy”. Posiada listę samochodów zalogowanego użytkownika oraz przycisk pozwalający dodać do niej nową pozycję - wygląd z pustą listą znajduje się na rys. 4.8. Formularz w tym przypadku posiada cztery



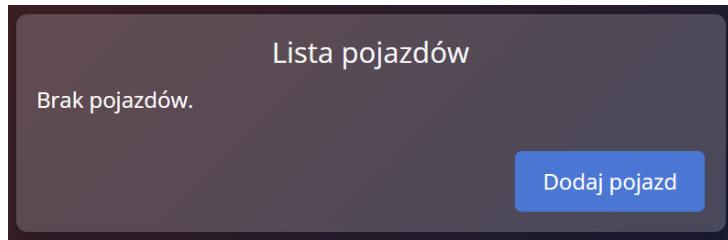
Rysunek 4.5: Zrzut ekranu przedstawiający zakładkę Kierowcy.



Rysunek 4.6: Zrzut ekranu przedstawiający formularz dodawania kierowcy.



Rysunek 4.7: Zrzut ekranu przedstawiający zakładkę Kierowcy z jednym kierowcą na liście.



Rysunek 4.8: Zrzut ekranu przedstawiający zakładkę Pojazdy.

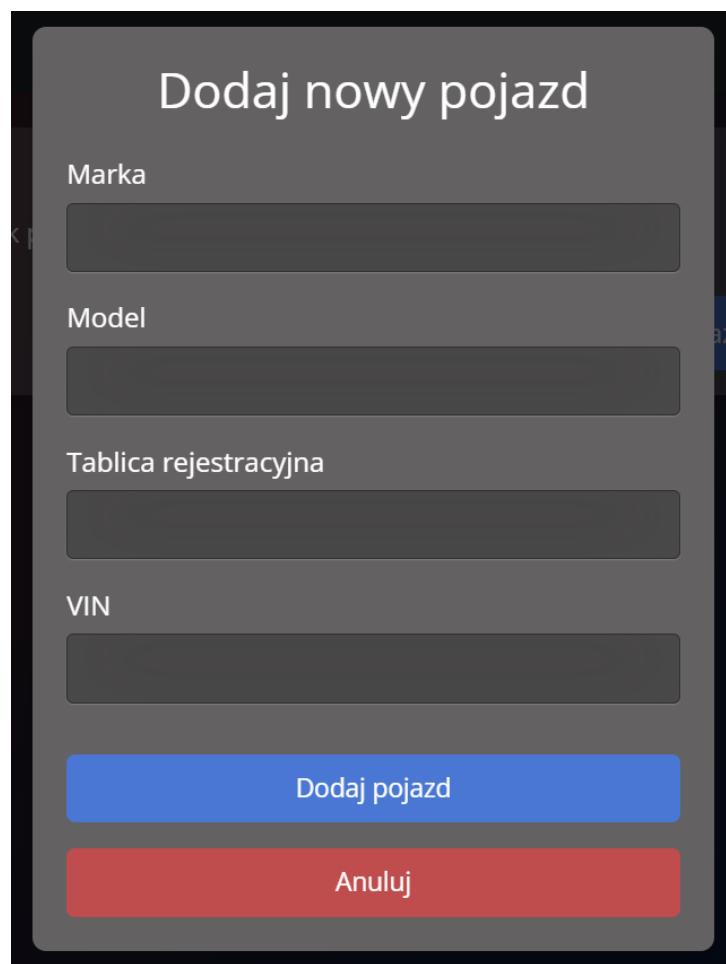
pola: marka, model, tablica rejestracyjna oraz VIN. Pierwsze dwa służą przede wszystkim zwiększeniu przejrzystości listy dla użytkownika, natomiast trzecie i czwarte pole umożliwiają rozróżnienie poszczególnych samochodów, podczas gdy są one tej samej marki oraz tego samego modelu. Rys. 4.9 przedstawia formularz dodawania pojazdu. Usuwanie elementów z listy jest dostępne również za pomocą przycisku znajdującego się po prawej stronie od każdego pojazdu.

„Lokalizatory” to trzecia zakładka o podobnej zasadzie działania. Zrzut ekranu z jej wyglądem jest dostępny na rys. 4.10. Lista zawiera lokalizatory dodane przez użytkownika. Po kliknięciu w przycisk zatytułowany „Dodaj lokalizator”, otworzy się formularz, tym razem posiadający trzy pola do wypełnienia (rys.4.11). Nazwa jest dowolna, ma na celu ułatwić użytkownikowi znalezienie pożądanego lokalizatora na liście, numer seryjny jest polem identyfikacyjnym, które jest niepowtarzalne dla każdego urządzenia, natomiast w pole typ należy wpisać model urządzenia. W tym formularzu pojawia się również dodatkowy przycisk z napisem "Instrukcja - konfiguracja lokalizatora". Otwiera on dodatkowe okno z punktami, które trzeba wykonać, aby skonfigurować urządzenie do działania z aplikacją (rys. 4.12). Można je zamknąć przyciskiem "Anuluj". Usuwanie odbywa się za pomocą przycisku „Usuń”, widocznego na rys. 4.10.

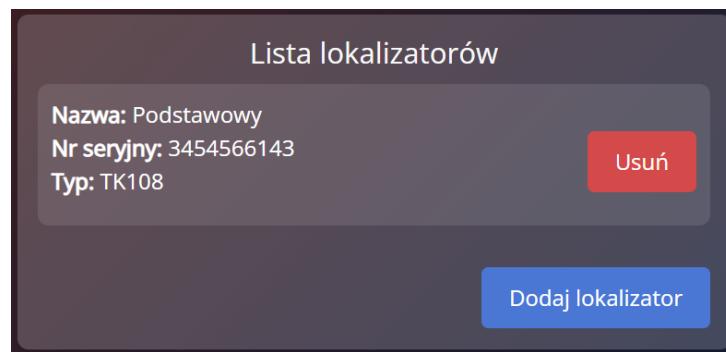
Po dodaniu kierowców, pojazdów i lokalizatorów można przejść do tworzenia powiązań między nimi. W tym celu należy otworzyć zakładkę „Moje konto”, która przedstawia panel użytkownika. Aplikacja pozwala na stworzenie dwóch rodzajów powiązań:

- kierowca pojazdu
- lokalizator w pojeździe

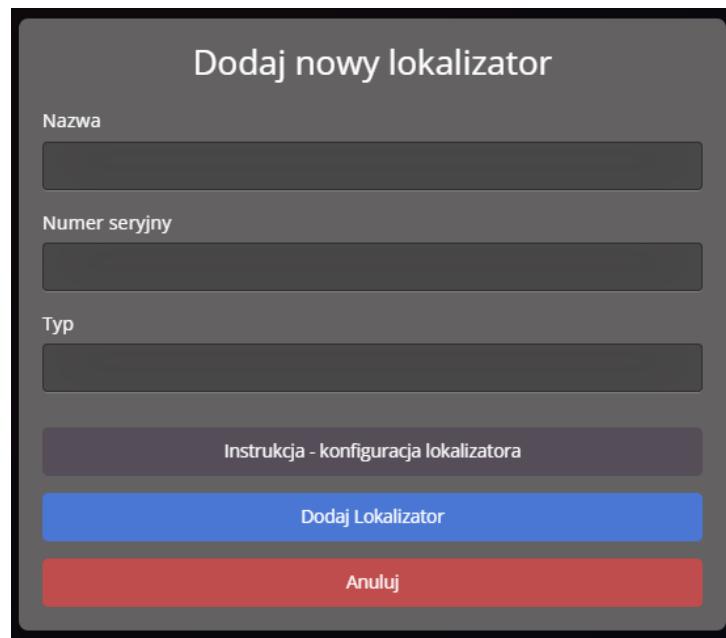
Kierowcy pojazdów, będący jednocześnie tytułem pierwszej listy w panelu użytkownika, definiują w jakim okresie czasu dany kierowca korzysta z danego pojazdu. Będzie to miało szczególne znaczenie podczas wyświetlania tras przebytych przez poszczególną osobę. W celu dodania powiązania, należy kliknąć przycisk „Dodaj kierowcę” (4.4). Aplikacja wyświetli formularz składający się z czterech pól (rys. 4.13). Pierwsze dwa są polami rozwijanymi, elementami pierwszego będą wszystkie pojazdy użytkownika, natomiast w



Rysunek 4.9: Zrzut ekranu przedstawiający formularz dodawania pojazdu.



Rysunek 4.10: Zrzut ekranu przedstawiający zakładkę Lokalizatory.



Rysunek 4.11: Zrzut ekranu przedstawiający formularz dodawania lokalizatora.



Rysunek 4.12: Zrzut ekranu przedstawiający formularz dodawania lokalizatora.

**Powiąż pojazd z kierowcą**

Wybierz pojazd

-- Wybierz pojazd --

Wybierz kierowcę

-- Wybierz kierowcę --

Data od

dd.mm.rrrr

Data do

dd.mm.rrrr

Dodaj powiązanie

Anuluj

Rysunek 4.13: Zrzut ekranu przedstawiający formularz dodawania powiązania kierowcy z pojazdem.

drugim znajdująć się będą jego kierowcy. Kolejne dwa pola to początek i koniec okresu użytkowania wybranego samochodu przez wybraną osobę. W celu wyznaczenia dat należy wybrać odpowiedni dzień z wyświetlnego kalendarza. Aby dokończyć proces tworzenia powiązania, należy kliknąć przycisk z napisem „Dodaj powiązanie”. Powiązanie zostanie zapisane do bazy danych i wyświetli się jako nowy element listy. Przycisk „Anuluj” czyści pola formularza jednocześnie go zamykając.

Drugą opcją powiązań są lokalizatory w pojazdach. Ich lista znajduje się pod listą kierowców pojazdów. Podobnie jak w przypadku pierwszego powiązania, formularz, który pozwala na dodanie nowego elementu od listy, ma cztery pola. Pierwsze pole jest niezmienne, co można zobaczyć na rys. 4.14. Jest to spowodowane faktem, iż to powiązanie również wykorzystuje pojazdy, zatem można w tym miejscu dokonać wyboru samochodu. Drugie pole uległo zmianie, jest to rozwijana lista lokalizatorów, z których należy wybrać jeden, który w danym okresie użytkownik planuje umieścić w pojeździe. Okres powiązania definiują kolejne dwa pola formularza, które również nie różnią się od pól wypełnianych

**Powiąż pojazd z lokalizatorem**

Wybierz pojazd

-- Wybierz pojazd --

Wybierz lokalizator

-- Wybierz lokalizator --

Data od

dd.mm.rrrr

Data do

dd.mm.rrrr

Dodaj powiązanie

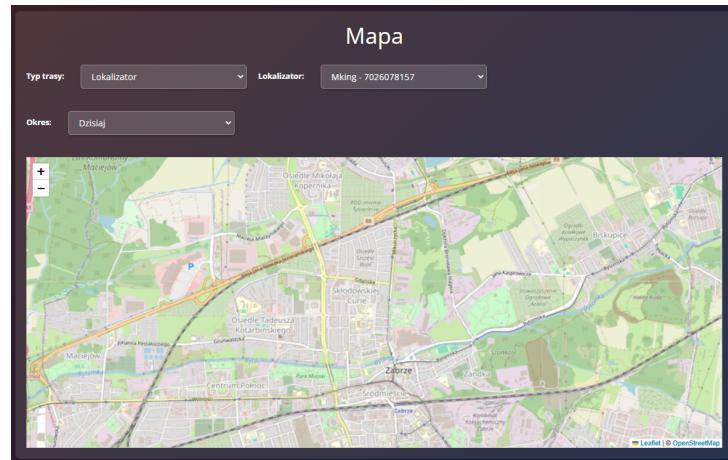
Anuluj

Rysunek 4.14: Zrzut ekranu przedstawiający formularz dodawania powiązania lokalizatora z pojazdem.

podczas dodawania powiązania kierowcy z pojazdem. Przyciski oraz ich działanie jest także analogiczne do powyżej opisanego powiązania.

Usuwanie powiązań dokonuje się w identyczny sposób, w jaki zostało opisane usuwanie pojedynczych obiektów. Niemniej jednak, ważnym aspektem usuwania kierowców, pojazdów i lokalizatorów jest fakt, iż w przypadku uczestnictwa danego obiektu w jakimkolwiek powiązaniu, niedozwolone jest jego usunięcie. Pierwszym krokiem jest wtedy wyeliminowanie powiązania, z którym związany jest wybrany przez użytkownika np. kierowca, a następnie przeprowadzenie procesu usunięcia kierowcy.

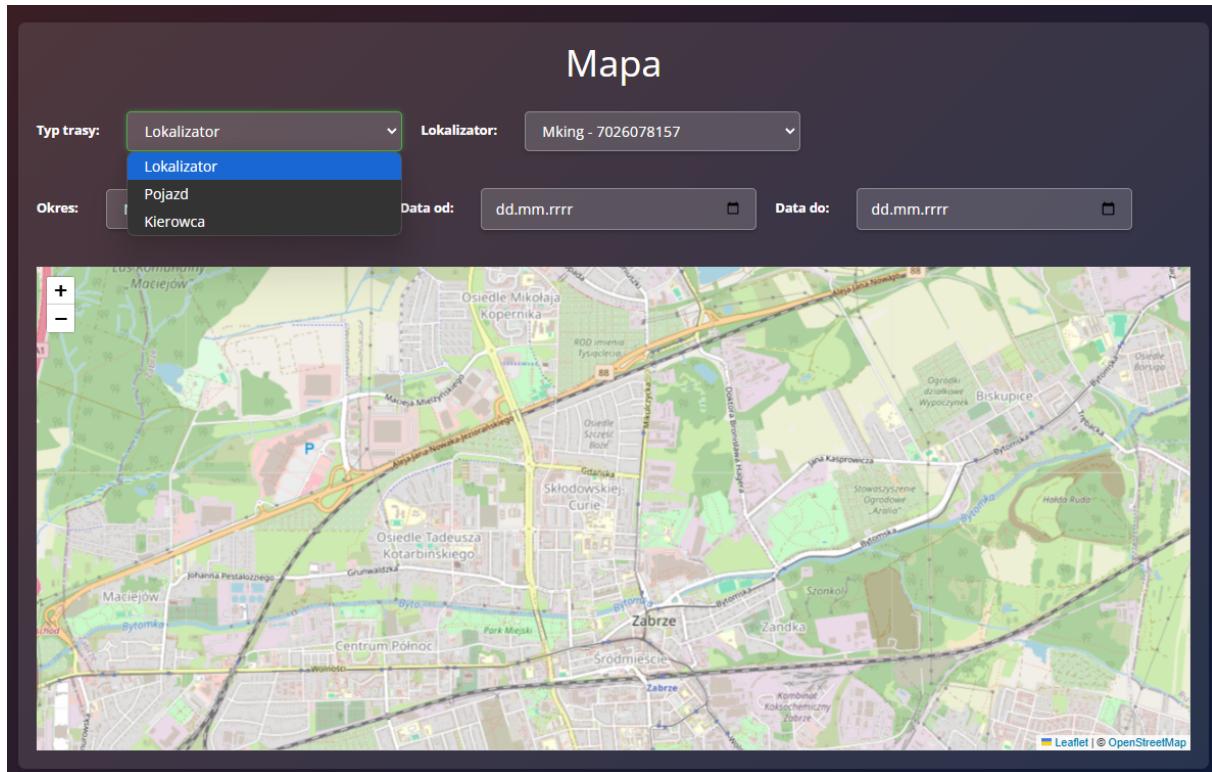
Zakładka pod tytułem „Mapa” pozwala na wyświetlenie trasy z danego okresu czasu. Po jej otwarciu oczom użytkownika ukazuje się widok z rys. 4.15. Mapę można oddalać oraz przybliżać, korzystając z kółka myszy, a także przesuwać, poruszając myszą przy wcisniętym lewym przycisku. Pierwszą trasą, która wyświetla się po otwarciu zakładki jest trasa pierwszego z listy lokalizatora z bieżącego dnia, oczywiście w przypadku gdy ten lokalizator wysyłał dane na serwer tego dnia, to znaczy nie był wyłączony, rozładowany oraz został poprawnie skonfigurowany. Rozwijana lista nazwana "Lokalizator" pozwala wybrać



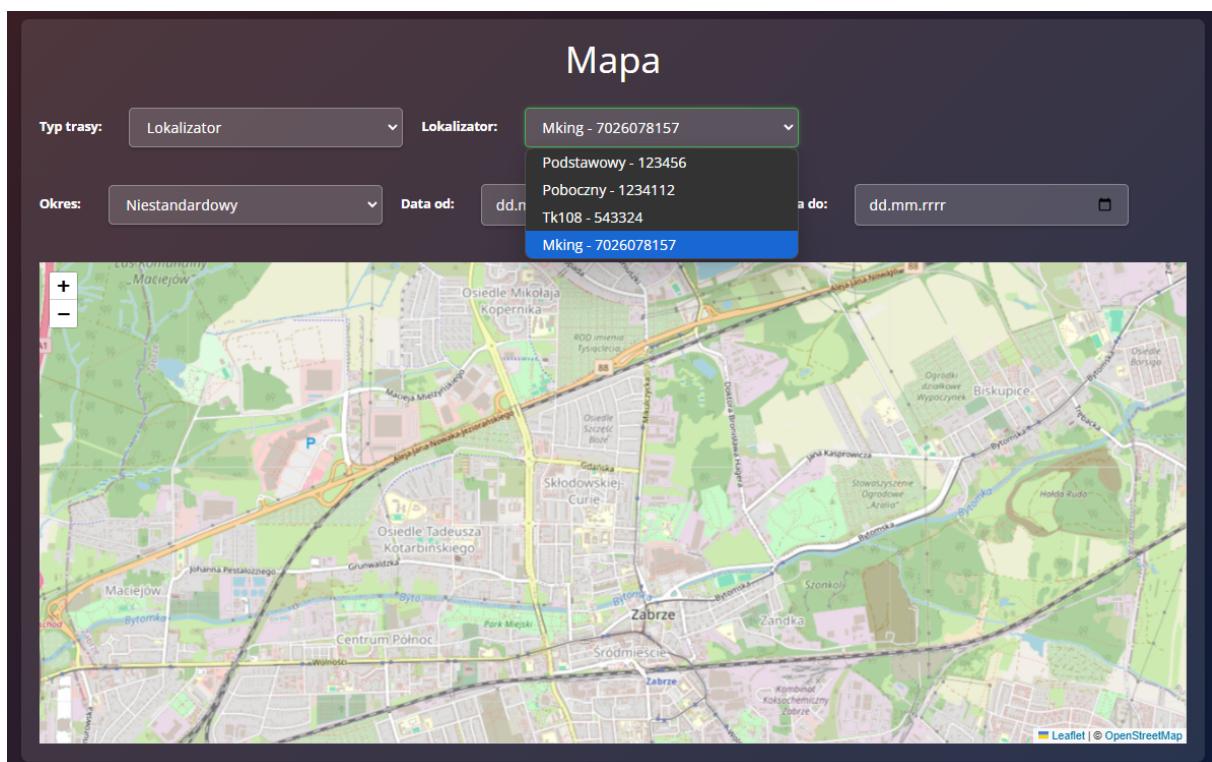
Rysunek 4.15: Zrzut ekranu przedstawiający zakładkę Mapa.

konkretne urządzenie, którego trasę chcemy zobaczyć na mapie (rys. 4.17). Aby zmienić trasę na kierowcę lub pojazd, należy rozwinąć pole opisane jako "Typ trasy". Można wybrać wtedy opcję z listy, co widać na rys. 4.16. W zależności od wybranego typu, rozwiniana lista oznaczona słowem "Lokalizator" zmienia się w listę wyboru kierowcy lub pojazdu. Ostatnią opcją do wyboru jest "Okres", czyli dzień lub zakres dat, z którego chcemy wyświetlić trasę. Zgodnie z tym, co znajduje się na rys. 4.18, można wybrać jedną z paru opcji domyślnych lub opcję "Niestandardowy", której kliknięcie skutkuje pojawiением się dodatkowych dwóch pól (rys. 4.18), w celu określenia daty początkowej i końcowej okresu do wyświetlenia trasy. Przykład ukazujący wygląd losowej trasy przedstawia rys. 4.19.

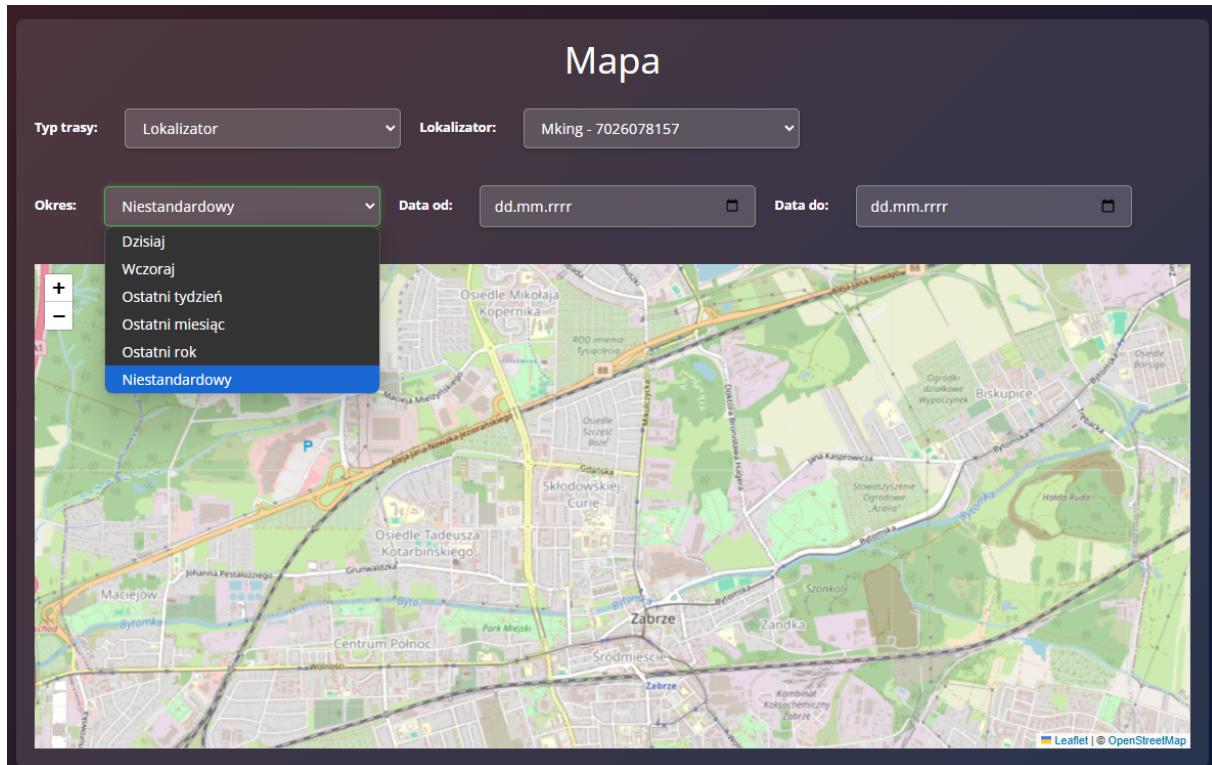
Na pasku nawigacyjnym zalogowanego użytkownika widnieje jeszcze jedna zakładka - „Wyloguj się” (rys. 4.4). Jest to tak naprawdę przycisk umożliwiający wylogowanie się z konta użytkownika. Po jego kliknięciu, aplikacja nas wylogowuje i przenosi do strony głównej. Pojawia się wtedy pasek nawigacji widoczny na rys. 4.1, co sprawia, iż w celu dostępu do funkcji zalogowanego użytkownika, należy ponownie się zalogować zgodnie z wyżej opisanymi krokami.



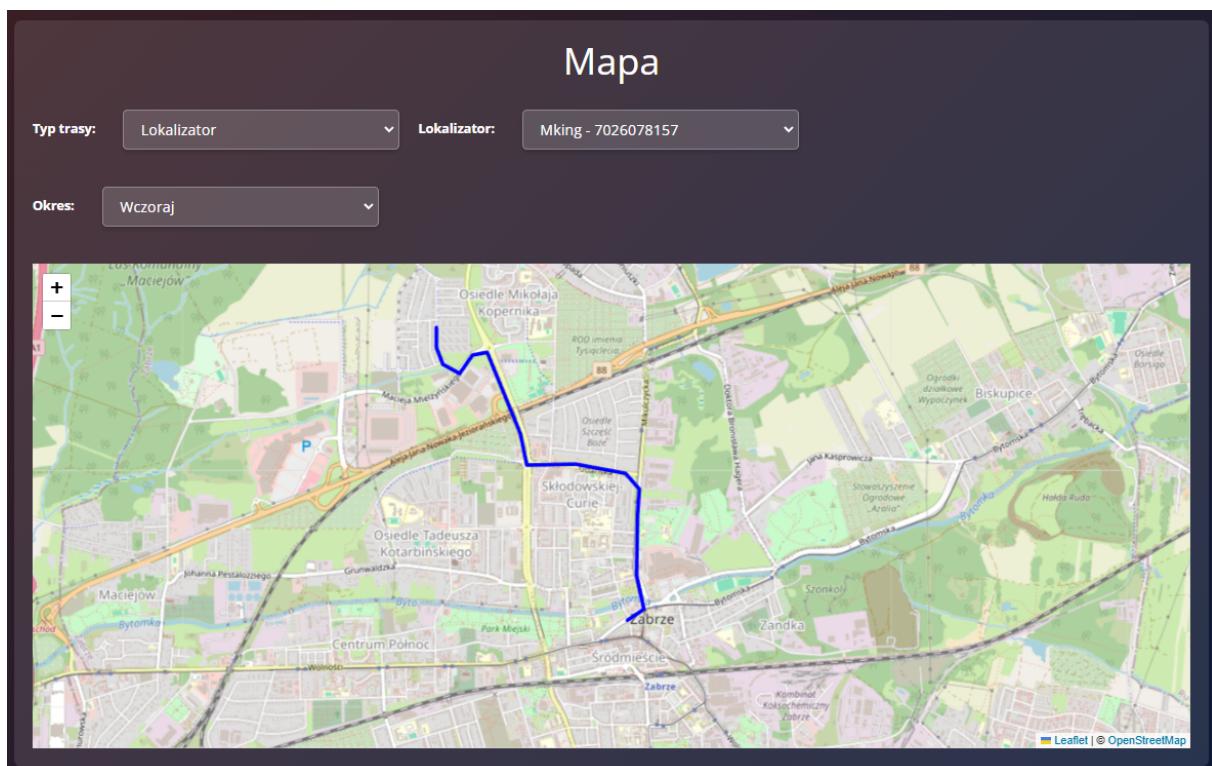
Rysunek 4.16: Zrzut ekranu przedstawiający formularz dodawania trasy kierowcy.



Rysunek 4.17: Zrzut ekranu przedstawiający formularz dodawania trasy pojazdu.



Rysunek 4.18: Zrzut ekranu przedstawiający mapę z jedną trasą.



Rysunek 4.19: Zrzut ekranu przedstawiający mapę z jedną trasą.



# Rozdział 5

## Specyfikacja wewnętrzna

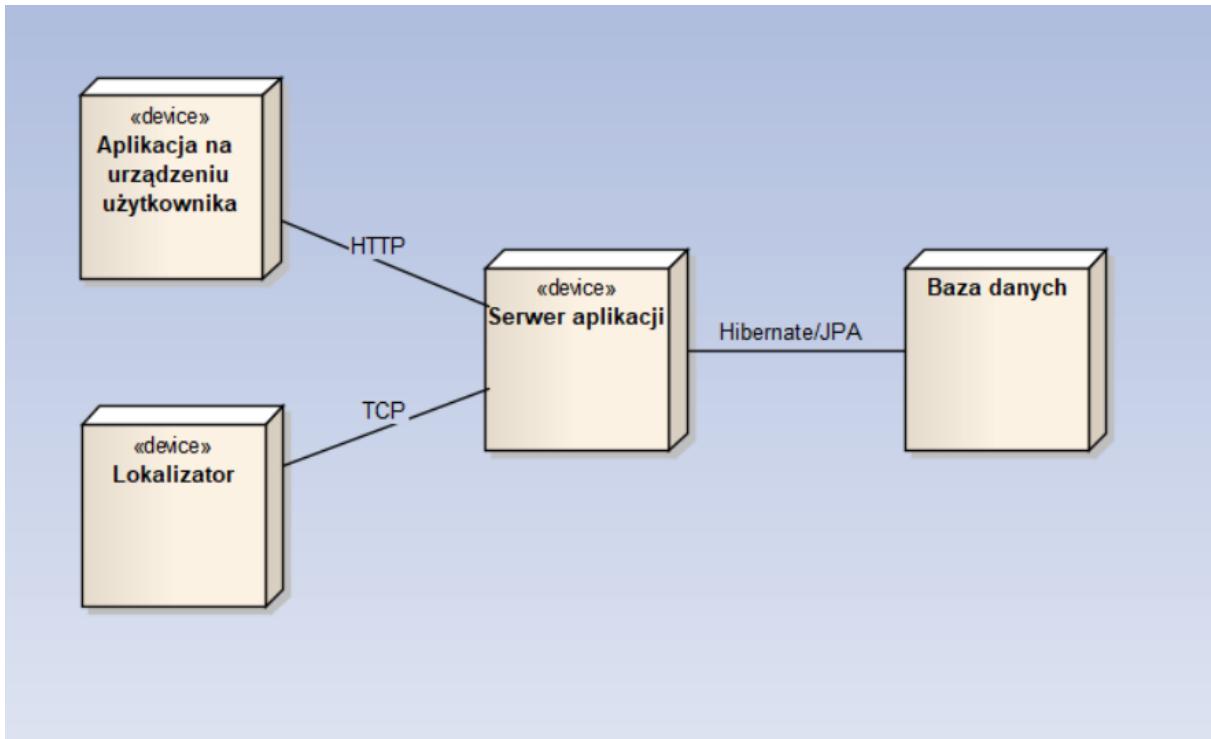
Specyfikacja wewnętrzna jest ważnym elementem dokumentacji technicznej. Pozwala na zrozumienie funkcjonowania aplikacji, dzięki zawartym opisom struktur danych, architektury systemu, jak również tłumaczeniom kodu źródłowego, popartym jego fragmentami.

### 5.1 Przedstawienie architektury systemu

Podczas pisania kodu źródłowego kierowano się podejściem database-first, co oznacza rozpoczęcie prac od zaprojektowania bazy danych. Pierwszym krokiem było przygotowanie schematu, który znajduje się na rys. 5.7, który jest opisany w kolejnym podrozdziale. W oparciu o schemat utworzono bazę danych, a następnie odpowiadające tabelom klasy encji w języku Java. Po zakończeniu poprzednich kroków, rozpoczęto pisanie kodu interfejsu oraz tego odpowiadającego za logikę aplikacji.

Aplikacja podzielona jest na trzy warstwy, co obrazuje schemat wdrożenia (rys. 5.1). Aplikacja na urządzeniu użytkownika reprezentuje frontend, czyli część związaną z interfejsem. Serwer aplikacji oznacza backend - część funkcjonalną. Dwie opisane warstwy programu komunikują się ze sobą przy użyciu ządań HTTP. Backend komunikuje się również z bazą danych - trzecią częścią aplikacji - z wykorzystaniem JPA oraz Hibernate. Ostatnim elementem schematu jest lokalizator, nie należący do żadnej z warstw aplikacji, natomiast jest on ważnym elementem podczas działania systemu i komunikuje się z częścią funkcjonalną za pomocą protokołu TCP.

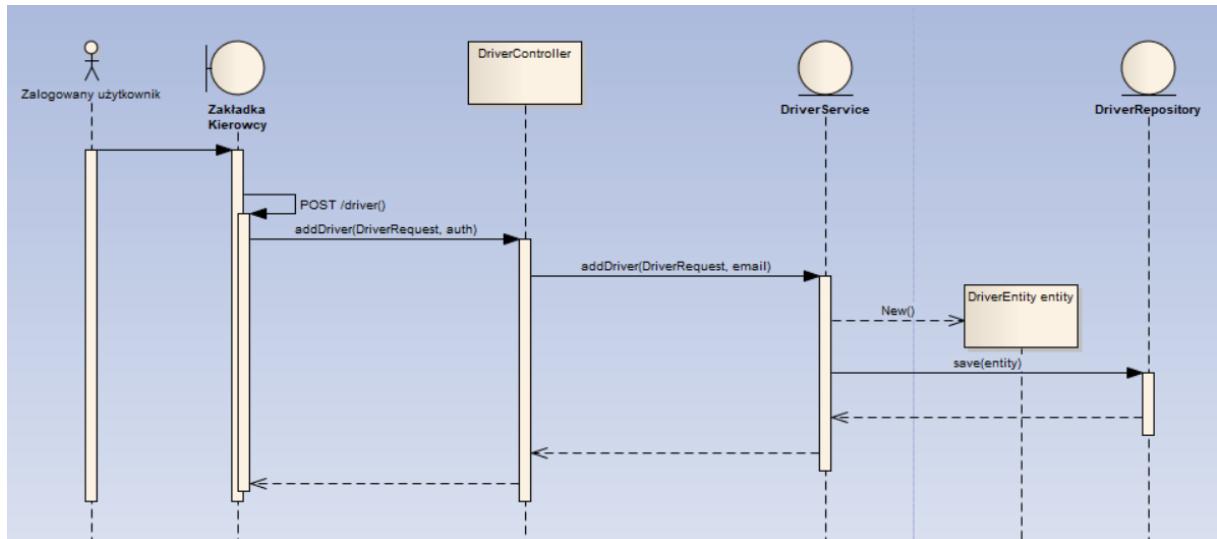
Widok użytkownika (frontend) napisany został w języku TypeScript, bazuje na komponentach biblioteki React. Podłożem aplikacji (backend) jest oparte na języku Java. Znanym wzorcem projektowym jest MVC (ang. Model View Controller), co przetłumaczone na język polski brzmi: model, widok, kontroler. Model to część odpowiadająca za przechowywanie danych. Widok to część dostępna dla użytkownika aplikacji, natomiast kontroler wykonuje operacje i jest odpowiedzialny za logikę programu. Do stworzenia programu



Rysunek 5.1: Diagram wdrożenia.

wykorzystano architekturę warstwową, która bazuje na MVC oraz skorzystano z wzorca projektowego o nazwie repozytorium. Repozytorium oznacza oddzielenie logiki biznesowej od bazy danych, natomiast architektura warstwowa zakłada istnienie takich warstw jak: encje - będące reprezentacją danych, kontrolery - obsługujące żądania HTTP, serwisy - odpowiedzialne za logikę biznesową i będące wywoływanymi przez kontrolery, repozytoria - mające dostęp do bazy danych, wykonujące operacje zapisu lub pobrania danych. Mapowanie encji na tabele w bazie danych, jak i odwrotnie, jest dostępne dzięki zastosowaniu adnotacji Hibernate.

Proces interakcji między częściami aplikacji podczas realizowania operacji zainicjowanej przez użytkownika przedstawia diagram sekwencji, znajdujący się na rys. 5.2. Obrazuje on przypadek dodania kierowcy. Po wciśnięciu na interfejsie przycisku odpowiadającego za dodanie kierowcy, wywoływany jest odpowiedni punkt końcowy (ang. endpoint). W efekcie tego, uruchomiona zostaje metoda w kontrolerze, której następstwem jest wywołanie metody w serwisie. Logika umieszczona w serwisie tworzy nowy obiekt encji kierowcy w oparciu o żądanie znajdujące się argumencie metody, które na początku zostało wysłane z interfejsu. Nowo utworzony obiekt zostaje następnie zapisany do bazy danych przy użyciu repozytorium.



Rysunek 5.2: Diagram sekwencji - dodanie kierowcy.

## 5.2 Opis struktur danych

Dane w kodzie języka Java są reprezentowane przez encje. Są to klasy posiadające pola - zmienne odzwierciedlające kolumny w tabelach w bazie danych. Ich obiekty są odwzorowaniem bazodanowych rekordów. Przykład kodu można zobaczyć na rys. 5.3. Każda zmienna posiada adnotację (poprzedzoną znakiem "@"), jest to składania Hibernate. Dzięki temu jest możliwe mapowanie między obiektem encji, a rekordem w tabeli. Encje posiadają również funkcje nazywane getterami - do pozyskiwania danych z konkretnych pól oraz setterami - do wpisywania wartości do pól.

Aby wysyłać i odbierać dane od tzw. frontendu, czyli części aplikacji dotyczącej interfejsu użytkownika, niezbędne są klasy reprezentujące żądania (ang. request) - odbierane są przez kontrolery oraz odpowiedzi (ang. response) - wysyłane dane do części z językiem TypeScript. Zazwyczaj mają pola analogiczne do encji. Posiadają również funkcje pobierające wartości z konkretnych zmiennych oraz zapisujące wartości do nich, tak jak jest to w przypadku encji. Odpowiedzi są również wyposażone w funkcje konwertujące obiekt encji na obiekt odpowiedzi. Przykład klasy żądania znajduje się na rys. 5.4, a metody do konwersji, będącej w klasie odpowiedzi, na rys. 5.5

Analogiczne typy danych zostały stworzone w języku TypeScript w formie interfejsów, pozwalają one w prosty sposób wysyłać żądania i odbierać odpowiedzi. Posiadają one zmienne o identycznych nazwach co klasy w języku Java oraz typy, które pozwalają na automatyczne ich konwertowanie między językami podczas wysyłania i odbierania żądań HTTP. Wygląd interfejsów przedstawia rys. 5.6.

```
1  @Entity
2  @Table(name = "user", schema = "adm")
3  public class UserEntity {
4      @Id
5      @Column
6      private String email;
7
8      @Column
9      private String password;
10
11     @OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
12     private Set<DriverEntity> driver;
13
14     public String getEmail() {
15         return email;
16     }
17
18     public void setEmail(String username) {
19         this.email = username;
20     }
21
22     public String getPassword() {
23         return password;
24     }
25
26     public void setPassword(String password) {
27         this.password = password;
28     }
29
30     public Set<DriverEntity> getDriver() {
31         return driver;
32     }
33
34     public void setDriver(Set<DriverEntity> driver) {
35         this.driver = driver;
36     }
37 }
```

---

Rysunek 5.3: Kod encji użytkownika - Java.

```
1 public class UserRequest {  
2  
3     @NotNull  
4     private String email;  
5  
6     @NotNull  
7     private String password;  
8  
9     public String getEmail() {  
10        return email;  
11    }  
12  
13    public void setEmail(String email) {  
14        this.email = email;  
15    }  
16  
17    public String getPassword() {  
18        return password;  
19    }  
20  
21    public void setPassword(String password) {  
22        this.password = password;  
23    }  
24 }
```

---

Rysunek 5.4: Kod żądania użytkownika.

```
1 public static TrackerResponse fromEntity(TrackerEntity entity){  
2     TrackerResponse dto = new TrackerResponse();  
3     dto.setSerialNumber(entity.getSerialNumber());  
4     dto.setName(entity.getName());  
5     dto.setType(entity.getType());  
6     return dto;  
7 }
```

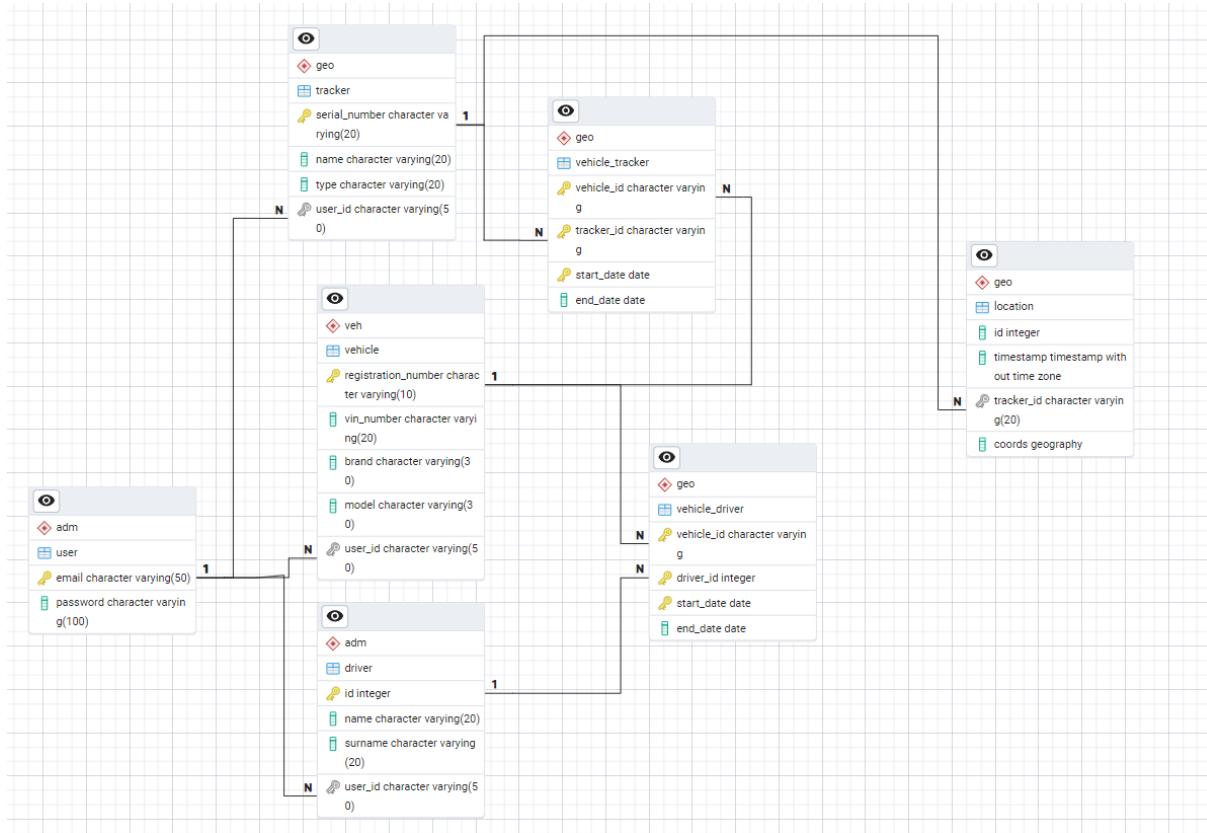
---

Rysunek 5.5: Funkcja konwertująca encję na odpowiedź.

```
1 export interface UserRequest {  
2     email: string | null;  
3     password: string | null;  
4 }
```

---

Rysunek 5.6: Kod żądania użytkownika - TypeScript.



Rysunek 5.7: Schemat bazy danych.

## 5.3 Model danych

Schemat bazy danych znajduje się na rys. 5.7. Podstawową tabelą jest tabela użytkownika ("user") - przechowuje ona dane użytkowników systemu. Aplikacja umożliwia dodanie wielu kierowców, pojazdów i lokalizatorów, dlatego tabele: "drivers", "vehicles" oraz "trackers" są powiązane z użytkownikiem relacjami N:1. Dzięki temu jest możliwe wyświetlanie użytkownikowi odpowiednich pozycji w listach, gdyż każdy obiekt jest przypisany do konkretnego użytkownika. Tabela "location" przechowuje dane geograficzne zebrane od urządzeń GPS. Dzięki powiązaniu z tabelą lokalizatorów, można uzyskać łatwy dostęp do historii położenia danego urządzenia. Aby uzyskać dostęp do trasy przebytej przez samochód, niezbędne jest powiązanie między lokalizatorem, a pojazdem reprezentowane przez tabelę "vehicle\_tracker". Analogiczne powiązanie - pojazdu z kierowcą ("vehicle\_driver") - umożliwia otrzymywanie danych lokalizacyjnych dla konkretnego kierowcy.

## 5.4 Szczegóły implementacji wybranych fragmentów

### 5.4.1 Komponenty React

Do stworzenia interfejsu użytkownika wykorzystano komponenty funkcyjne z biblioteki React. Każda zakładka aplikacji posiada oddzielnny komponent - każdy w osobnym

---

```

1 return (
2     <div className="main-container">
3         <h1 className="welcome-title">Witaj w GeoApp</h1>
4         <p className="text-description1">
5             </p>
6         <p className="text-description2">
7             </p>
8     </div>
9 );

```

---

Rysunek 5.8: Zwracany JSX przez zakładkę strony głównej.

---

```

1 type FormState = {
2     name: string;
3     surname: string;
4 };

```

---

Rysunek 5.9: Typ danych do formularza.

pliku. Zwracają one JSX, czyli kod zbliżony do HTML, w którym można zagnieździć inne elementy. Niemniej jednak, należy pamiętać o zasadzie, iż zwrócić można tylko jeden element nadzędny - to w nim można umieścić kolejne. Rys.5.8 pokazuje zwracany JSX przez zakładkę strony głównej (tekst z elementów <p> został wycięty).

Ważnym elementem stworzonej aplikacji są formularze, to dzięki nim użytkownik dodaje obiekty czy powiązania. W tym celu została wykorzystana biblioteka React Hook Form, dostępna w React. Stworzono typy danych, którego przykład można zobaczyć na rys. 5.9. Aby skorzystać z biblioteki, wykorzystano Hook do destrukturyzacji niezbędnych stałych (rys. 5.10). Aby skorzystać z obsługi formularza, zarejestrowano pola oraz dodano validacje - przykład na rys. 5.11.

Mapa została zaimplementowana przy użyciu biblioteki Leaflet. Inicjalizacja mapy następuje w `useEffect()`, czyli w funkcji React, która dzięki pustej tablicy zależności, znajdującej się na samym końcu, wykonuje się raz, gdy komponent mapy zostaje stworzony. Inicjalizację przedstawia rys. 5.12. W celu wyświetlania tras na mapie, wykorzystano

---

```

1 const [formState, setFormState] = useState<string>("addDriver");
2 const {
3     register,
4     handleSubmit,
5     formState: { errors },
6     reset,
7 } = useForm<FormState>();

```

---

Rysunek 5.10: Destrukturyzacja stałych z biblioteki React Hook Form.

```
1 <label className="form-label" htmlFor="name">Imie</label>
2 <input
3   className="form-input"
4   id="name"
5   {...register("name", { required: "Imię jest wymagane" })} 
6 />
7 { errors.name && <p className="form-error form-validation-text">{
  errors.name.message}</p>}
```

---

Rysunek 5.11: Rejestracja i validacja pól.

funkcję polyline(), która łączy linią kolejne punkty, przekazane jako argument w formie tablicy. Ta część kodu widnieje na rys. 5.13. Utworzona trasa zostaje dodana do mapy.

#### 5.4.2 Proces generowania i obsługi żądań

Pierwszym etapem jest wywołanie w komponencie React funkcji wysyłającej żądanie HTTP. Przykładowym procesem będzie pobranie listy kierowców danego użytkownika. Na rys. 5.14 wywołana zostaje funkcja getDrivers(), której implementację przedstawia rys. 5.15. Jest to funkcja w języku TypeScript, która umożliwia wysłanie żądania z wykorzystaniem metody GET, na bazowy URL z punktem końcowym "/driver". Następnym krokiem jest odebranie żądania w kontrolerze, czyli przez część funkcjonalną aplikacji. Implementacja metody kontrolera związana z pobraniem listy kierowców znajduje się na rys. 5.16. Za logikę jest odpowiedzialny serwis, stąd metoda widniejąca w kontrolerze jest niewielkich rozmiarów. Niemniej jednak, warto zwrócić uwagę na argument o nazwie "auth" on pobiera dane obecnie zalogowanego użytkownika, które są niezbędne do znalezienia odpowiednich kierowców. Serwis wykorzystuje repozytorium do wyszukania odpowiednich danych z bazy danych - co można zaobserować na rys. 5.17, a następnie zwraca listę kierowców, która przez kontroler jest wysyłana jako odpowiedź. Ostatecznie te dane trafiają do komponentu React.

```
1  useEffect(() => {
2      if (!mapRef.current && document.getElementById("map")) {
3          mapRef.current = leaflet.map("map").setView([50.299,
4              18.787], 13);
5
6          leaflet
7              .tileLayer("https://tile.openstreetmap.org/{z}/{x}/{y}."
8                  .png", {
9                  maxZoom: 19,
10                 attribution:
11                     '&copy; <a href="http://www.openstreetmap.org/
12                     copyright">OpenStreetMap</a>',
13
14                 })
15                 .addTo(mapRef.current);
16
17             loadTrackerData();
18             loadVehicleData();
19             loadDriverData();
20
21         return () => {
22             mapRef.current?.remove();
23             mapRef.current = null;
24         };
25     }, []);
26 }
```

---

Rysunek 5.12: Inicjalizacja mapy.

```
1 useEffect(() => {
2     if (!mapRef.current) return;
3
4     mapRef.current.eachLayer((layer) => {
5         if (layer instanceof leaflet.Polyline) {
6             mapRef.current?.removeLayer(layer);
7         }
8     });
9
10    const latLngs: [number, number][] = locations
11        .filter(loc => loc.latitude !== null && loc.longitude !==
12            null)
13        .map(loc => [loc.latitude as number, loc.longitude as
14            number]);
15
16    if (latLngs.length > 1) {
17        leaflet.polyline(latLngs, { color: "blue", weight: 4 })
18            .addTo(mapRef.current!);
19    }
20 }, [locations]);
```

---

Rysunek 5.13: Dodanie trasy do mapy.

```
1 const loadDriverData = async () => {
2     const data = await getDrivers();
3     setDrivers(data);
4 };
```

---

Rysunek 5.14: Wywołanie funkcji "getDrivers()", wysyłającej żądanie HTTP.

```
1 export async function getDrivers(): Promise<DriverResponse[] | null> {
2     try {
3         const response = await request("GET", "/driver");
4         return response.data;
5     } catch (error) {
6         console.error("Error getting drivers:", error);
7         return null;
8     }
9 }
```

---

Rysunek 5.15: Kod funkcji wysyłającej żądanie HTTP w celu pobrania listy kierowców.

---

```
1 @GetMapping( "")  
2 public ResponseEntity<DriverResponse[]> getDrivers(  
3     Authentication auth) {  
4     UserResponse user = (UserResponse) auth.getPrincipal();  
5     return ResponseEntity.ok(driverService.getDrivers(user.  
6         getEmail()));  
7 }
```

---

Rysunek 5.16: Metoda kontrolera, pobierająca listę kierowców.

---

```
1 @Transactional  
2 public DriverResponse[] getDrivers(String email) {  
3     return driverRepository.findAllByEmail(email).stream()  
4         .map(DriverResponse::fromEntity)  
5         .toArray(DriverResponse[]::new);  
6 }
```

---

Rysunek 5.17: Metoda serwisu, pobierająca listę kierowców.



# Rozdział 6

## Weryfikacja i walidacja

Testowanie jest nieodzowną częścią podczas prac nad projektem. Pozwala wyeliminować błędy, powstałe na skutek implementacji skomplikowanych obliczeń czy też niedopatrzenia ze strony autora. Zrezygnowanie z tej czynności wiąże się z dużo większym prawdopodobieństwem istnienia problemów związanych z aplikacją, a to znaczająco zwiększa ryzyko stworzenia niekompletnego programu.

### 6.1 Sposób testowania

Aby przetestować działanie serwera oraz prawidłowość w konfiguracji lokalizatora w pierwszej kolejności skonfigurowano router oraz komputer lokalny, celem wykorzystania przekierowania portów. Napotkano problem, związany z dostawcą internetu, co uniemożliwiało odbieranie danych z lokalizatora przez router. Następnie zainstalowano narzędzie Ngrok, które umożliwia przekierowanie portów oraz udostępnia publiczną domenę. Znaleziono problem w tym sposobie testowania, gdyż publiczna domena zmieniała adres IP w bardzo krótkim czasie. Po skonfigurowaniu lokalizatora na nowy adres, czyli wysłaniu odpowiedniej komendy SMS z nowym adresem, adres IP otrzymywany przez Ngrok ponownie się zmieniał, co uniemożliwiło otrzymanie pakietów TCP od urządzenia GPS. Ostatecznie wykorzystano maszynę wirtualną z systemem Linux Ubuntu, która posiadała publiczny adres IP. Zakupiono lokalizator Mking MK07A oraz kartę SIM z pakietem internetu oraz wysyłką wiadomości SMS. Następnie urządzenie zostało skonfigurowany poprzez SMS-y tak, aby wysyłało dane na adres maszyny wirtualnej z użyciem portu 8080. Jednak serwer w języku Java był dostępny na lokalnym komputerze stacjonarnym z systemem Windows. Aby przekierować pakiety na serwer, skorzystano z programu Moba-XTerm, który umożliwia tunelowanie portów TCP. Kolejnym korkiem było zatrzymywanie serwera w odpowiednich miejscach w programie IntelliJ Idea, dzięki czemu następnie eliminowano błędy wynikające z niewłaściwej interpretacji pakietów oraz poprawiano kod serwera.

W celu sprawdzenia aplikacji pod kątem błędów skorzystano z testowania modelem V. Jest to model sekwencyjny, który składa się z testów takich jak testy integracyjne, systemowe czy akceptacyjne. Podczas testów integracyjnych, przechodzono przez zakładki aplikacji, sprawdzając poprawność działania. Ważnym aspektem programu były formularze, zatem im również poświecono uwagę - testowano ich pola, oraz zastosowane w nich walidacje, w tym czy dane zostają przesłane, podczas gdy formularz jest niepoprawnie wypełniony i zawiera błędy. Sprawdzone zostało również działanie aplikacji na ekranach różnej wielkości, aby upewnić się, czy aplikacja jest uniwersalna i dostosowana do urządzeń mobilnych.

## 6.2 Wykryte i usunięte błędy

Podczas testów aplikacji wykryto, że formularze pozwalają wpisać w pola dat zakończenia daty wcześniejsze niż te będące już w polach dat rozpoczęcia. Rozwiązano ten błąd poprzez wprowadzenie walidacji na pola z datami, aby data do była musiała być późniejszą datą niż data od.

# Rozdział 7

## Podsumowanie i wnioski

Projekt pozwolił autorowi na zgłębienie wiedzy na temat lokalizatorów GPS oraz ich działania, jak również dotyczącej komunikacji za pomocą protokołu TCP. Poznano wiele narzędzi do zarządzania połączeniami sieciowymi, takich jak Ngrok oraz MobaXTerm. Jednoosobowe budowanie aplikacji przeglądarkowej pozwoliło na rozwój w kierunku programisty typu fullstack - oznaczającego specjalizowanie się w pełnym sotsie technologicznym, który obejmuje: backend, frontend i bazę danych.

### 7.1 Uzyskane wyniki w świetle postawionych celów i zdefiniowanych wymagań

Aplikacja została napisana zgodnie z wymaganiami, które zostały przedstawione. Odpowiednio skonfigurowany lokalizator - co ułatwia instrukcja dostępna w programie - poprawnie współpracuje z serwerem, w wyniku czego można korzystać z aplikacji w sposób zgodny z początkowymi celami. Prosty interfejs pozwala użytkownikowi na sprawne i przyjemne obsługiwanie programu. Widok został dostosowany do różnej wielkości ekranów, co wskazuje na sukces podczas wprowadzania skalowalności. Wszystkie operacje, które miały być dostępne do wykonania przez użytkownika, i które miały na celu osiągnięcie użytecznej aplikacji, zostały wprowadzone.

### 7.2 Kierunki dalszego rozwoju systemu

Dalszym etapem rozwijania aplikacji byłoby przede wszystkim zwiększenie dostępnych typów lokalizatorów, wspieranych przez program. W tym celu należy rozbudować kod serwera, odbierającego dane geograficzne od urządzeń. Przykładem innego lokalizatora jest Coban TK108, który również umożliwia wysyłanie pakietów TCP na niestandardowy adres IP oraz port. Takie działania zwiększyłyby grupę potencjalnych użytkowników aplikacji, gdyż nie byłby oni ograniczeni do zakupu jednego modelu lokalizatora.



# Bibliografia

- [1] Joshua Bloch. *Java. Efektywne programowanie*. Gliwice: Wydawnictwo Helion, 2018. ISBN: 978-83-283-9896-2.
- [2] Spring Boot. *Spring Boot - oficjalna strona producenta*. 2025. URL: <https://spring.io/projects/spring-boot>.
- [3] David Flanagan. *JavaScript. The Definitive Guide*. Sebastopol: O'Reilly Media, 2020. ISBN: 978-1565923928.
- [4] Git. *Git - oficjalna strona producenta*. 2025. URL: <https://git-scm.com/doc>.
- [5] Hibernate. *Hibernate - oficjalna strona producenta*. 2025. URL: <https://hibernate.org/orm/>.
- [6] Jeff Hurn. *GPS: A Guide to the Next Utility*. Sunnyvale: Trimble Navigation Ltd, 1989. ISBN: 978-999-9253-7-015.
- [7] PostGIS. *PostGIS - oficjalna strona producenta*. 2025. URL: <https://postgis.net/documentation/>.
- [8] PostgreSQL. *PostgreSQL - oficjalna strona producenta*. 2025. URL: <https://www.postgresql.org/docs/>.
- [9] React. *React - oficjalna strona producenta*. 2025. URL: <https://react.dev/learn>.
- [10] TypeScript. *TypeScript - oficjalna dokumentacja producenta*. 2025. URL: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>.



# **Dodatki**



# **Lista dodatkowych plików, uzupełniających tekst pracy**

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,



# Spis rysunków

2.1	Zrzut ekranu przedstawiający wygląd aplikacji MKING GPS. . . . .	5
2.2	Zrzut ekranu przedstawiający wygląd aplikacji iTrackLive. . . . .	6
3.1	Diagram przypadków użycia niezalogowanego użytkownika. . . . .	9
3.2	Diagram przypadków użycia zalogowanego użytkownika. . . . .	9
4.1	Zrzut ekranu przedstawiający stronę główną. . . . .	16
4.2	Zrzut ekranu przedstawiający formularz logowania. . . . .	17
4.3	Zrzut ekranu przedstawiający formularz rejestracji. . . . .	17
4.4	Zrzut ekranu przedstawiający zakładkę Moje konto. . . . .	18
4.5	Zrzut ekranu przedstawiający zakładkę Kierowcy. . . . .	19
4.6	Zrzut ekranu przedstawiający formularz dodawania kierowcy. . . . .	19
4.7	Zrzut ekranu przedstawiający zakładkę Kierowcy z jednym kierowcą na liście.	19
4.8	Zrzut ekranu przedstawiający zakładkę Pojazdy. . . . .	20
4.9	Zrzut ekranu przedstawiający formularz dodawania pojazdu. . . . .	21
4.10	Zrzut ekranu przedstawiający zakładkę Lokalizatory. . . . .	21
4.11	Zrzut ekranu przedstawiający formularz dodawania lokalizatora. . . . .	22
4.12	Zrzut ekranu przedstawiający formularz dodawania lokalizatora. . . . .	22
4.13	Zrzut ekranu przedstawiający formularz dodawania powiązania kierowcy z pojazdem. . . . .	23
4.14	Zrzut ekranu przedstawiający formularz dodawania powiązania lokalizatora z pojazdem. . . . .	24
4.15	Zrzut ekranu przedstawiający zakładkę Mapa. . . . .	25
4.16	Zrzut ekranu przedstawiający formularz dodawania trasy kierowcy. . . . .	26
4.17	Zrzut ekranu przedstawiający formularz dodawania trasy pojazdu. . . . .	26
4.18	Zrzut ekranu przedstawiający mapę z jedną trasą. . . . .	27
4.19	Zrzut ekranu przedstawiający mapę z jedną trasą. . . . .	27
5.1	Diagram wdrożenia. . . . .	30
5.2	Diagram sekwencji - dodanie kierowcy. . . . .	31
5.3	Kod encji użytkownika - Java. . . . .	32
5.4	Kod żądania użytkownika. . . . .	33

5.5	Funkcja konwertująca encję na odpowiedź . . . . .	33
5.6	Kod żądania użytkownika - TypeScript . . . . .	33
5.7	Schemat bazy danych . . . . .	34
5.8	Zwracany JSX przez zakładkę strony głównej . . . . .	35
5.9	Typ danych do formularza . . . . .	35
5.10	Destrukturyzacja stałych z biblioteki React Hook Form . . . . .	35
5.11	Rejestracja i walidacja pól . . . . .	36
5.12	Inicjalizacja mapy . . . . .	37
5.13	Dodanie trasy do mapy . . . . .	38
5.14	Wywołanie funkcji "getDrivers()", wysyłającej żądanie HTTP . . . . .	38
5.15	Kod funkcji wysyłającej żądanie HTTP w celu pobrania listy kierowców . . . . .	38
5.16	Metoda kontrolera, pobierająca listę kierowców . . . . .	39
5.17	Metoda serwisu, pobierająca listę kierowców . . . . .	39