

**Politechnika Wrocławska**  
**Wydział Informatyki i Telekomunikacji**

---

Kierunek: **Informatyka techniczna (ITE)**  
Specjalność: **Inżynieria systemów informatycznych (INS)**

**PRACA DYPLOMOWA**  
**INŻYNIERSKA**

**Aplikacja webowa do optymalizacji  
zleceń w transporcie towarów**

**Web application for optimizing orders  
in the transport of goods**

Krystian Tomczyk

Opiekun pracy  
**dr. inż. Paweł Rogaliński**

Słowa kluczowe: aplikacja, transport, web

---

WROCŁAW, 2025

## Streszczenie

Tu będzie streszczenie po Polsku

**Słowa kluczowe:** aplikacja, transport, web

## Abstract

Tu będzie streszczenie po angielsku

**Keywords:** application, transport, web

# Spis treści

<b>1. Wstęp</b>	<b>7</b>
1.1. Cel i zakres projektu	8
1.2. Wymagania aplikacji	8
<b>2. Projekt systemu CargoLink</b>	<b>10</b>
2.1. Identyfikacja aktorów	10
2.2. Projekt interfejsu użytkownika	10
2.3. Diagramy przypadków użycia	11
2.3.1. Niezalogowany użytkownik	12
2.3.2. Przewoźnik	24
2.3.3. Zleceniodawca	34
2.3.4. Moderator i Administrator	36
2.4. Projekt bazy danych	38
<b>3. Implementacja</b>	<b>40</b>
3.1. Opis narzędzi	40
3.2. Opis architektury	42
3.2.1. Architektura warstw	42
3.3. Opis implementacji	44
3.3.1. Implementacja mechanizmu publikacji ogłoszeń o planowanej trasie	44
3.3.2. Implementacja wizualizacji tras na mapie	47
3.3.3. Implementacja algorytmu rekomendacji ogłoszeń	52
3.4. Opis testów aplikacji	55
3.4.1. Testy jednostkowe	55
3.4.2. Testy end-to-end	57
3.4.3. Testy wydajnościowe	58
<b>4. Podsumowanie i wnioski</b>	<b>60</b>
4.1. Napotkane problemy	60
4.2. Możliwości rozwoju	60
4.3. Wnioski	60
<b>Literatura</b>	<b>61</b>
<b>A. Instrukcja wdrożeniowa</b>	<b>62</b>

# Spis rysunków

2.1. Wybrane parametry dotyczące interfejsu użytkownika . . . . .	11
2.2. Diagram przedstawiający przypadki użycia aktora Niezalogowany użytkownik .	12
2.3. Formularz rejestracji w wersji mobilnej: a) Menu logowania, b) Wybór typu konta, c) Wprowadzenie danych do rejestracji . . . . .	13
2.4. Formularz rejestracji w wersji desktopowej: a) Menu logowania, b) Wybór typu konta	13
2.5. Formularz rejestracji w wersji desktopowej: Wprowadzenie danych do rejestracji . .	14
2.6. Formularz rejestracji w wersji mobilnej: a) Wybór reprezentacji konta, b) Wybór języków, którymi użytkownik się posługuje, . . . . .	14
2.7. Formularz rejestracji w wersji desktopowej: a) Wybór reprezentacji konta, b) Wybór języków, którymi użytkownik się posługuje, . . . . .	15
2.8. Menu nawigacji po aplikacji w wersji mobilnej: a) Menu nawigacji, b) Menu nawi- gacji z klikniętym przyciskiem wyboru języka . . . . .	16
2.9. Menu wyboru języka aplikacji w wersji desktopowej . . . . .	16
2.10. Wyszukiwanie użytkownika w wersji mobilnej . . . . .	17
2.11. Wyszukiwanie użytkownika w wersji desktopowej . . . . .	17
2.12. Przeglądarka zleceń i planowanych tras w wersji mobilnej: a) Przeglądarka zleceń, b) Przeglądarka ogłoszeń planowanych tras . . . . .	18
2.13. Przeglądarka zleceń i planowanych tras w wersji desktopowej: a) Przeglądarka zle- ceń, b) Przeglądarka ogłoszeń planowanych tras . . . . .	18
2.14. Wyświetlenie ogłoszenia w wersji mobilnej: a) Ogłoszenie zlecenia, b) Ogłoszenie o planowanej trasie . . . . .	20
2.15. Wyświetlenie ogłoszenia w wersji desktopowej: a) Ogłoszenie zlecenia, b) Ogłosze- nie o planowanej trasie . . . . .	21
2.16. Mapa ze wszystkimi trasami w wersji mobilnej . . . . .	22
2.17. Mapa ze wszystkimi trasami w wersji desktopowej . . . . .	22
2.18. Profil użytkownika w wersji mobilnej . . . . .	23
2.19. Profil użytkownika w wersji desktopowej . . . . .	24
2.20. Diagram przedstawiający przypadki użycia aktora Przewoźnik . . . . .	24
2.21. Dodawanie nowego ogłoszenia o planowanej trasie w wersji mobilnej: a) Menu po zalogowaniu jako przewoźnik b) Formularz dodawania ogłoszenia o planowanej trasie	25
2.22. Dodawanie nowego ogłoszenia o planowanej trasie w wersji desktopowej: a) Menu po zalogowaniu jako przewoźnik b) Formularz dodawania ogłoszenia o planowanej trasie . . . . .	26
2.23. Czat w wersji mobilnej: a) Menu wyboru konwersacji, b) Czat . . . . .	26
2.24. Czat w wersji desktopowej . . . . .	27
2.25. Formularz generujący umowę w wersji mobilnej . . . . .	28
2.26. Formularz generujący umowę w wersji desktopowej . . . . .	28
2.27. Zaakceptowanie warunków umowy w wersji mobilnej . . . . .	29
2.28. Zaakceptowanie warunków umowy w wersji mobilnej . . . . .	30
2.29. Edycja profilu w wersji mobilnej . . . . .	31

2.30. Edycja profilu w wersji mobilnej . . . . .	32
2.31. Dodawanie opinii o użytkowniku w wersji mobilnej: a) Profil użytkownika gdy dostępne jest dodanie opinii b) Formularz dodawania opinii . . . . .	33
2.32. Dodawanie opinii o użytkowniku w wersji desktopowej: a) Profil użytkownika gdy dostępne jest dodanie opinii b) Formularz dodawania opinii . . . . .	33
2.33. Diagram przedstawiający przypadki użycia aktora Zleceniodawca . . . . .	34
2.34. Dodawanie nowego zlecenia w wersji mobilnej: a) Menu po zalogowaniu jako zleceniodawca b) Formularz dodawania zlecenia . . . . .	35
2.35. Dodawanie nowego zlecenia w wersji desktopowej: a) Menu po zalogowaniu jako zleceniodawca b) Formularz dodawania zlecenia . . . . .	35
2.36. Diagram przedstawiający przypadki użycia aktorów Moderator i Administrator . . . . .	36
2.37. Akceptacja nowych zleceń i ogłoszeń o planowanej trasie w wersji mobilnej: a) Menu po zalogowaniu jako moderator lub administrator b) Panel akceptacji nowych ogłoszeń . . . . .	37
2.38. Akceptacja nowych zleceń i ogłoszeń o planowanej trasie w wersji desktopowej: a) Menu po zalogowaniu jako moderator lub administrator b) Panel akceptacji nowych ogłoszeń . . . . .	37
2.39. Projekt bazy danych . . . . .	38
3.1. Diagram architektury warstw aplikacji . . . . .	42
3.2. Wyniki uruchomienia testów jednostkowych . . . . .	56
3.3. Wyniki uruchomienia testów end-to-end . . . . .	57
3.4. Wyniki testów wydajnościowych . . . . .	58
3.5. Pochodzenie danych do testów wydajnościowych . . . . .	59

# Spis listingów

2.1. Przykład kwerendy SQL . . . . .	39
3.1. Wstępna walidacja danych przed dodaniem ogłoszenia do bazy danych . . . . .	44
3.2. Schemat walidacji danych wejściowych . . . . .	45
3.3. Implementacja dodawania ogłoszenia do bazy danych . . . . .	46
3.4. Argumenty komponentu mapy . . . . .	47
3.5. Implementacja pobierania najkrótszych tras między punktami . . . . .	48
3.6. Implementacja rysowania tras na mapie . . . . .	50
3.7. Implementacja rekomendacji ogłoszeń . . . . .	52
3.8. Funkcja sprawdzająca warunki powiązania ogłoszeń . . . . .	53
3.9. Funkcje obliczająca odległość między punktami . . . . .	54
3.10. Funkcja sprawdzająca czy ogłoszenie zawiera się w odpowiednim zakresie właściwości fizycznych towaru . . . . .	55
3.11. Przykładowy test jednostkowy paska nawigacyjnego . . . . .	56

\*

# Rozdział 1

## Wstęp

Transport towarów odgrywa kluczową rolę w globalnej gospodarce, łącząc producentów i konsumentów na całym świecie. Efektywność transportu ma bezpośredni wpływ na koszty operacyjne firm oraz na ceny finalnych produktów. W zależności od specyfiki przewożonych towarów oraz potrzeb zleceniodawców, transport może przyjmować dwie następujące formy:

Transport regularny, inaczej łańcuch dostaw, to przewóz towarów, który odbywa się według ustalonego harmonogramu i stałych tras. Charakteryzuje się regularnością kursów, co oznacza, że pojazdy wykonują swoje trasy w określonych, z góry ustalonych terminach. Przykładami transportu regularnego są linie autobusowe, kolejowe czy lotnicze, które działają według stałego rozkładu jazdy.

Transport okazjonalny to przewóz towarów, który odbywa się bez ustalonego z góry rozkładu jazdy. Pojazdy wykonują swoje trasy w zależności od zapotrzebowania klientów, najczęściej jest to usługa jednorazowa. Sam przewóz zaś zlecany jest na potrzebę klienta, nie musi on jednak określać dokładnego terminu odbycia trasy, ani przez kogo ma on być zrealizowany.

Podczas swoich tras przewoźnicy czasami są zmuszeni do przebycia części drogi bez żadnego ładunku. Powoduje to, że przewozy nie są w pełni zoptymalizowane względem kosztów, jakie niesie za sobą pokonywana trasa. Możliwe jest jednak zredukowanie występowania takich sytuacji poprzez odpowiednie powiązanie przewoźników i osób zlecających transport okazjonalny. Zleceniodawca, który nie potrzebuje dostawy towaru w konkretnej dacie, mógłby wtedy zlecić transport z nieokreślonym dokładnie terminem dotarcia towaru, w zamian za niższe ceny przewozowe. Przykład: dyrektor szkoły, w czasie wakacji, zamówił dużych rozmiarów tablicę interaktywną, która nie zmieściłaby się w standardowym samochodzie osobowym. Z racji, że zamówienie zostało złożone w czasie, gdy dzieci nie chodzą do szkoły, nie zależy mu na dokładnym terminie dostawy. Może on w takim przypadku zlecić dostawę tablicy w formie transportu okazjonalnego, z mniejszymi kosztami transportu. Przewoźnik mógłby zabrać towar i zawieźć go na miejsce docelowe, gdy akurat wykonywałby trasę bez ładunku i kierował się w przybliżonym kierunku. Taka sytuacja jest korzystna dla obu stron, przewoźnik może odbywać przewozy bardziej efektywnie, dzięki nie marnowaniu zasobów na puste przebiegi. Zleceniodawca natomiast, może oczekiwać niższych kosztów przewozu towarów.

Połączenie między osobami zlecającymi usługi transportowe - zleceniodawcami, a osobami oferującymi przewóz towaru - przewoźnikami, może odbywać się za pomocą serwisu oferującego dodawanie publicznych ogłoszeń. Ogłoszenia dzielić się będą na dwie kategorie, ogłoszenie z planowaną trasą przejazdu oraz zlecenie z wymaganym towarem do przewiezienia z punktu początkowego do punktu docelowego.

## 1.1. Cel i zakres projektu

Celem projektu jest stworzenie aplikacji webowej, pod tytułem CargoLink. Serwis ten pozwalać będzie na dodawanie ogłoszeń oraz zleceń dotyczących transportów okazjonalnych. Aplikacja przyczyni się do zoptymalizowania procesów logistycznych, eliminując nieefektywne wykorzystanie zasobów transportowych. Dodatkowo pozwoli ona przewoźnikom i zleceniodawcom, na łatwą i szybką komunikację między sobą. Główne cele projektu:

1. **Możliwość umieszczania ogłoszeń i zleceń:** aplikacja pozwalać ma na dodawanie ogłoszeń o trasach przejazdu, planowanych przez przewoźników oraz zleceń transportowych na konkretny towar przez zleceniodawców
2. **Ułatwienie szukania odpowiednich ogłoszeń i zleceń:** poprzez system powiązania zleceń transportowych do planowanych tras przewoźników oraz ogłoszeń o planowanej trasie do zleceń zamieszczonych w serwisie, aplikacja skróci czas potrzebny na znalezienie odpowiednich ofert.
3. **Komunikacja między przewoźnikami i zleceniodawcami:** aplikacja umożliwi szybką komunikację między użytkownikami serwisu poprzez czat tekstowy.
4. **Redukcja kosztów transportu:** dzięki lepszemu dopasowaniu potencjalnych przewoźników i zleceniodawców, aplikacja pozwoli na obniżenie kosztów transportu zarówno dla zleceniodawców, jak i przewoźników.

Nowoczesna aplikacja transportowa powinna przyczynić się do efektywnego zrealizowania tych celów, co wspomogę rynek zleceń transportowych, przynosząc korzyści zarówno dla zleceniodawców, jak i przewoźników.

## 1.2. Wymagania aplikacji

Na podstawie analizy celów wymienionych w podrozdziale 1.1, można wywnioskować, że do efektywnego działania serwisu, będą musiały zostać zrealizowane następujące wymagania funkcjonalne:

1. **Uwierzytelnianie:** aplikacja będzie wykorzystywała system rejestracji oraz logowania.
2. **Regulamin:** podczas rejestrowania się do serwisu, użytkownik musi zaakceptować regulamin korzystania z aplikacji.
3. **Dodawanie zleceń transportowych:** serwis pomagał będzie znaleźć odpowiedniego przewoźnika poprzez udostępnienie możliwości dodania ogłoszenia zlecenia. W ogłoszeniu zlecenia znajdować się będzie:
  - tytułu zlecenia,
  - opis zlecenia (niewymagane),
  - wymagana trasa przewozu (miejsce startu oraz miejsce docelowe),
  - przybliżony termin dostarczenia (przedział dat),
  - waga towarów do przewiezienia,
  - wymiary przewożonych dóbr,
  - kategoria każdego z towarów,
  - informacja o wymaganych specjalnych warunkach podczas transportu (np. jedzenie wymagać będzie przewozu w odpowiedniej temperaturze, pole niewymagane),
  - imienia i nazwiska zleceniodawcy bądź nazwy firmy, która zleceniodawca reprezentuje,
4. **Dodawanie ogłoszeń o planowanej trasie:** system powinien pozwalać przewoźnikom, na dodawanie publicznych informacji o planowanych przez siebie trasach. Ogłoszenie będzie składało się z:
  - tytułu ogłoszenia,



- opisu ogłoszenia (niewymagane),
  - planowanej trasy przewozu (miejsce startu oraz miejsce docelowe),
  - daty planowanego przejazdu,
  - dostępnego miejsca w pojeździe (wymiary liczone w europaletach),
  - maksymalnej wagi towaru,
  - marki i modelu pojazdu,
  - danych kontaktowych,
  - imienia i nazwiska przewoźnika bądź nazwy firmy, która przewoźnik reprezentuje,
5. **System rekomendacji ogłoszeń:** podczas wprowadzania danych o trasie, użytkownik będzie informowany o sugerowanych zleceniach dodanych przez innych użytkowników (np. gdy zlecenie dotyczy trasy, która w przybliżeniu pokrywa się z tą jaką przewoźnik planuje się poruszać). Analogicznie gdy zleceniodawca zamierza dodać ogłoszenie zlecenia, zostanie on poinformowany o proponowanych ogłoszeniach przewoźników.
  6. **Umożliwienie kontaktu między użytkownikami:** aby zapewnić kontakt między użytkownikami, w aplikacji zostanie dodany czat tekstowy umożliwiający korespondencje między zleceniodawcami, a przewoźnikami, bezpośrednio w aplikacji. Ma on pełnić rolę komunikacji na wzór tej oferowanej przez tradycyjną pocztę elektroniczną.
  7. **Generowanie szablonu umowy:** przewoźnik i zleceniodawca, po negocjacji warunków umowy, otrzymają wygenerowany przez serwis szablon dokumentu finalizujący transakcję.
  8. **Weryfikacja dodawanych ogłoszeń:** zanim ogłoszenie wyświetlać się będzie dla wszystkich użytkowników, moderator serwisu będzie musiał je zatwierdzić.
  9. **Graficzne przedstawienie trasy:** podczas dodawania ogłoszenia o planowanej trasie, przewoźnikowi wyświetlać się będzie mapa z zaznaczonymi trasami, dodanymi przez zleceniodawców, które przebiegają w przybliżonym kierunku. Zleceniodawcy natomiast, podczas dodawania oferty, będą widzieć mapę ze wszystkimi trasami planowanymi przez przewoźników.
  10. **System ocen użytkownika:** po 2 dniach po upływie terminu przewozu, użytkownicy będą mogli dodać opinie na temat użytkownika, którego umowa dotyczyła. Opinia składała się będzie z oceny w skali 1-5 oraz komentarza. Opinie będą wyświetlały się na stronie profilu tego użytkownika oraz na dodanych przez niego ogłoszeniach.

Aplikacja musi być niezawodna i przyjazna do użytkowania dla wszystkich. Do komfortowego korzystania z serwisu przez użytkowników, niezbędna będzie realizacja następujących wymagań niefunkcjonalnych

1. **Innowacyjność:** Aplikacja musi używać nowoczesnych technologii, takich jak TypeScript, Next.js, Tailwind CSS, Node.js oraz bazę danych PostgreSQL, zapewni to wysoką wydajność, skalowalność i bezpieczeństwo aplikacji.
2. **Intuicyjny interfejs użytkownika:** Aplikacja musi posiadać prosty i intuicyjny interfejs użytkownika, który umożliwi łatwą obsługę zarówno dla zleceniodawców, jak i przewoźników.
3. **Dostępność na różnych urządzeniach:** Aplikacja musi być responsywna i dostosowana do różnych urządzeń, takich jak komputery, tablety i smartfony, co zapewni wygodę użytkowania w dowolnym miejscu i czasie.
4. **Wielojęzyczność:** użytkownicy korzystający z aplikacji, muszą mieć możliwość wyboru jednego z trzech przewidzianych języków: polski, angielski oraz niemiecki. Dodatkowo użytkownicy podczas rejestracji muszą mieć możliwość wybrania dowolnych języków, którymi się posługują. Informacje te będą zamieszczone na profilu użytkownika.

## Rozdział 2

# Projekt systemu CargoLink

W tym rozdziale zostanie opisany projekt systemu, analiza wymagań pod kątem przypadków użycia oraz projekt bazy danych. Każdy przypadek użycia zostanie omówiony, a jego działanie zilustrowane za pomocą makiet aplikacji. Przygotowana została makieta aplikacji w wersji na telefony komórkowe oraz wersja na standardowe monitory. Pierwszym podpunktem w nowym rozdziale będzie identyfikacja aktorów. Jest to kluczowy element w projektowaniu diagramu przypadków użycia aplikacji.

### 2.1. Identyfikacja aktorów

Biorąc pod uwagę wymagania opisane w poprzednich podrozdziałach, zaprojektowany został diagram przypadków użycia aplikacji. W systemie wyróżnić można następujących aktorów:

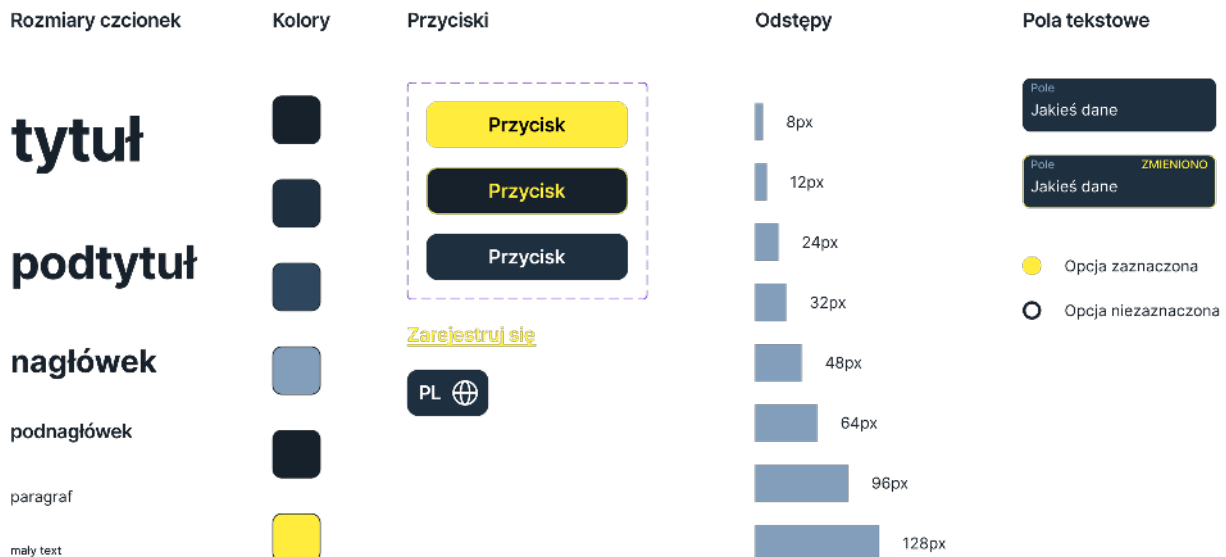
1. **Użytkownik niezalogowany** - nowy gość w serwisie, może się zalogować oraz przeglądać ogłoszenia przewozu i zlecenia dodane przez pozostałych użytkowników.
2. **Przewoźnik** - aktor odpowiedzialny za transport towarów. Przewoźnik może przeglądać dostępne zlecenia, dodawać ogłoszenia o planowanych trasach, komunikować się z autorami ogłoszeń, przyjmować zlecenia oraz oceniać i komentować kontrahentów.
3. **Zleceniodawca** - użytkownik systemu, który zleca transport towarów. zleceniodawca może dodawać nowe zlecenia transportowe, podobnie jak przewoźnik, może również przeglądać ogłoszenia przewoźników oraz komunikować się z autorami ogłoszeń.
4. **Moderator** - osoba odpowiedzialna za zarządzanie systemem. Moderator zatwierdza lub usuwa nowe ogłoszenia i zlecenia.
5. **Administrator** - użytkownik umiejscowiony najwyżej w hierarchii systemu. Może on wykonywać wszystko co moderator oraz ma możliwość dodawania nowych moderatorów lub usuwania obecnych.

### 2.2. Projekt interfejsu użytkownika

Makieta aplikacji została wykonana w darmowym programie Figma, który pozwala na tworzenie interfejsu użytkownika i oferuje wiele funkcji ułatwiających pracę, takich jak utrzymanie spójności w rozmiarach czcionek, kolorach czy odstępach. Figma świetnie nadaje się także do tworzenia komponentów wielokrotnego użytku oraz ich wariantów. Komponenty te są również wykorzystywane w frameworku Next.js, dlatego uwzględnienie ich już na etapie projektowania ułatwia późniejszą implementację w kodzie.

Poniżej znajdują się założenia wizualne, które zostaną zastosowane podczas projektowania makiety aplikacji. Zdefiniowane zostały wartości takie jak:

- rozmiary czcionek dla odpowiednich elementów aplikacji,
- paleta kolorów,
- trzy warianty przycisków, główny, drugorzędny oraz trzeciorzędny,
- wygląd hiperłącz na stronie,
- przycisk do wybierania wersji językowej,
- warianty odstępów między elementami,
- pola tekstowe (ang. inputs) wraz ze swoim drugim wariantem,
- pola jednokrotnego wyboru.



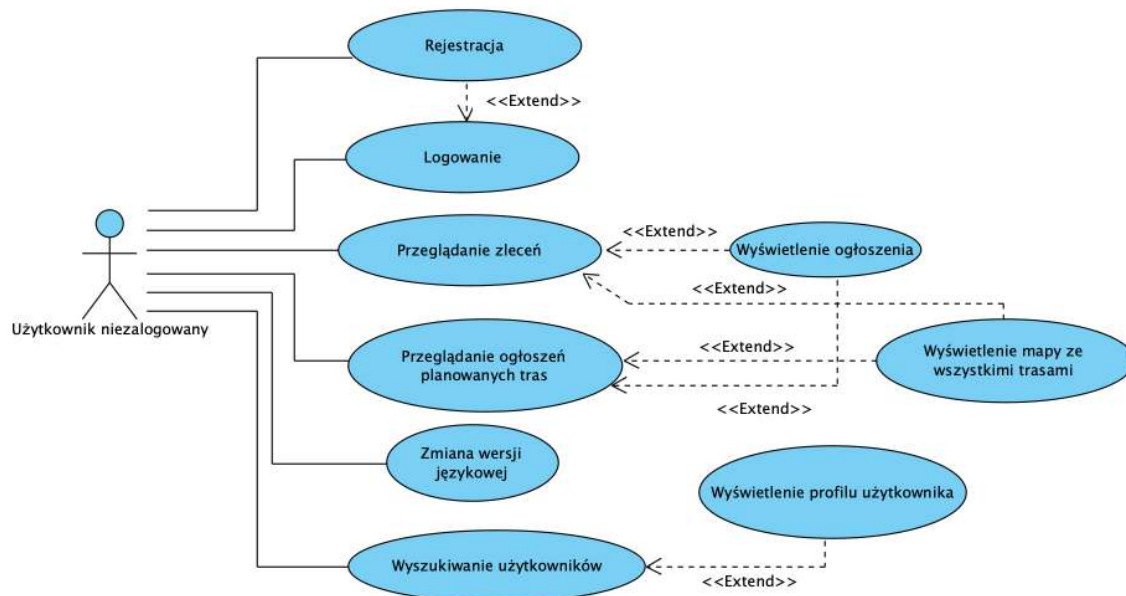
Rys. 2.1: Wybrane parametry dotyczące interfejsu użytkownika

Parametry te pozwolą na ujednolicenie wyglądu interfejsu, co przełoży się na lepsze odczucia podczas korzystania z aplikacji.

## 2.3. Diagramy przypadków użycia

W tym podrozdziale przedstawione zostaną diagramy przypadków użycia serwisu CargoLink, korzystając z definicji aktorów opisanych powyżej. Dodatkowo opisane i graficznie przedstawione zostaną wszystkie przypadki użycia użyte w diagramach.

### 2.3.1. Niezalogowany użytkownik



Rys. 2.2: Diagram przedstawiający przypadki użycia aktora Niezalogowany użytkownik

Na powyższym diagramie przedstawione zostały przypadki użycia dla Niezalogowanego użytkownika.

#### Rejestracja

Zdarzenie inicjujące: Kliknięcie przycisku Nie masz konta? Zarejestruj się.

Warunki początkowe: Użytkownik nie jest zalogowany.

Przebieg podstawowy realizacji przypadku użycia:

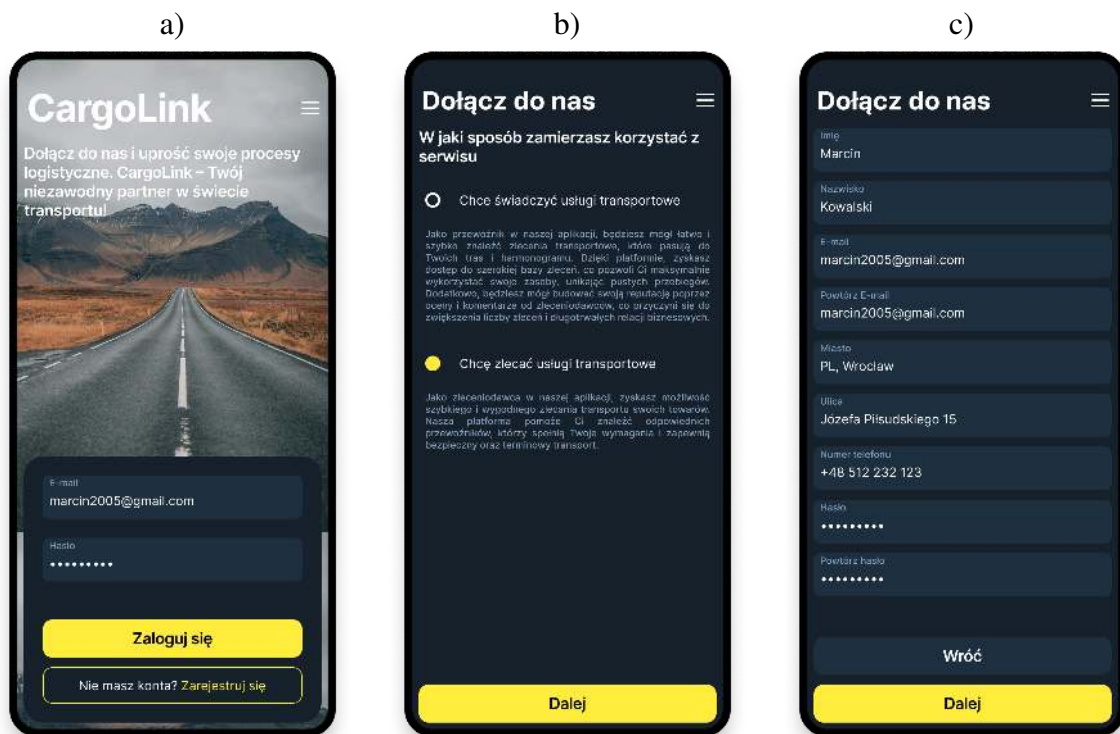
1. Kliknięcie przycisku Nie masz konta? Zarejestruj się (Rys. 2.3.a lub 2.4.a);
2. Wybór typu konta: zleceniodawca lub przewoźnik;
3. Kliknięcie przycisku dalej (Rys. 2.3.b lub 2.4.b);
4. Wpisanie danych:
  - imię,
  - nazwisko,
  - email (dwukrotnie w celu potwierdzenia),
  - adres, miasto i ulica (w celu generowania umów między użytkownikami),
  - numer telefonu,
  - hasło (dwukrotnie w celu potwierdzenia)
5. Kliknięcie przycisku Dalej; (Rys. 2.3.c lub 2.5)
6. System sprawdza w bazie danych czy istnieje już użytkownik o podanym emailu oraz sprawdza poprawność wprowadzonych danych;
7. Wybór czy konto ma reprezentować przedsiębiorstwo (wymagane będzie podane pełnej nazwy firmy, wraz z NIP'em oraz adresem siedziby), bądź osobę fizyczną;
8. Jeżeli wybrane zostało przedsiębiorstwo, system sprawdza poprawność wprowadzonych danych;
9. Kliknięcie przycisku Dalej; (Rys. 2.6.a lub 2.7.a)

10. Zaznaczenie języków, którymi użytkownik umie się posługiwać (informacje te pokazywać się będą w oknie czatu oraz na profilu, aby ułatwić użytkownikom porozumienie się);
11. Akceptacja regulaminu;
12. Kliknięcie przycisku Zarejestruj się; (Rys. 2.6.a lub 2.7.a)

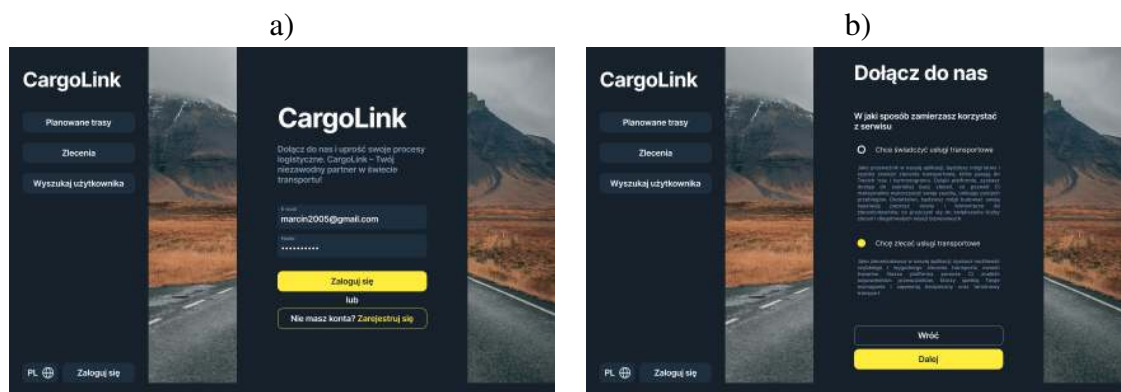
Przebieg alternatywny realizacji podpunktu (6a): W bazie danych istnieje już użytkownik o podanym emailu bądź użytkownik podał błędne dane. System informuje o niezgodności.

Przebieg alternatywny realizacji podpunktu (7a): Jeżeli konto ma reprezentować osobę fizyczną, pola do wpisania informacji o przedsiębiorstwie nie wyświetlą się.

Warunki końcowe: Dodanie utworzonego użytkownika do bazy, a następnie przekierowanie go na przeglądarkę ogłoszeń o planowanych trasach lub zleceniach, w zależności od typu konta.



Rys. 2.3: Formularz rejestracji w wersji mobilnej: a) Menu logowania, b) Wybór typu konta, c) Wprowadzenie danych do rejestracji



Rys. 2.4: Formularz rejestracji w wersji desktopowej: a) Menu logowania, b) Wybór typu konta

**CargoLink**

Planowane trasy

Zlecenia

Wyszukaj użytkownika

PL

Zaloguj się

**Dołącz do nas**

Imię: Marcin

Nazwisko: Kowalski

E-mail: marcin2005@gmail.com

Nowy E-mail: marcin2005@gmail.com

Hasło: .....

Nowe Hasło: .....

Wróć

Dalej

Rys. 2.5: Formularz rejestracji w wersji desktopowej: Wprowadzenie danych do rejestracji

**Dołącz do nas**

Zamierzasz korzystać z aplikacji jako przedsiębiorstwo czy osoba fizyczna?

Dane wpisane w tych polach będą używane, aby generować umowę wynagrodzenia dla Ciebie jak najbardziej jasno i precyzyjnie.

☒ Przedsiębiorstwo

Polna nazwa przedsiębiorstwa: Promatic SP. z o.o.

NIP: 1234123410

Miasto: PL, Wrocław

Ulica: Fabryczna 63

☐ Osoba fizyczna

Wróć

Dalej

**Dołącz do nas**

Jakie języki znasz?

Zaczęsz języki, którymi się posługujesz. Dedy one wywołane na Twoim profilu. Ułatwi to komunikację z innymi użytkownikami.

☐ Język polski

☐ English

☐ Deutsch

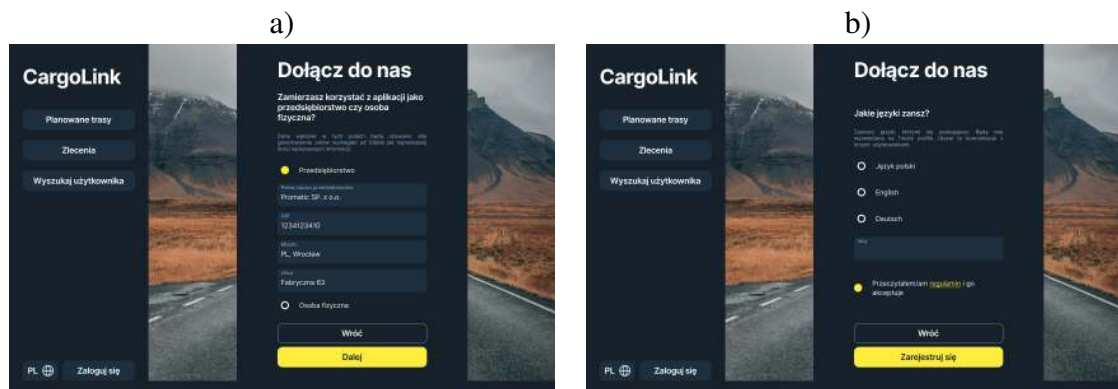
Inny: .....

☒ Przeczytałem/am regulamin i go akceptuję

Wróć

Zarejestruj się

Rys. 2.6: Formularz rejestracji w wersji mobilnej: a) Wybór reprezentacji konta, b) Wybór języków, którymi użytkownik się posługuje,



Rys. 2.7: Formularz rejestracji w wersji desktopowej: a) Wybór reprezentacji konta, b) Wybór języków, którymi użytkownik się posługuje,

### Logowanie

Zdarzenie inicjujące: Wpisanie danych logowania oraz kliknięcie przycisku **Zaloguj się**.

Warunki początkowe: Użytkownik nie jest zalogowany.

Przebieg podstawowy realizacji przypadku użycia:

1. Wpisanie danych, email oraz hasło;
2. Kliknięcie przycisku **Zaloguj się**; (Rys. 2.3.a lub 2.4.a)
3. System sprawdza w bazie danych czy istnieje już użytkownik o podanym emailu oraz czy wprowadzone hasło jest prawidłowe;
4. Zalogowanie użytkownika;
5. Przejście do wyszukiwarki ogłoszeń o planowanych trasach, bądź zleceń (w zależności od wybranego typu konta).

Przebieg alternatywny realizacji podpunktu (3a): W bazie danych nie znajduje się użytkownik o podanym emailu oraz hasle. System informuje o niepowodzeniu.

Warunki końcowe: W zależności od typu konta, przenosi do odpowiedniego miejsca w serwisie.

### Zmiana wersji językowej aplikacji

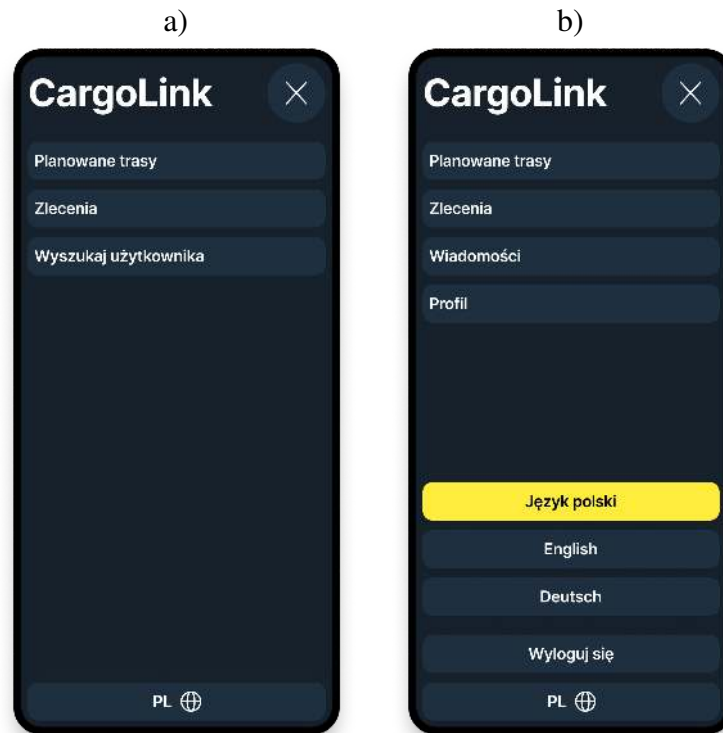
Zdarzenie inicjujące: Wersja mobilna aplikacji - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a). Wersja desktopowa aplikacji - kliknięcie przycisku z ikoną globu (np. Rys. 2.4.a).

Warunki początkowe: Brak.

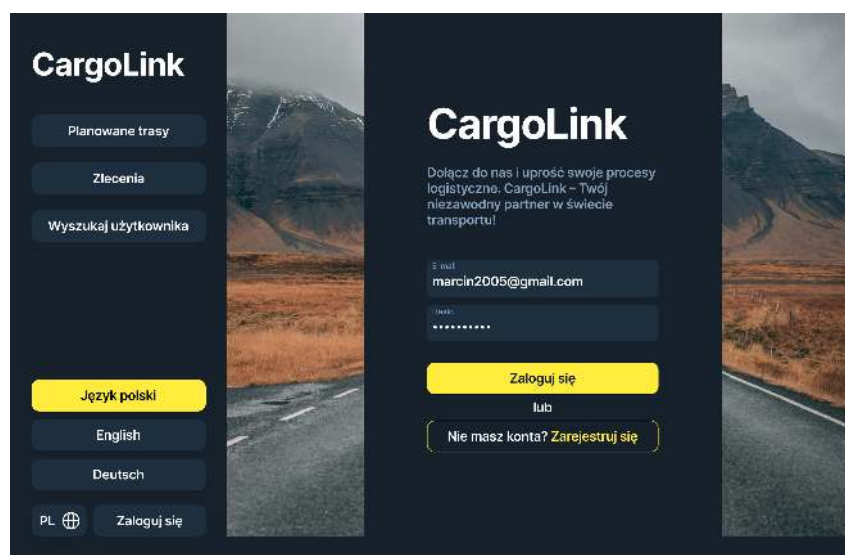
Przebieg podstawowy realizacji przypadku użycia:

1. Jeżeli w wersji mobilnej - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a);
2. Kliknięcie przycisku z ikoną globu (Rys. 2.8.a lub np. Rys. 2.5);
3. Wybranie jednego z trzech języków z menu (Rys. 2.8.b lub Rys. 2.9);
4. Zmianienie języka w jakim wyświetlana jest aplikacja;





Rys. 2.8: Menu nawigacji po aplikacji w wersji mobilnej: a) Menu nawigacji, b) Menu nawigacji z klikniętym przyciskiem wyboru języka



Rys. 2.9: Menu wyboru języka aplikacji w wersji desktopowej

### Wyszukiwanie użytkowników

Zdarzenie inicjujące: Wersja mobilna aplikacji - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a). Wersja desktopowa aplikacji - kliknięcie przycisku Wyszukaj użytkownika (np. Rys. 2.4.a).

Warunki początkowe: Brak.

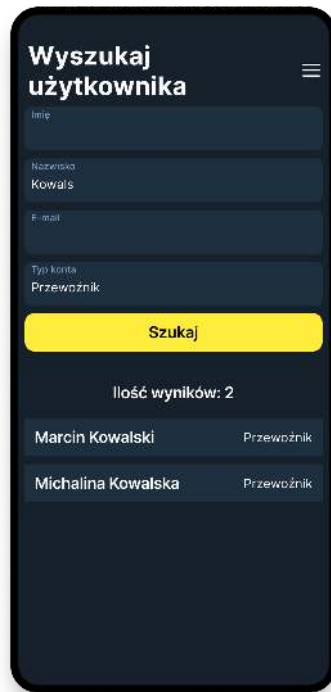
Przebieg podstawowy realizacji przypadku użycia:

1. Jeżeli w wersji mobilnej - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a);

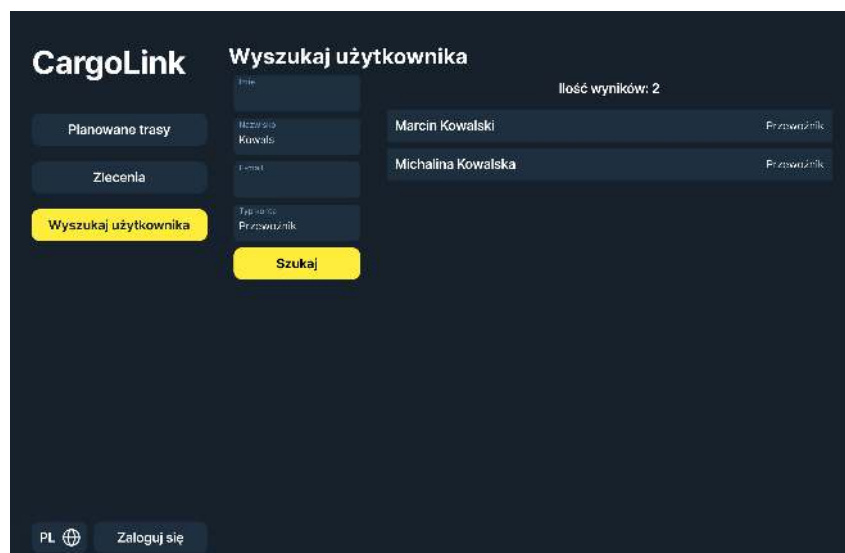


2. Kliknięcie przycisku Wyszukaj użytkownika (Rys. 2.8.a lub 2.4.a);
3. Wpisanie danych szukanego użytkownika, takich jak imię, nazwisko, e-mail lub typ konta (Rys. 2.10 lub 2.11);
4. Kliknięcie przycisku Szukaj (Rys. 2.10 lub 2.11);
5. Wyświetlenie wszystkich użytkowników spełniających podane kryteria.

Warunki końcowe: Serwis zwraca wszystkie profile spełniające wpisane wymagania.



Rys. 2.10: Wyszukiwanie użytkownika w wersji mobilnej



Rys. 2.11: Wyszukiwanie użytkownika w wersji desktopowej

### Przeglądanie zleceń

Zdarzenie inicjujące: Wersja mobilna aplikacji - kliknięcie w trzy poziome kreski w na-

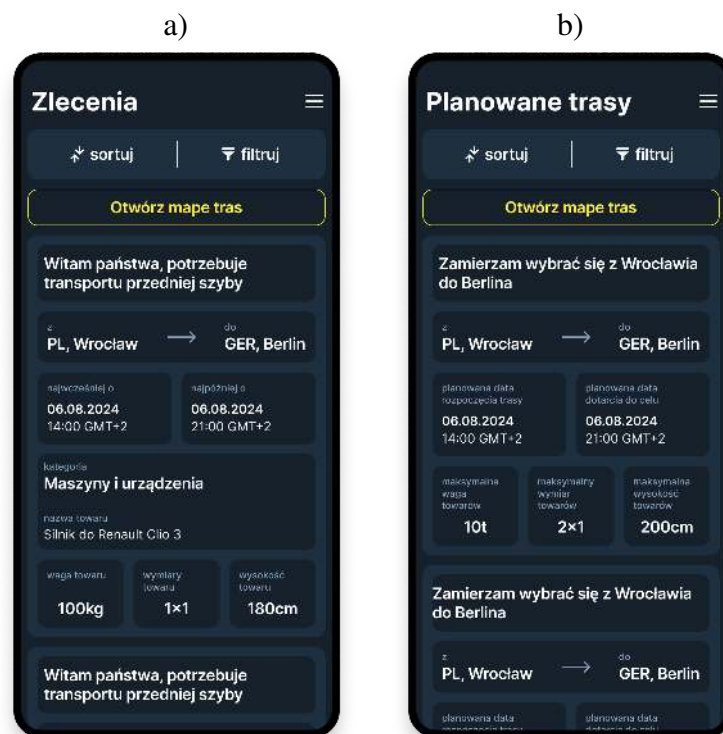
główku, aby otworzyć menu (np. Rys. 2.3.a). Wersja desktopowa aplikacji - kliknięcie przycisku Zlecenia (np. Rys. 2.4.a).

Warunki początkowe: Brak.

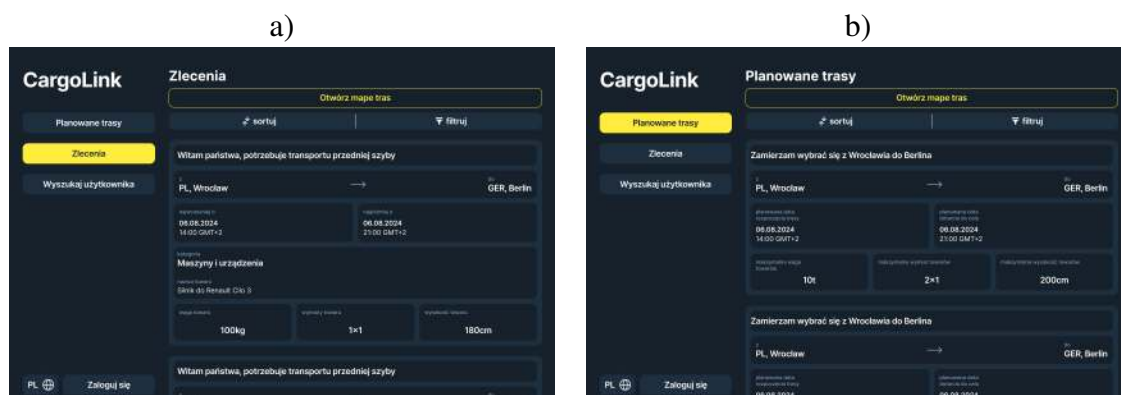
Przebieg podstawowy realizacji przypadku użycia:

1. Jeżeli w wersji mobilnej - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a);
2. Kliknięcie przycisku Zlecenia (Rys. 2.8.a lub 2.4.a);
3. Wyświetlenie listy ogłoszeń dodanych przez zleceniodawców.

Warunki końcowe: Wyświetlenie przeglądarki zleceń. (Rys. 2.12.a lub 2.13.a)



Rys. 2.12: Przeglądarka zleceń i planowanych tras w wersji mobilnej: a) Przeglądarka zleceń, b) Przeglądarka ogłoszeń planowanych tras



Rys. 2.13: Przeglądarka zleceń i planowanych tras w wersji desktopowej: a) Przeglądarka zleceń, b) Przeglądarka ogłoszeń planowanych tras

#### Przeglądanie ogłoszeń planowanych tras

Zdarzenie inicjujące: Wersja mobilna aplikacji - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a). Wersja desktopowa aplikacji - kliknięcie przycisku Zlecenia (np. Rys. 2.4.a).

Warunki początkowe: Brak.

Przebieg podstawowy realizacji przypadku użycia:

1. Jeżeli w wersji mobilnej - kliknięcie w trzy poziome kreski w nagłówku, aby otworzyć menu (np. Rys. 2.3.a);
2. Kliknięcie przycisku Planowane trasy (Rys. 2.8.a lub 2.4.a);
3. Wyświetlenie listy planowanych przez przewoźników tras.

Warunki końcowe: Wyświetlenie przeglądarki ogłoszeń o planowanych trasach. (Rys. 2.12.b lub 2.13.b)

#### Wyświetlenie ogłoszenia

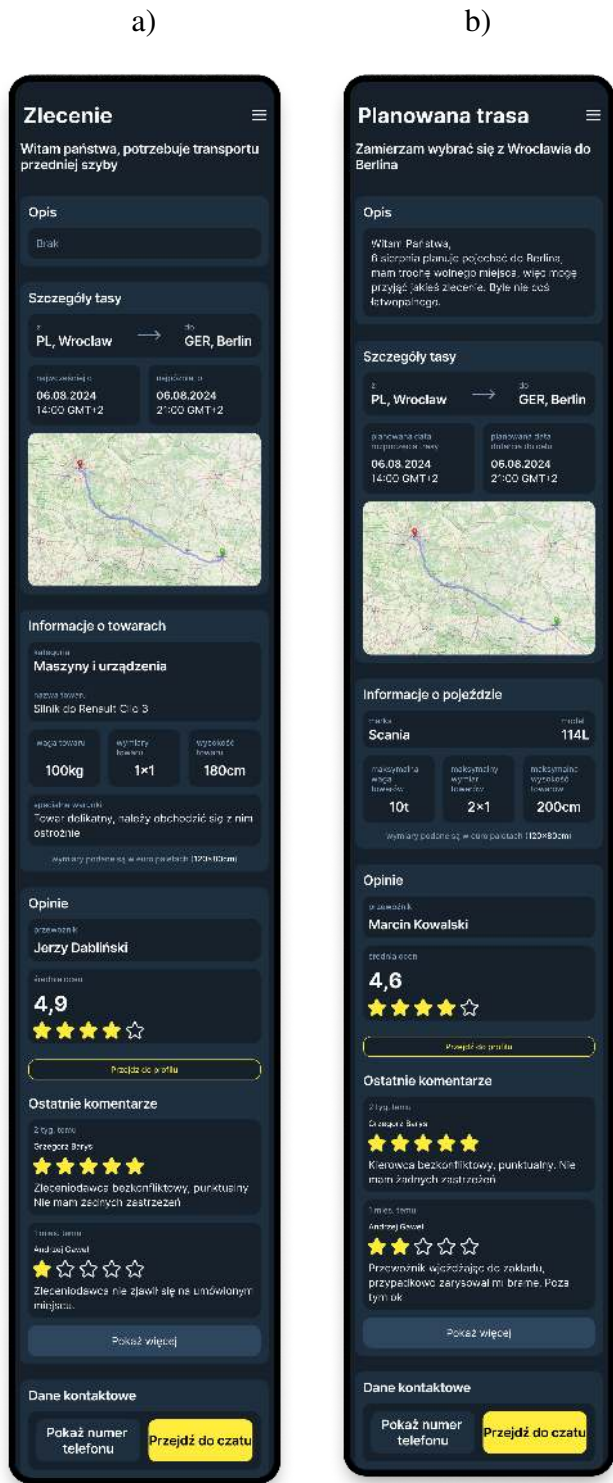
Zdarzenie inicjujące: Kliknięcie w dowolne ogłoszenie (Rys. 2.12.a lub 2.12.b lub 2.13.a lub 2.13.b).

Warunki początkowe: Brak.

Przebieg podstawowy realizacji przypadku użycia:

1. Wykonanie przypadku użycia 2.3.1 lub 2.3.1;
2. Kliknięcie w dowolne ogłoszenie;
3. Wyświetlenie klikniętego ogłoszenia.

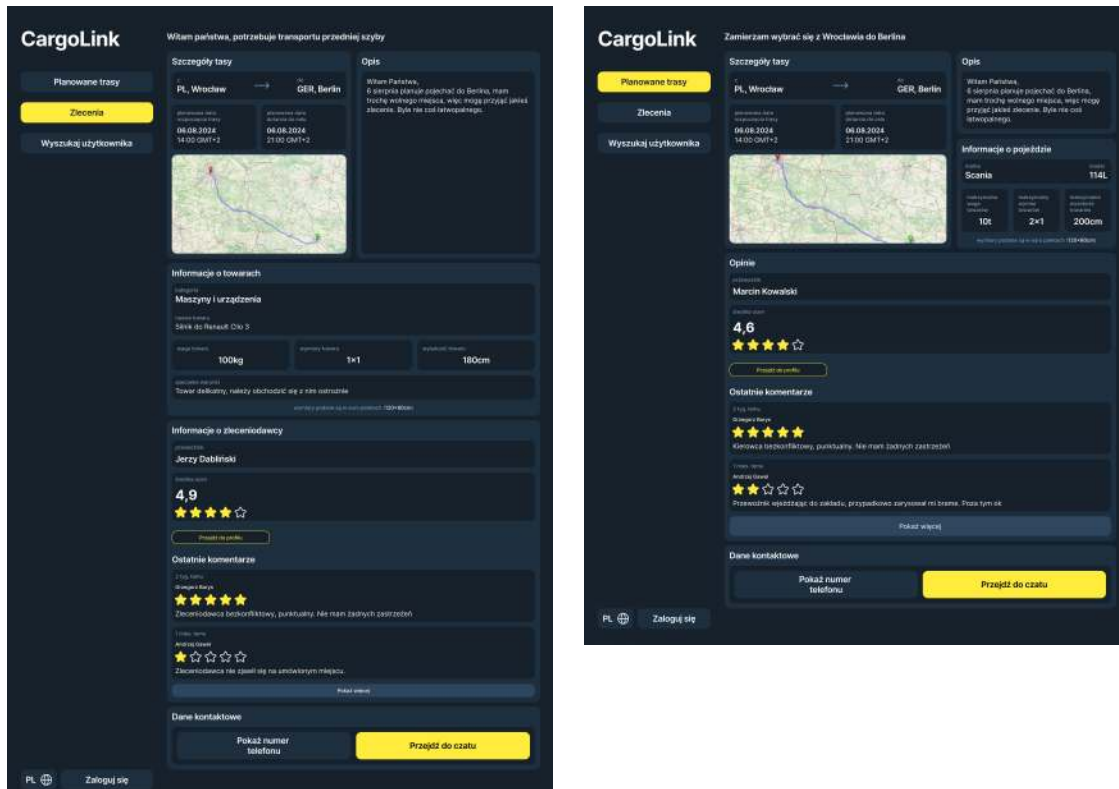
Warunki końcowe: Wyświetlenie klikniętego ogłoszenia. (Rys. 2.14.a lub 2.14.b lub 2.15.a lub 2.15.b)



Rys. 2.14: Wyświetlenie ogłoszenia w wersji mobilnej: a) Ogłoszenie zlecenia, b) Ogłoszenie o planowanej trasie

a)

b)



Rys. 2.15: Wyświetlenie ogłoszenia w wersji desktopowej: a) Ogłoszenie zlecenia, b) Ogłoszenie o planowanej trasie

#### Wyświetlenie mapy ze wszystkimi trasami

Zdarzenie inicjujące: Kliknięcie w przycisk **Otwórz mapę tras** (Rys. 2.12.a lub 2.12.b lub 2.13.a lub 2.13.b).

Warunki początkowe: Brak.

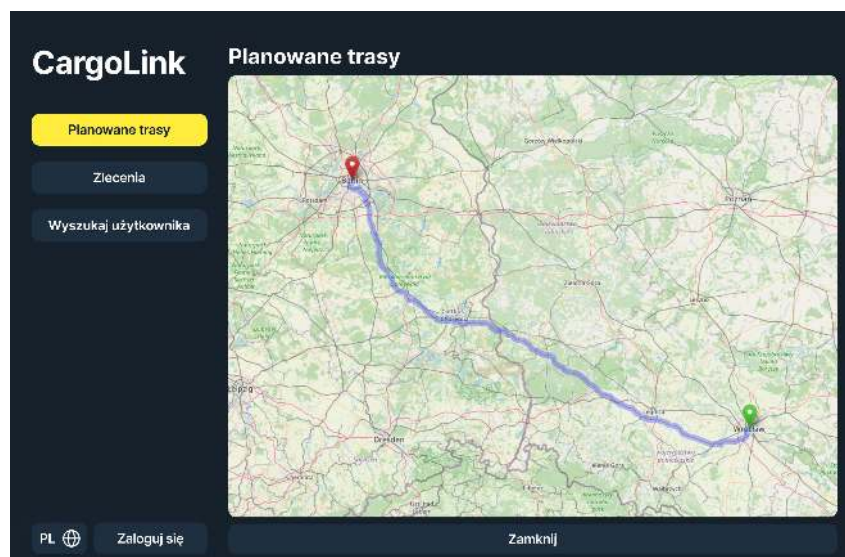
Przebieg podstawowy realizacji przypadku użycia:

1. Wykonanie przypadku użycia 2.3.1 lub 2.3.1;
2. Kliknięcie w przycisk **Otwórz mapę tras** (Rys. 2.12.a lub 2.12.b lub 2.13.a lub 2.13.b);
3. Wyświetlenie mapy z zaznaczonymi wszystkimi trasami w bazie danych, które są aktualne.

Warunki końcowe: Użytkownikowi ukazują się mapa z zaznaczonymi trasami wszystkich aktualnych ogłoszeń w bazie danych (Rys. 2.16 lub 2.17).



Rys. 2.16: Mapa ze wszystkimi trasami w wersji mobilnej



Rys. 2.17: Mapa ze wszystkimi trasami w wersji desktopowej

### Wyświetlenie profilu użytkownika

Zdarzenie inicjujące: Kliknięcie w dowolnego użytkownika (Rys. 2.12.a lub 2.12.b) lub kliknięcie w imię i nazwisko autora dowolnego ogłoszenia (Rys. 2.14.a lub 2.14.b lub 2.15.a lub 2.15.b)

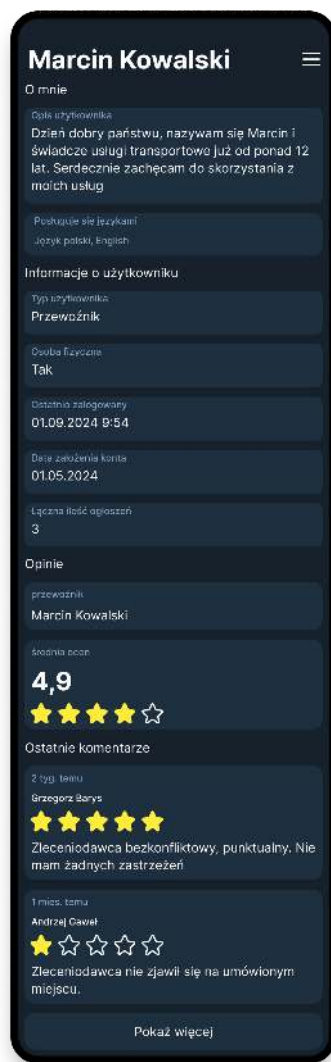
Warunki początkowe: Brak.

Przebieg podstawowy realizacji przypadku użycia:

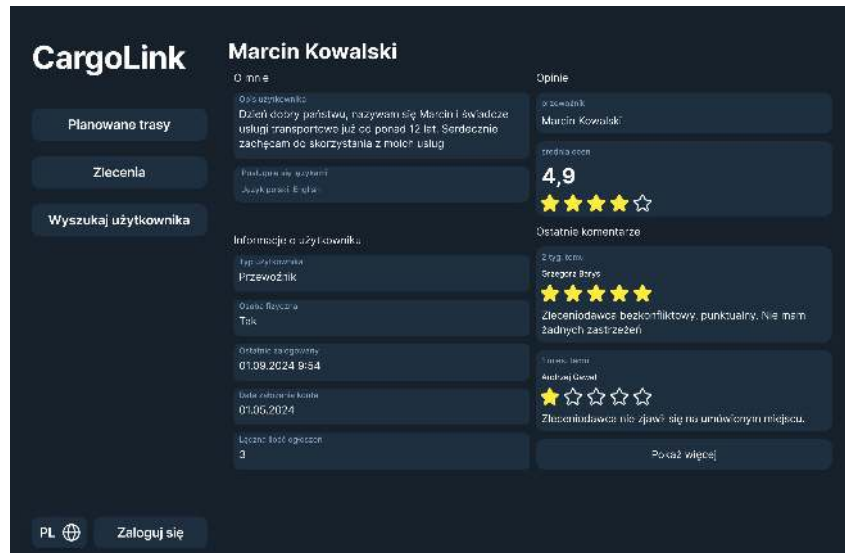
1. Kliknięcie w dowolnego użytkownika (Rys. 2.12.a lub 2.12.b) lub kliknięcie w imię i nazwisko autora dowolnego ogłoszenia (Rys. 2.14.a lub 2.14.b lub 2.15.a lub 2.15.b);
2. Wyświetlenie profilu wybranego użytkownika (Rys. 2.18 lub 2.19).



Warunki końcowe: Użytkownik przechodzi do strony profilu wybranego użytkownika (Rys. 2.18 lub 2.19).

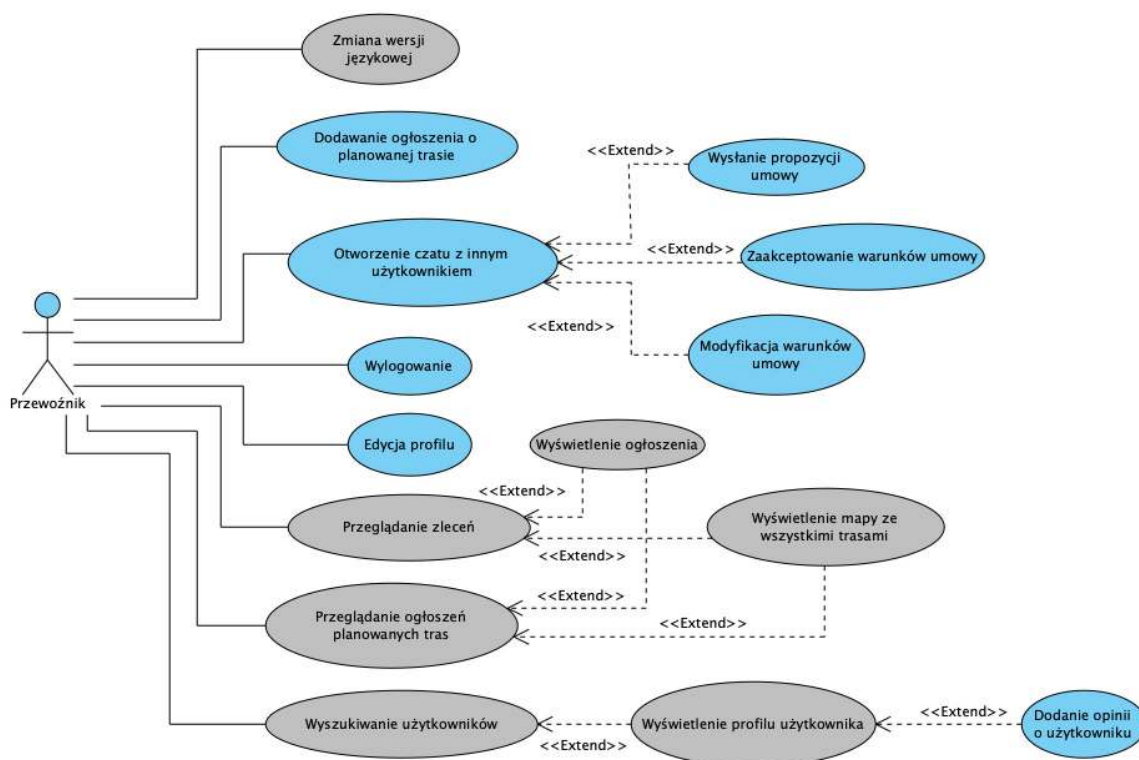


Rys. 2.18: Profil użytkownika w wersji mobilnej



Rys. 2.19: Profil użytkownika w wersji desktopowej

### 2.3.2. Przewoźnik



Rys. 2.20: Diagram przedstawiający przypadki użycia aktora Przewoźnik

Na powyższym diagramie przedstawione zostały przypadki użycia dla Przewoźnika. Kolorem szarym oznaczone zostały przypadki użycia, które zostały już opisane w poprzednich podsekcjach.



### Dodawanie nowego ogłoszenia o planowanej trasie

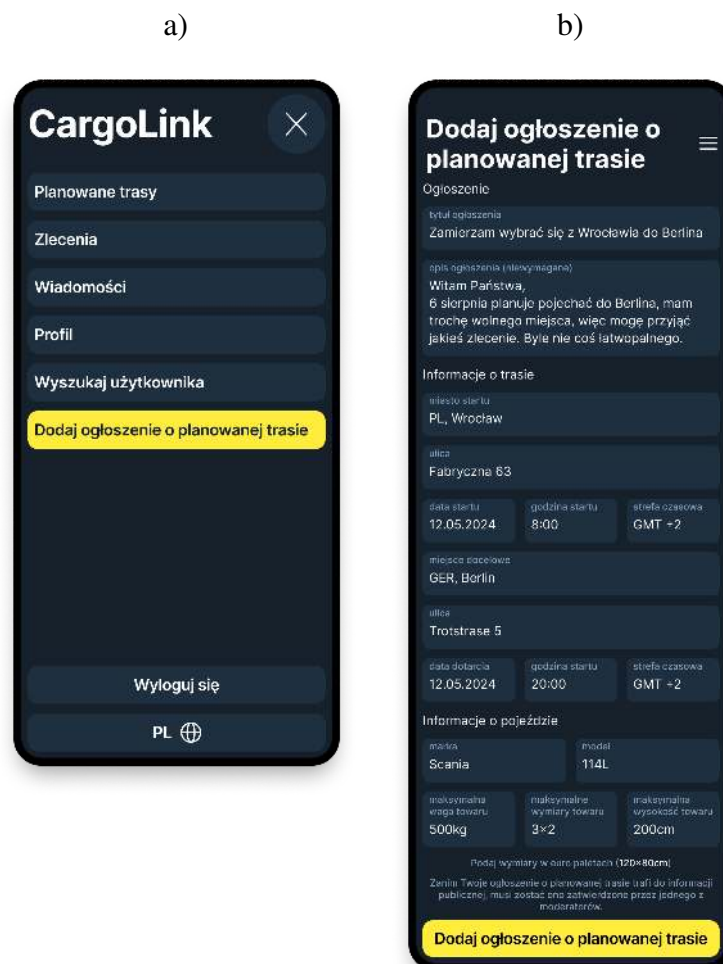
Zdarzenie inicjujące: Po zalogowaniu na konto z typem przewoźnik, do menu nawigacji dokładane jest klikka nowych opcji. Kliknięcie w przycisk Dodaj ogłoszenie o planowanej trasie (Rys. 2.21.a lub 2.22.a).

Warunki początkowe: Bycie zalogowanym jako przewoźnik.

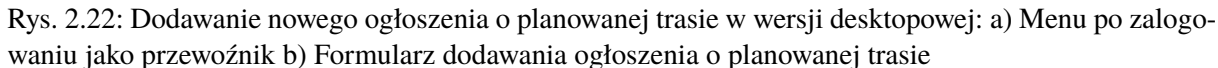
Przebieg podstawowy realizacji przypadku użycia:

1. Kliknięcie w przycisk Dodaj ogłoszenie o planowanej trasie (Rys. 2.21.a lub 2.22.a);
2. Wypełnienie formularza (Rys. 2.21.b lub 2.22.b);
3. Kliknięcie przycisku Dodaj ogłoszenie o planowanej trasie;
4. System sprawdza poprawność wprowadzonych danych;
5. Ogłoszenie wysyłane jest do akceptacji przez jednego z moderatorów.

Warunki końcowe: Ogłoszenie o planowanej trasie wysyłane jest do moderatorów w celu akceptacji. Przebieg alternatywny realizacji podpunktu (4a): Wprowadzone dane są niepoprawne. System informuje o niepowodzeniu.



Rys. 2.21: Dodawanie nowego ogłoszenia o planowanej trasie w wersji mobilnej: a) Menu po zalogowaniu jako przewoźnik b) Formularz dodawania ogłoszenia o planowanej trasie



Zdarzenie inicjujące: Kliknięcie w przycisk Przejdź do czatu (Rys. 2.14.a lub 2.14.b lub 2.15.a lub 2.15.b)

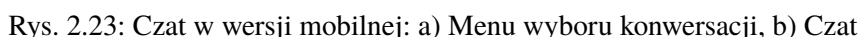
### Przebieg podstawowy realizacji przypadku użycia:

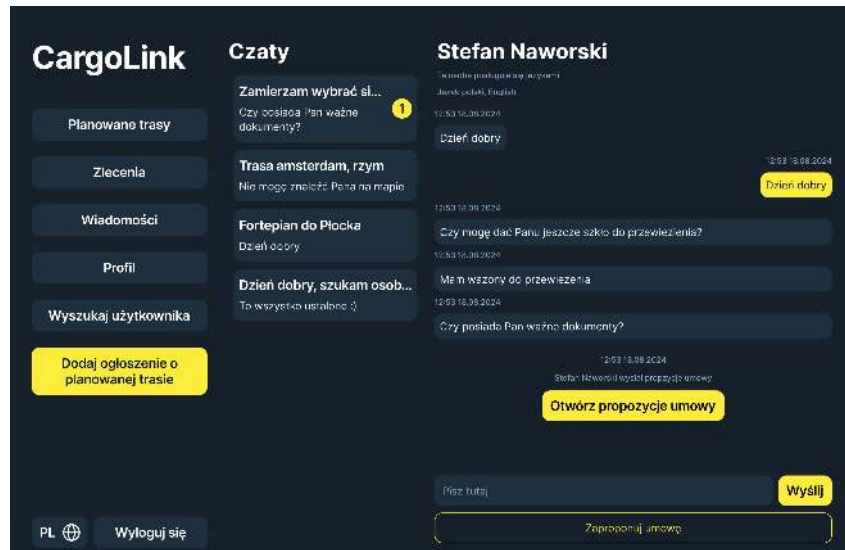
1. Kliknięcie w przycisk **Przejdź do czatu** (Rys. 2.14.a lub 2.14.b lub 2.15.a lub 2.15.b);
2. Otworzenie czatu z autorem ogłoszenia (Rys. 2.23.b lub 2.24).

Warunki końcowe: Użytkownik przechodzi do konwersacji z wybranym użytkownikiem (Rys. 2.23.b lub 2.24)).

### Przebieg alternatywny realizacji przypadku użycia:

1. Kliknięcie w przycisk Wiadomości (Rys. 2.8.a lub 2.14.b lub 2.15.a lub 2.15.b);
2. Otworzenie czatu z autorem ogłoszenia (Rys. 2.23.b lub 2.24).





Rys. 2.24: Czat w wersji desktopowej

### Wysyłanie propozycji umowy

Zdarzenie inicjujące: Kliknięcie przycisku **Zaproponuj umowę**.

Warunki początkowe: Użytkownik jest zalogowany.

Przebieg podstawowy realizacji przypadku użycia:

1. Wykonanie przypadku użycia 2.3.2;
2. Kliknięcie przycisku **Zaproponuj umowę** (Rys. 2.23.b lub 2.24);
3. Wypełnienie formularza (Rys. 2.25 lub 2.26);
4. System sprawdza poprawność wprowadzonych danych;
5. System wysyła propozycje umowy do rozmówcy.

Warunki końcowe: System wysyła propozycje umowy do rozmówcy.

Przebieg alternatywny w realizacji podpunktu (4a): Wprowadzone przez użytkownika dane są nieprawidłowe, system informuje o błędach w formularzu.

**Wygeneruj umowę**

Sprawdź poprawność danych, możesz je w tym miejscu edytować

**Zleceniodawca**

pełna nazwa firmy  
Promatic SP. z o.o.

NIP  
1234123423

miasto  
PL, Wrocław

ulica  
Józefa Piłsudskiego 15

**Przewoźnik**

imię i nazwisko  
Marcin Kowalski

miasto  
PL, Wrocław

ulica  
Józefa Piłsudskiego 15

**Przedmiot umowy**

kategoria  
Maszyny i urządzenia

nazwa towaru  
Silnik do Renault Clio 3

waga towaru  
100kg

wymiary towaru  
1x1

wysokość towaru  
180cm

**Informacje o trasie**

z  
PL, Wrocław  
Józefa Piłsudskiego 15

do  
GER, Berlin  
Trotstrasse 5

napływający o  
06.08.2024  
14:00 GMT+2

opływający o  
14.08.2024  
21:00 GMT+2

**Wyślij propozycje umowy**

Rys. 2.25: Formularz generujący umowę w wersji mobilnej

**CargoLink**

Planowane trasy

Zlecenia

Wiadomości

Profil

Wyszukaj użytkownika

**Dodaj ogłoszenie o planowanej trasie**

**Wygeneruj umowę**

Sprawdź poprawność danych, możesz je w tym miejscu edytować

**Zleceniodawca**

pełna nazwa firmy  
Promatic SP. z o.o.

NIP  
1234123423

miasto  
PL, Wrocław

ulica  
Józefa Piłsudskiego 15

**Przewoźnik**

imię i nazwisko  
Marcin Kowalski

miasto  
PL, Wrocław

ulica  
Józefa Piłsudskiego 15

**Przedmiot umowy**

kategoria  
Maszyny i urządzenia

nazwa towaru  
Silnik do Renault Clio 3

waga towaru  
100kg

wymiary towaru  
1x1

wysokość towaru  
180cm

**Informacje o trasie**

z  
PL, Wrocław  
Józefa Piłsudskiego 15

do  
GER, Berlin  
Trotstrasse 5

napływający o  
06.08.2024  
14:00 GMT+2

opływający o  
14.08.2024  
21:00 GMT+2

**Wyślij propozycje umowy**

PL Wyloguj się

Rys. 2.26: Formularz generujący umowę w wersji desktopowej

**Zaakceptowanie warunków umowy**

Zdarzenie inicjujące: Kliknięcie przycisku **Otwórz propozycje umowy**

Warunki początkowe: Bycie zalogowanym, rozmówca musi wysłać propozycje umowy

Przebieg podstawowy realizacji przypadku użycia:

1. kliknięcie przycisku **Otwórz propozycje umowy** (Rys. 2.23.b lub 2.24);
2. kliknięcie przycisku **Akceptuje warunki** (Rys. 2.27 lub 2.28).

Warunki końcowe: System generuje gotowy do podpisania dokument w formacie PDF.

**Potwierdź warunki umowy**

Stefan Naworski wysłał propozycje umowy, możesz edytować dane i odesłać swoją propozycje umowy

**Zleceńiodawca**

pełna nazwa firmy  
Promatic SP. z o.o.

NIP  
1234123423

miasto  
PL, Wrocław

ulica  
Józefa Piłsudskiego 15 **ZMIENIONO**

**Przewoźnik**

imię i nazwisko  
Marcin Kowalski

miasto  
PL, Wrocław

ulica  
Józefa Piłsudskiego 15

**Przedmiot umowy**

kategoria  
Maszyny i urządzenia

nazwa towaru  
Silnik do Renault Clio 3

waga towaru  
100kg

wymiary towaru  
1x1

wysokość towaru  
180cm

**Informacje o trasie**

z  
PL, Wrocław  
Józefa Piłsudskiego 15

do  
GER, Berlin  
Trotstrase 5

najwcześniej o  
06.08.2024  
14:00 GMT+2

najpóźniej o  
14.08.2024  
21:00 GMT+2

Zaproponuj nową umowę

**Akceptuje warunki**

Rys. 2.27: Zaakceptowanie warunków umowy w wersji mobilnej

**CargoLink**

**Potwierdź warunki umowy**

Stefan Naworski wysłał propozycję umowy, możesz edytować dane i odesłać swoją propozycję umowy

**Zleceniodawca**

Imię i nazwisko: Rafał  
Firma: ABC Sp. z o.o.

**NIP:**  
1234123423

**Adres:**  
PL, Wrocław  
Józefa Piłsudskiego 15

**Przewoźnik**

Imię i nazwisko: Marcin Kowalski

**Adres:**  
PL, Wrocław  
Józefa Piłsudskiego 15

**Przedmiot umowy**

Kategoria: Maszyny i urządzenia

Nazwa towaru: Silnik do Renault Clio 3

**Waga towaru:** 100kg **Wymiary towaru:** 1x1 **Wysokość towaru:** 180cm

**Informacja o trasie**

PL, Wrocław  
Józefa Piłsudskiego 15 → GER, Berlin  
Tobolskie 5

**Wysokość towaru:** 14.08.2024 21:00 GMT+2

**Waga towaru:** 06.08.2024 14:00 GMT+2

**Zaproponuj nową umowę**

**Wyślij propozycję umowy**

**Wyloguj się**

Rys. 2.28: Zaakceptowanie warunków umowy w wersji mobilnej

### Modyfikacja warunków umowy

Zdarzenie inicjujące: Kliknięcie przycisku **Otwórz propozycję umowy**

Warunki początkowe: Bycie zalogowanym, rozmówca musi wysłać propozycję umowy

Przebieg podstawowy realizacji przypadku użycia:

1. kliknięcie przycisku **Otwórz propozycję umowy** (Rys. 2.23.b lub 2.24);
2. użytkownik zmienia wartości w formularzu;
3. kliknięcie przycisku **Zaproponuj nową umowę** (Rys. 2.27 lub 2.28).

Warunki końcowe: Do rozmówcy wysyłana jest nowa wersja umowy wraz z zaznaczonymi wartościami, które uległy zmianie.

Przebieg alternatywny w realizacji podpunktu (3a): Wprowadzone dane są niepoprawne, system informuje o niepowodzeniu.

### Wylogowanie

Zdarzenie inicjujące: kliknięcie przycisku **Wyloguj się** (Rys. 2.21.a lub np. 2.22.a).

Warunki początkowe: Użytkownik jest zalogowany.

Przebieg podstawowy realizacji przypadku użycia:

1. kliknięcie przycisku **Wyloguj się** (Rys. 2.21.a lub np. 2.22.a).

Warunki końcowe: Wylogowanie użytkownika.

### Edycja profilu

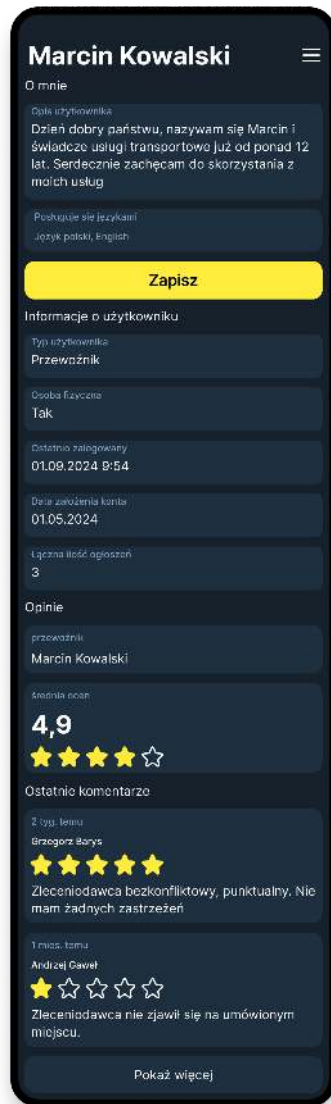
Zdarzenie inicjujące: Kliknięcie przycisku **Profil**

Warunki początkowe: Bycie zalogowanym

Przebieg podstawowy realizacji przypadku użycia:

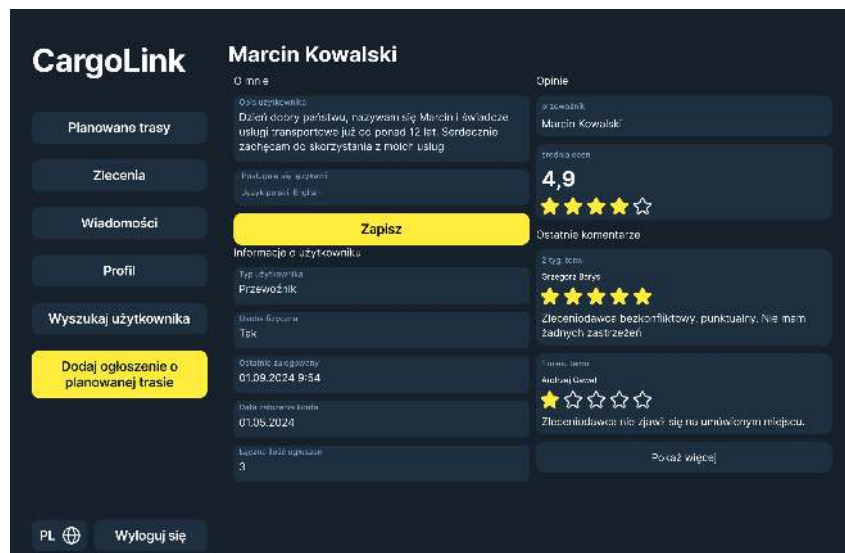
1. Zdarzenie inicjujące: Kliknięcie przycisku **Profil** (Rys. 2.21.a lub np. 2.22.b);
2. Zmiana danych na profilu (Rys. 2.29 lub 2.29);
3. Kliknięcie przycisku **Zapisz**.

Warunki końcowe: zapisanie w bazie danych zmienionych danych.



Rys. 2.29: Edycja profilu w wersji mobilnej





Rys. 2.30: Edycja profilu w wersji mobilnej

### Dodanie opinii o użytkowniku

Zdarzenie inicjujące: Wykonanie przypadku użycia 2.3.1

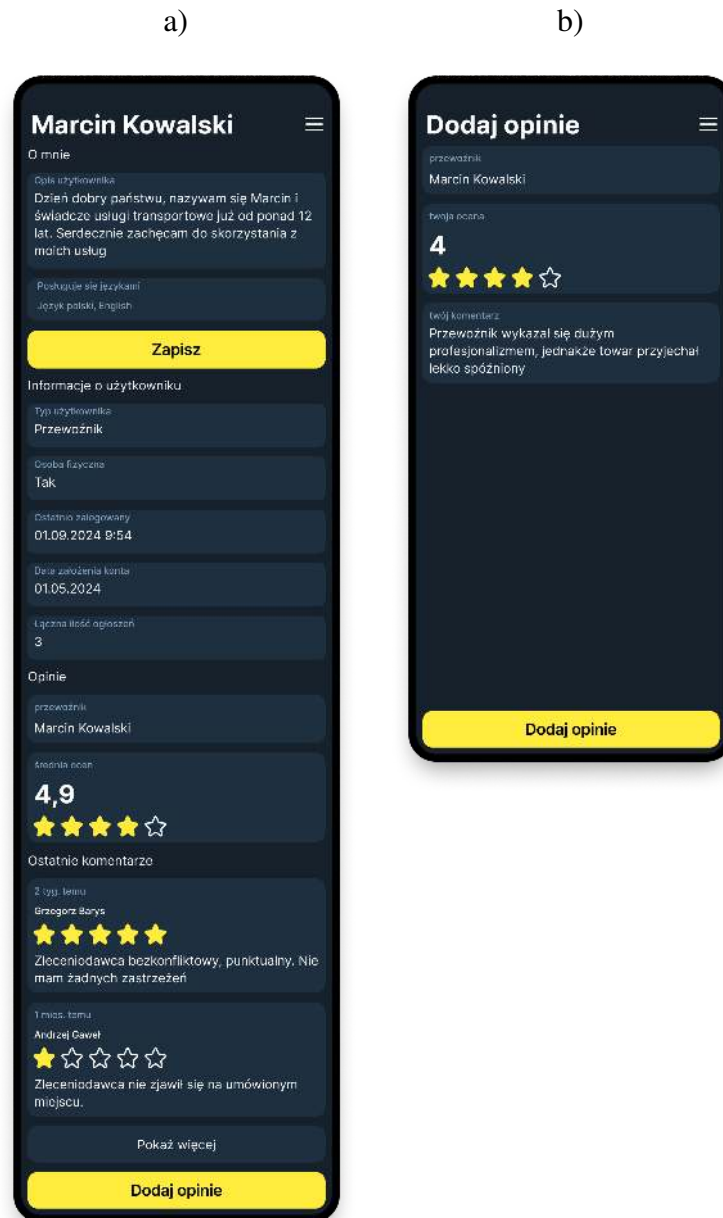
Warunki początkowe: Od wygenerowania umowy między użytkownikami musi minąć 7 dni.

Przebieg podstawowy realizacji przypadku użycia:

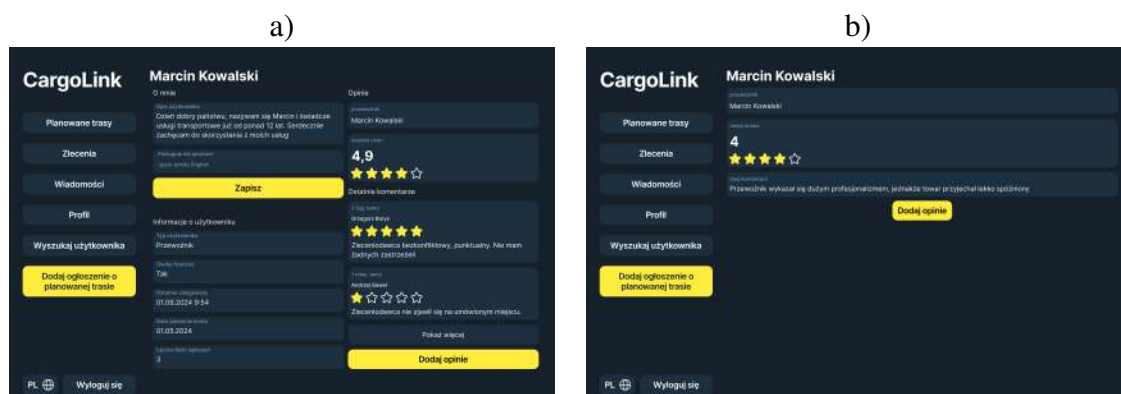
1. Wykonanie przypadku użycia 2.3.1;
2. Gdy dodanie opinii będzie możliwe, pojawi się nowy przycisk Dodaj opinie (Rys. 2.31.a lub 2.32.a);
3. Wypełnienie formularza dodawania opinii (Rys. 2.31.b lub 2.32.b);
4. Dodanie opinii o użytkowniku.

Warunki końcowe: Dodanie opinii o użytkowniku



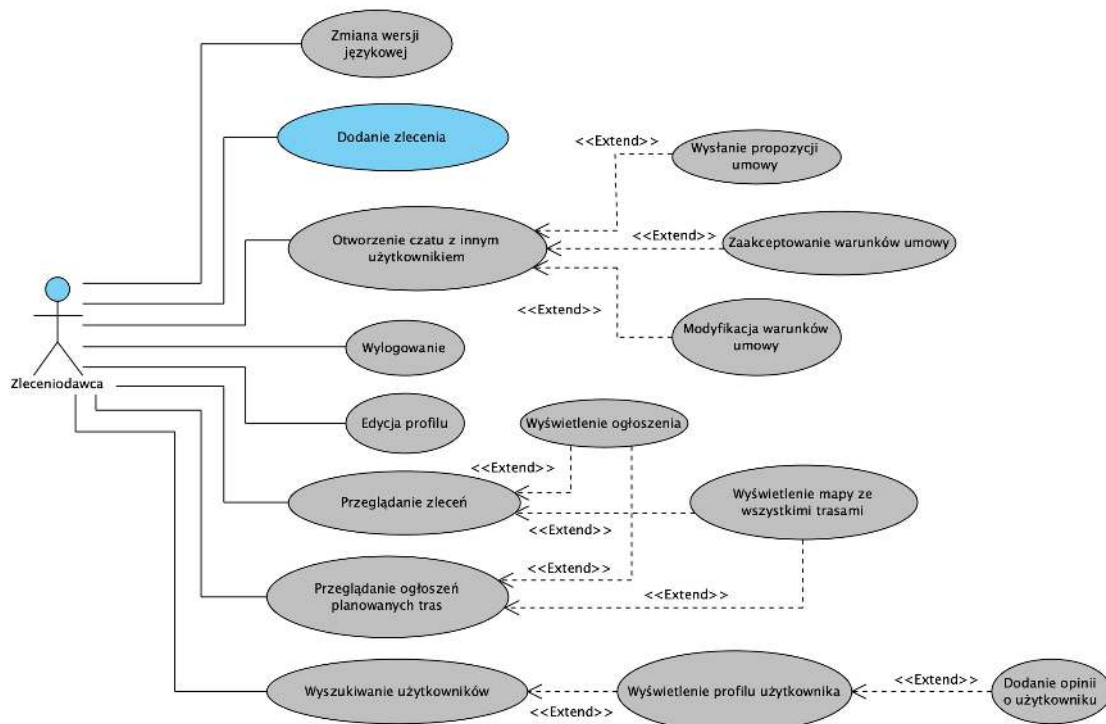


Rys. 2.31: Dodawanie opinii o użytkowniku w wersji mobilnej: a) Profil użytkownika gdy dostępne jest dodanie opinii b) Formularz dodawania opinii



Rys. 2.32: Dodawanie opinii o użytkowniku w wersji desktopowej: a) Profil użytkownika gdy dostępne jest dodanie opinii b) Formularz dodawania opinii

## 2.3.3. Zleceniodawca



Rys. 2.33: Diagram przedstawiający przypadki użycia aktora Zleceniodawca

Na powyższym diagramie przedstawione zostały przypadki użycia dla Zleceniodawcy. Kolorem szarym oznaczone zostały przypadki użycia, które zostały już opisane w poprzednich podsekcjach.

## Dodawanie nowego zlecenia

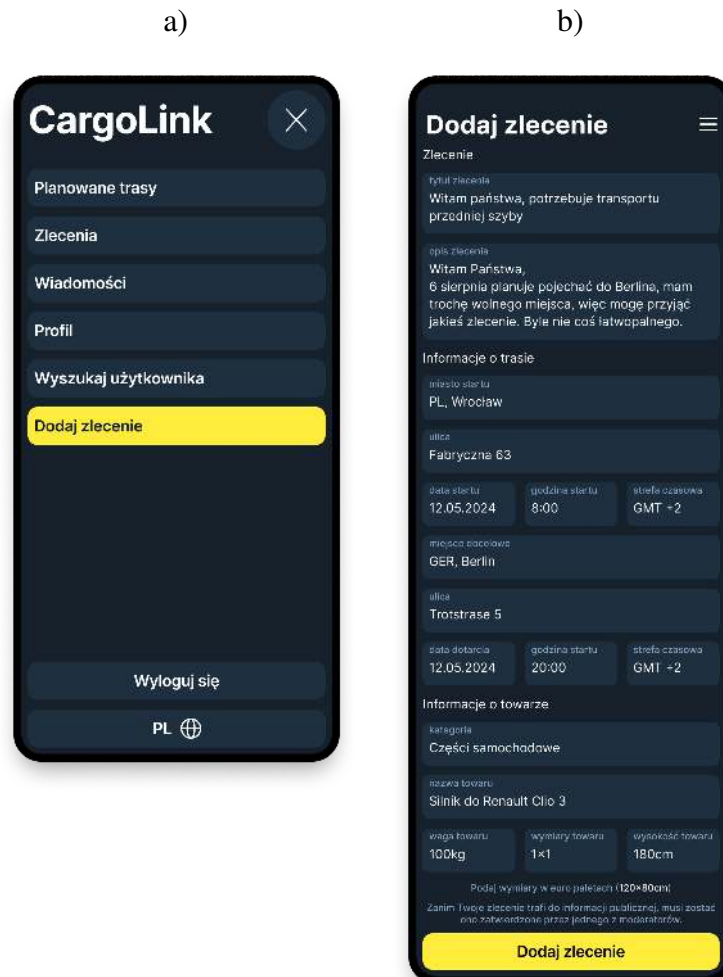
Zdarzenie inicjujące: Po zalogowaniu na konto z typem zleceniodawca, do menu nawigacji dokładane jest kliknięcie w przycisk Dodaj zlecenie (Rys. 2.34.a lub 2.35.a).

Warunki początkowe: Bycie zalogowanym jako zleceniodawca.

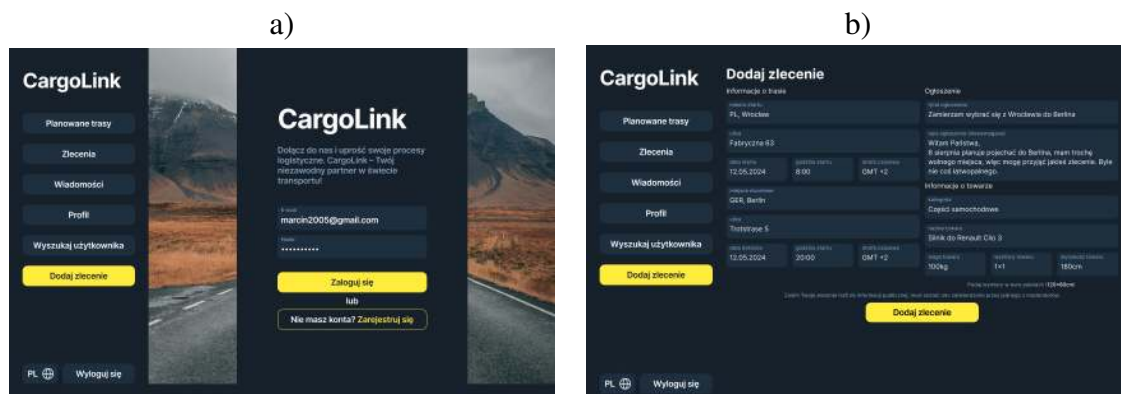
Przebieg podstawowy realizacji przypadku użycia:

1. Kliknięcie w przycisk Dodaj zlecenie (Rys. 2.34.a lub 2.35.a);
2. Wypełnienie formularza (Rys. 2.34.b lub 2.35.b);
3. Kliknięcie przycisku Dodaj zlecenie;
4. System sprawdza poprawność wprowadzonych danych;
5. Zlecenie wysyłane jest do akceptacji przez jednego z moderatorów.

Warunki końcowe: Zlecenie wysyłane jest do moderatorów w celu akceptacji. Przebieg alternatywny realizacji podpunktu (4a): Wprowadzone dane są niepoprawne. System informuje o niepowodzeniu.

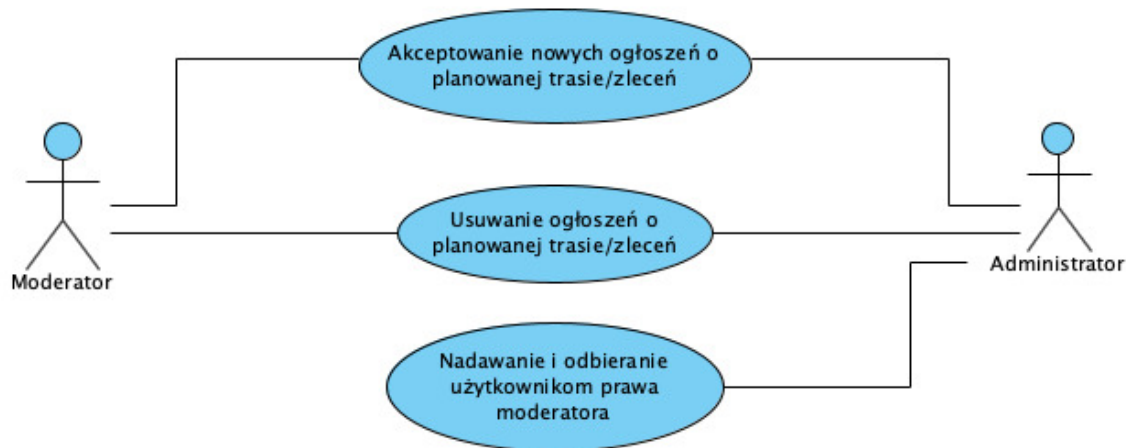


Rys. 2.34: Dodawanie nowego zlecenia w wersji mobilnej: a) Menu po zalogowaniu jako zleceniodawca b) Formularz dodawania zlecenia



Rys. 2.35: Dodawanie nowego zlecenia w wersji desktopowej: a) Menu po zalogowaniu jako zleceniodawca b) Formularz dodawania zlecenia

### 2.3.4. Moderator i Administrator



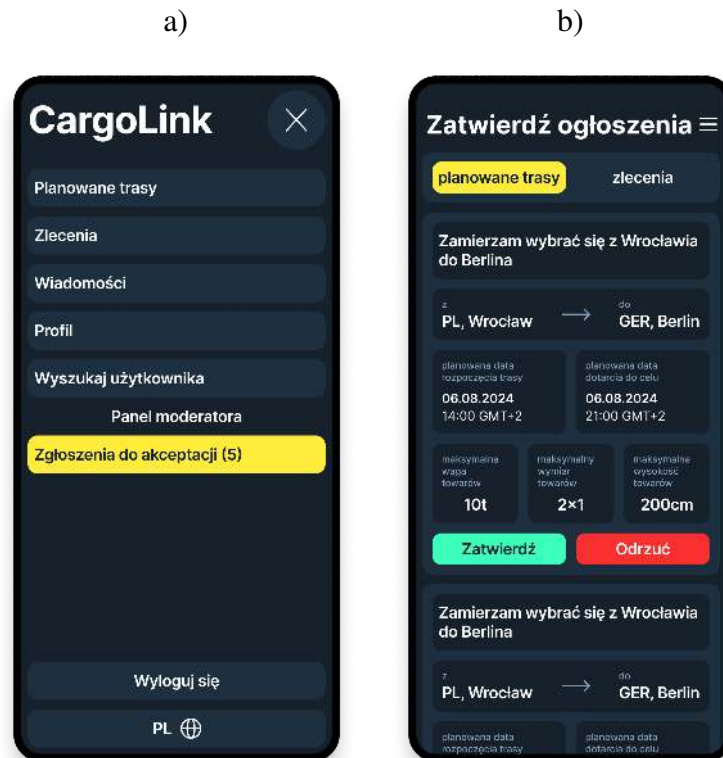
Rys. 2.36: Diagram przedstawiający przypadki użycia aktorów Moderator i Administrator

Na powyższym diagramie przedstawione zostały przypadki użycia dla Moderadora oraz Administratora.

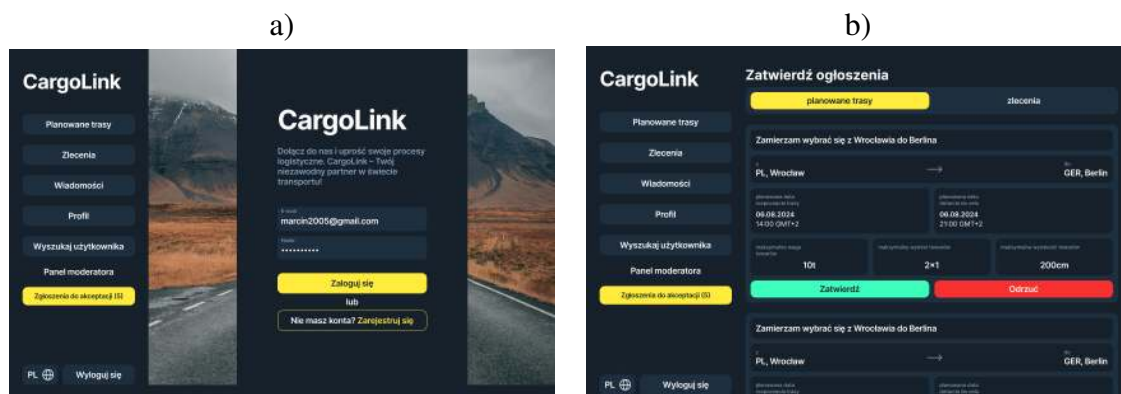
**Akceptacja nowych zleceń oraz ogłoszeń o planowanej trasie**  
 Zdarzenie inicjujące: Kliknięcie w przycisk Zgłoszenia do akceptacji (x).  
 Warunki początkowe: Bycie zalogowanym jako moderator lub administrator.  
 Przebieg podstawowy realizacji przypadku użycia:

1. Kliknięcie w przycisk Zgłoszenia do akceptacji (x) (Rys. 2.37.a lub 2.38.a);
2. Użytkownikowi wyświetla się lista oczekujących zgłoszeń;
3. Osoba akceptująca zatwierdza lub odrzuca zgłoszenia klikając w odpowiednie przyciski (Rys. 2.37.b lub 2.38.b).

Warunki końcowe: Ogłoszenie zostaje odrzucone bądź zaakceptowane.



Rys. 2.37: Akceptacja nowych zleceń i ogłoszeń o planowanej trasie w wersji mobilnej: a) Menu po zalogowaniu jako moderator lub administrator b) Panel akceptacji nowych ogłoszeń



Rys. 2.38: Akceptacja nowych zleceń i ogłoszeń o planowanej trasie w wersji desktopowej: a) Menu po zalogowaniu jako moderator lub administrator b) Panel akceptacji nowych ogłoszeń

**Nadawanie i odbieranie użytkownikom prawa moderatora**

Zdarzenie inicjujące: Wykonanie przypadku użycia 2.3.1.

Warunki początkowe: Bycie zalogowanym jako administrator.

Przebieg podstawowy realizacji przypadku użycia:

1. Wykonanie przypadku użycia 2.3.1;
2. Kliknięcie na pole Typ użytkownika rozwija listę z opcjami Przewoźnik, Zleceniodawca, Moderator, Administrator (Rys. 2.18 lub 2.19);
3. Typ użytkownika zostaje zmieniony na wybraną opcję.

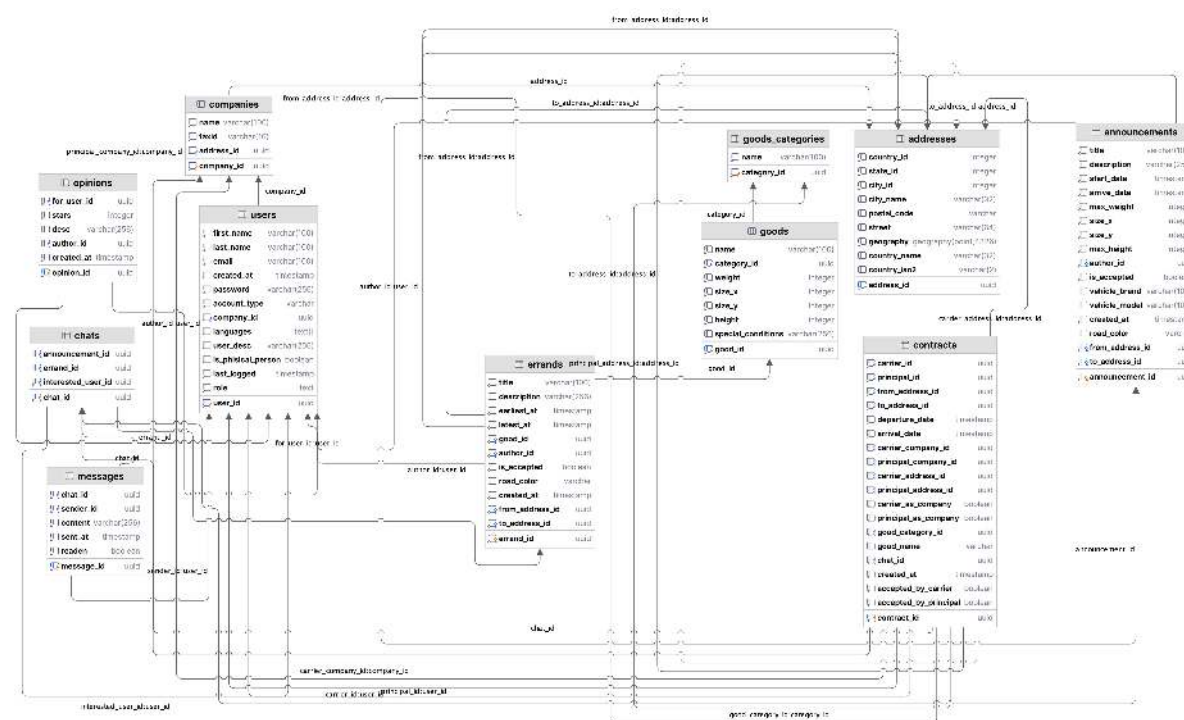
Warunki końcowe: Typ użytkownika zostaje zmieniony na wybraną opcję.



## 2.4. Projekt bazy danych

Baza danych została wykonana w języku PostgreSQL [4]. Jest to zaawansowany, open-source'owy system zarządzania bazami danych. Między innymi jest znany ze swojej elastyczności, skalowalności i zgodności ze standardami SQL.

Diagram bazy danych został wygenerowany przez program DataGrip. DataGrip to zaawansowane narzędzie do zarządzania bazami danych, stworzone przez JetBrains. Jest to wieloplatformowe środowisko IDE (zintegrowane środowisko programistyczne), które obsługuje wiele różnych systemów bazodanowych, takich jak PostgreSQL, MySQL, Oracle, SQL Server, SQLite i inne. Jedną z jego głównych zalet jest inteligentny edytor SQL, który oferuje funkcje automatycznego uzupełniania kodu, podpowiedzi dotyczących zapytań oraz wykrywania błędów w czasie rzeczywistym. DataGrip umożliwia również łatwe przeglądanie struktury baz danych, poprzez graficzny interfejs. Narzędzie jest szczególnie cenione przez deweloperów i administratorów baz danych za swoją intuicyjność.



Rys. 2.39: Projekt bazy danych

Dla tabel zastosowano nazewnictwo w języku angielskim, co jest dobrą praktyką podczas pracy w zespołach międzynarodowych, ułatwiając współpracę i zrozumienie struktury danych. Klucze główne we wszystkich tabelach są typu UUID, co jest nowoczesnym podejściem do identyfikacji wierszy w bazach danych. W odróżnieniu od tradycyjnych kluczy głównych opartych na autoinkrementujących liczbach całkowitych, UUID są unikalnymi identyfikatorami, które mogą być generowane niezależnie w różnych systemach bez ryzyka konfliktów. Jest to szczególnie istotne w systemach rozproszonych, gdzie wiele serwerów może dodawać dane równocześnie. Co więcej, UUID zwiększa bezpieczeństwo, ponieważ trudno jest przewidzieć kolejny identyfikator, co może mieć znaczenie w aplikacjach wymagających większej prywatności i bezpieczeństwa danych.

Przykładowy klucz główny UUID: 2d8e72dc-aa4e-4c4c-b02b-b08655d891aa

Dodatkowo, zamiast przechowywać hasła w postaci zwykłego tekstu, hasła użytkowników są szyfrowane przy użyciu algorytmu `bcrypt`. Dzięki któremu nawet w przypadku nieautoryzowanego dostępu do bazy danych, hasła pozostają bezpieczne, ponieważ odzyskanie oryginalnej wartości z hasha jest praktycznie niemożliwe. Dodatkowo technika solenia (`and salt`) stosowana w `pgcrypto` (rozszerzenie do PostgreSQL) dodaje losowy ciąg znaków do każdego hasła przed jego zhashowaniem, co zapobiega atakom typu "rainbow table" i sprawia, że dwa identyczne hasła będą miały różne zaszyfrowane wartości.

Przykładowe zaszyfrowane hasło:

`$2a$06$M/okyRSZvHY80DUo6r1Ti.SNaZCZF5/NisSB5PMZCPeaPwnFZplAy.`

Po odszyfrowaniu okazuje się, że ukryty ciąg znaków to: `twojehaslo`

Do przechowywania informacji o współrzędnych geograficznych w tabeli `Addresses`, użyto typu danych `geography(point,4326)`. Ten typ danych pozwala na precyzyjne przechowywanie współrzędnych geograficznych zgodnych z globalnym systemem odniesienia WGS 84. Umożliwia to wykonywanie zaawansowanych operacji geograficznych, takich jak obliczanie odległości czy znajdowanie punktów w określonym promieniu.

Przykładowa wartość tego typu danych może wyglądać w następujący sposób:

`0101000020E61000004B598638D68D4940E0F3C308E1093140`

Odpowiednią kwerendą, uzyskać możemy szerokość i wysokość geograficzną

Listing 2.1: Przykład kwerendy SQL

```
SELECT  ST_ X( addresses. geography:: geometry) AS  longitude ,
          ST_ Y( addresses. geography:: geometry) AS  latitude
FROM    addresses;
```

W tym przypadku jest to szerokość: 51.1081 oraz wysokość 17.03859, czyli współrzędne geograficzne Wrocławia.

# Rozdział 3

## Implementacja

W niniejszym rozdziale przedstawiona zostanie szczegółowa analiza aspektów technicznych związanych z implementacją aplikacji. Na początku zaprezentowane zostaną wykorzystane narzędzia wraz z uzasadnieniem ich wyboru w kontekście wymagań projektowych. Następnie omówiona zostanie architektura warstw systemu. W dalszej części rozdziału szczegółowo opisane zostaną poszczególne moduły systemu oraz ich wzajemne powiązania. Następna część poświęcona będzie omówieniu implementacji kluczowych funkcjonalności: mechanizmu publikacji ogłoszeń, modułu wizualizacji tras na mapie oraz algorytmu rekomendacji ogłoszeń. W ostatniej części rozdziału przedstawione zostanie podejście do testowania aplikacji oraz proces jej wdrożenia na środowisko produkcyjne.

### 3.1. Opis narzędzi

Aplikacja została wykonana w technologii webowej, przy użyciu fullstackowego frameworka `Next.js` [3] (w wersji 15.1.3). Jest to popularny sposób tworzenia aplikacji webowych, zapewniający wiele przydatnych funkcji poprawiających optymalizację. Między innymi `Next.js` skraca czas potrzebny do załadowania aplikacji poprzez renderowanie kodu HTML strony, już podczas kompilacji aplikacji. Został on oparty na innym frameworku - `React` [5], jest to jedna z najpopularniejszych bibliotek JavaScript do budowania interfejsów użytkownika. Opiera się ona na koncepcji komponentów - niezależnych, wielokrotnego użytku bloków interfejsu. Komponenty te w `Next.js` mogą być renderowane po stronie serwera, co znacząco przyspiesza pierwsze ładowanie strony. Dodatkowo renderowanie komponentów aplikacji po stronie serwera pozytywnie wpływa na algorytmy SEO, co skutkuje lepszym pozycjonowaniem serwisu w wyszukiwarkach internetowych. `Next.js` posiada zintegrowane rozwiązanie fullstackowe, oznacza to, że zarówno backend jak i frontend aplikacji wykorzystują tę samą bazę kodu. Takie podejście znacząco upraszcza proces developmentu i utrzymania aplikacji, eliminując potrzebę zarządzania oddzielnymi projektami dla części serwerowej i klienckiej. Dodatkowo ułatwia proces wdrożenia aplikacji na środowisko produkcyjne.

Zamiast standardowych klas CSS, do interfejsu użytkownika została użyta biblioteka `Tailwind` [6] (w wersji 3.4.1). Jest to narzędzie typu `utility-first` CSS, które pozwala na szybkie tworzenie responsywnych interfejsów poprzez zastosowanie predefiniowanych klas bezpośrednio w kodzie HTML.

Do wyświetlania na mapie tras z ogłoszeń użyte zostało API `Leaflet`. Jest to darmowy serwis oferujący interaktywne mapy, który umożliwia łatwą implementację funkcjonalności takich jak: wyświetlanie markerów, rysowanie tras czy obsługa zdarzeń związanych z interakcją użytkownika z mapą. Biblioteka została zoptymalizowana pod kątem urządzeń mobilnych, zapewniając płynne działanie i responsywność na różnych rozmiarach ekranów.

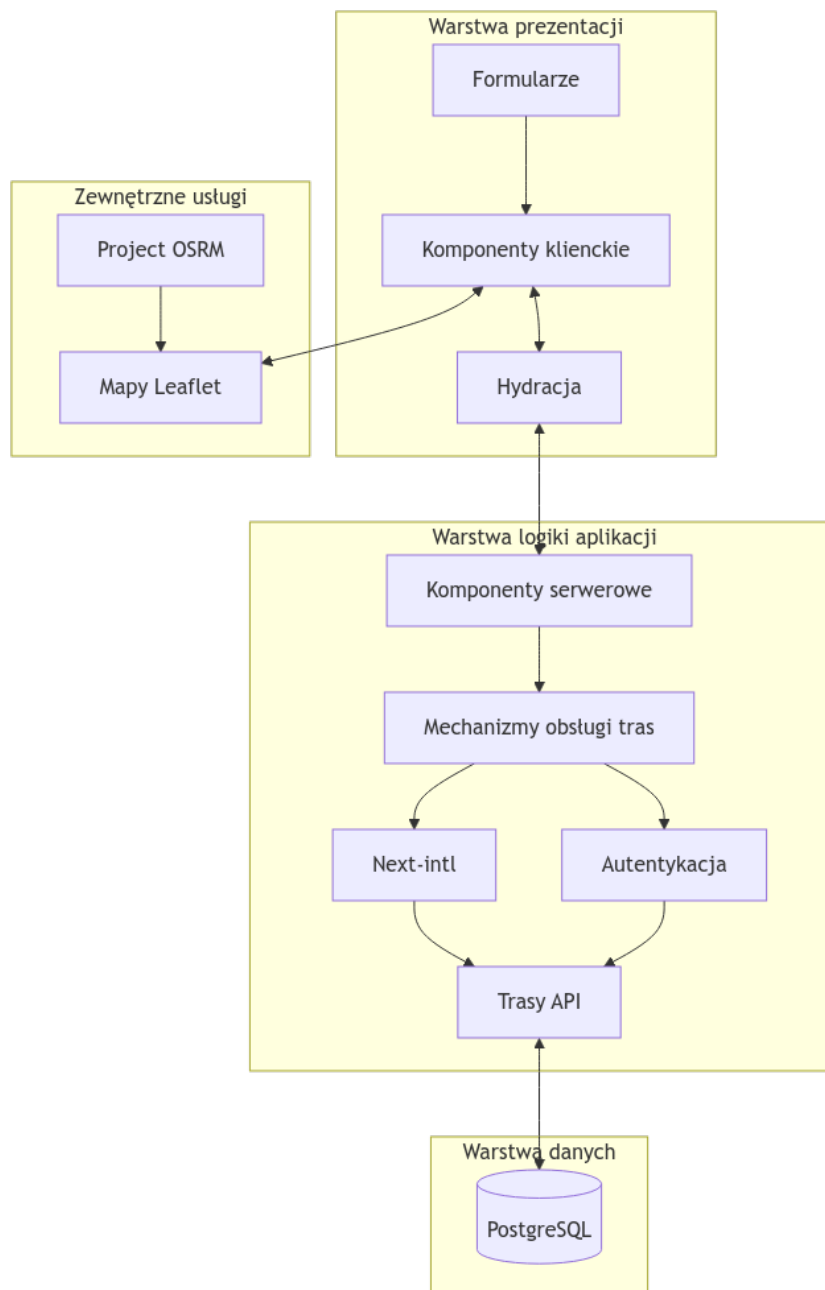


Testy jednostkowe aplikacji wykonane zostały przy użyciu biblioteki Jest [2] (w wersji 29.7.0). Framework ten został wybrany ze względu na jego natywną integrację z Next.js oraz możliwość łatwego tworzenia i uruchamiania testów dla komponentów React.

W celu usprawnienia procesu wdrożenia aplikacji na środowisko produkcyjne, zastosowano narzędzie Docker [1]. Technologia ta pozwala na izolację poszczególnych modułów aplikacji w dedykowanych kontenerach, zapewniając jednakowe środowisko uruchomieniowe niezależnie od platformy. Kontenery Docker stanowią wyizolowane, lekkie maszyny wirtualne, w których uruchamiana jest aplikacja. Zastosowanie takiej architektury umożliwia zdefiniowanie wszystkich wymaganych zależności na etapie budowania obrazu kontenera, eliminując konieczność ich konfiguracji w środowisku produkcyjnym.

## 3.2. Opis architektury

### 3.2.1. Architektura warstw



Rys. 3.1: Diagram architektury warstw aplikacji

Powyższy diagram przedstawia architekturę warstw aplikacji. Zostały na nim wyszczególnione następujące warstwy:

1. **Warstwa prezentacji** - odpowiada za bezpośrednią interakcję z użytkownikiem. W jej skład wchodzi:
  1. Formularze - stanowią pośrednika między użytkownikiem, a warstwą logiki aplikacji. Pozwalają na wysyłanie wprowadzonych danych do warstwy logiki aplikacji,
  2. Komponenty klienckie (ang. *client components*) - interaktywne elementy interfejsu użytkownika renderowane po stronie przeglądarki końcowego użytkownika, które:

- obsługują zdarzenia użytkownika (kliknięcia, wprowadzanie danych),
  - zarządzają stanem lokalnym aplikacji przy użyciu haków React (ang. *React hooks*),
  - implementują dynamiczne zachowania interfejsu bez przeładowywania strony,
  - komunikują się z API poprzez żądania HTTP,
  - renderują interaktywne elementy mapy przy użyciu biblioteki Leaflet.
3. Hydracja - kluczowy proces w architekturze aplikacji *Next.js*. Przekształca początkowy, statyczny widok strony w pełni interaktywną aplikację React. Poprzez wypełnianie komponentów klienckich danymi pozyskanymi z warstwy logicznej aplikacji.
2. **Warstwa logiki aplikacji** - są w niej zawarte wszelkie mechanizmy, do których użytkownik końcowy nie może mieć wglądu z uwagi na bezpieczeństwo. Warstwa ta składa się z:
1. Komponentów serwerowych (ang. *server components*) - komponenty interfejsu użytkownika, które:
    - są renderowane po stronie serwera, a użytkownikowi zwracany jest gotowy kod HTML,
    - redukują ilość JavaScript przesyłanego do przeglądarki,
    - mogą bezpośrednio komunikować się z bazą danych,
    - mają dostęp do wrażliwych danych i zmiennych środowiskowych,
    - są odpowiednie do renderowania statycznej zawartości i elementów niewymagających interaktywności.
  2. Mechanizmów obsługi tras (ang. *route handlers*) - odpowiadają za przetwarzanie żądań przychodzących i wysyłanie odpowiedzi.
    - obsługują metody HTTP (*GET, POST, PUT, DELETE*),
    - zapewniają bezpieczną komunikację między frontendem a backendem aplikacji.
  3. *Next-intl* - biblioteka do internationalizacji, jest ona wykorzystywana do prezentacji treści w różnych językach (polskim, angielskim oraz niemieckim).
  4. Autentykacja - zapewnia bezpieczeństwo poprzez blokowanie wybranych tras API nieautoryzowanym użytkownikom.
  5. Trasy API - specjalne endpointy w aplikacji, są wykorzystywane do:
    - zarządzania ogłoszeniami (dodawanie, zatwierdzanie, usuwanie zleceń),
    - obsługi procesu rejestracji i logowania użytkowników,
    - pobierania i filtrowania listy dostępnych ogłoszeń,
    - komunikacji z zewnętrznym API OSRM dla wyznaczania tras,
3. **Warstwa danych** - Warstwa odpowiadająca za przechowywanie oraz dostarczanie wszelkich danych aplikacji. W aplikacji wykorzystano relacyjną bazę danych PostgreSQL, która przechowuje informacje o:
- użytkownikach systemu i ich rolach,
  - zleceniach transportowych i ich statusach,
  - lokalizacjach geograficznych (punkty początkowe i końcowe tras),
  - czatach między użytkownikami oraz umowami zawartymi między nimi,
4. **Zewnętrzne usługi** - systemy i API wykorzystywane do rozszerzenia funkcjonalności aplikacji:
1. Project OSRM (ang. *Open Source Routing Machine*), używany jest do zapewnienia funkcji wyznaczania tras między punktami. Oferuje API do integracji z aplikacjami transportowymi.

2. Mapy Leaflet, jest to biblioteka służąca do interaktywnej wizualizacji map, która umożliwia wyświetlanie markerów lokalizacji. Pozwala również na rysowanie tras na mapie oraz zapewnia intuicyjną nawigację i przybliżanie mapy.

### 3.3. Opis implementacji

#### 3.3.1. Implementacja mechanizmu publikacji ogłoszeń o planowanej trasie

Dodawanie nowego ogłoszenia o planowanej trasie odbywa się poprzez wysłanie przez użytkownika wypełnionego formularza. W bazie danych muszą zostać zawarte informacje o: adresie początkowym, adresie końcowym, datach odjazdu i przyjazdu, tytule ogłoszenia oraz jego opisie, a także o ładowności pojazdu (maksymalny ciężar towaru, maksymalne wymiary towaru oraz maksymalna wysokość towaru). Walidacja wprowadzonych przez użytkownika danych odbywa się po stronie serwera. Poniżej zamieszczony został fragment funkcji dodającej do bazy danych nowe ogłoszenie o planowanej trasie.

Listing 3.1: Wstępna walidacja danych przed dodaniem ogłoszenia do bazy danych

```
export async function addAnnouncement( state:
  NewAnnouncementFormState, formData: FormData) {
  const t = await getTranslations(' addPost');
  const validatedFields = newAnnouncementSchema( t). safeParse({
    title: formData.get(' title'),
    brand: formData.get(' brand'),
    model: formData.get(' model'),
    maxWeight: formData.get(' maxWeight'),
    maxSize: formData.get(' maxSize'),
    maxHeight: formData.get(' maxHeight'),
    fromCity: formData.get(' fromCity'),
    toCity: formData.get(' toCity'),
    departureDate: formData.get(' departureDate'),
    arrivalDate: formData.get(' arrivalDate'),
    desc: formData.get(' desc'),
    from: JSON. parse( formData.get(' from') as string) as
    Address,
    to: JSON. parse( formData.get(' to') as string) as Address,
  });
  if (! validatedFields. success) {
    return {
      errors: validatedFields. error. flatten(). fieldErrors,
    };
  }
  // Dalszy fragment funkcji
}
```

W powyższym fragmencie funkcji `addAnnouncement` widoczny jest mechanizm dodawania nowego ogłoszenia o planowanej trasie. Funkcja realizuje kilka kluczowych zadań:

1. **Walidacja danych wejściowych** - funkcja wykorzystuje schemat walidacji `newAnnouncementSchema` zdefiniowany przy pomocy biblioteki `Zod`. Do schematu podawana jako argument jest funkcja pobierająca tłumaczenia dla komunikatów błędów.

2. **Obsługa danych** - dane są pobierane z obiektu `FormData`, dostarczanego podczas wysyłania formularza. Adresy początkowy i końcowy są parsowane z obiektu `JSON` zawierającego dokładne informacje o: współrzędnych, identyfikatorów krajów, stanów, miast, kodzie pocztowym oraz innych niezbędnych danych.
3. **Mechanizm obsługi błędów** - w przypadku niepowodzenia walidacji funkcja zwraca do formularza obiekt z wyszczególnionymi błędami dla poszczególnych pól.

Walidacja danych odbywa się po stronie serwera, jest to bezpieczniejszy sposób niż walidacja po stronie klienta, gdyż użytkownik końcowy nie ma wglądu w kod sprawdzający poprawność danych i nie może go obejść.

Listing 3.2: Schemat walidacji danych wejściowych

```
export const newAnnouncementSchema = ( t: any ) =>
  z. object({
    title: z
      . string()
      . min(1, { message: t(' mustNotBeEmpty') })
      . trim(),
    brand: z
      . string()
      . min(1, { message: t(' mustNotBeEmpty') })
      . trim(),
    model: z
      . string()
      . min(1, { message: t(' mustNotBeEmpty') })
      . trim(),
    maxWeight: z. coerce
      . number()
      . refine( inRange(1,50000), { message: t('
valueBetween') + '1-50000' })),
    maxSize: z. string(). regex(/^[1-9][0-9]* x[1-9][0-9]* $/, {
      message: t(' formatIssue'),
    }),
    maxHeight: z. coerce
      . number()
      . refine( inRange(1, 1000), { message: t('
valueBetween') + '1-1000' })),
    desc: z. string(). trim(),
    departureDate: z. coerce. date(),
    arrivalDate: z. coerce. date(),
    from: AddressSchema( t),
    to: AddressSchema( t),
  });
```

Przedstawiony schemat walidacji danych wejściowych `newAnnouncementSchema` definiuje reguły sprawdzania poprawności dla nowego ogłoszenia transportowego przy użyciu biblioteki `Zod`. Schemat obejmuje następujące kluczowe elementy:

1. **Walidacja tekstowa** - pola takie jak tytuł, marka, model wymagają niepustej wartości, która jest dodatkowo przycinana z białych znaków.
2. **Walidacja liczbowa** - maksymalna waga towaru musi zawierać w zakresie 1-50000, a maksymalna wysokość towaru w zakresie 1-1000.

3. **Walidacja formatu** - maksymalny rozmiar towaru wymaga formatu *szerokość* x *wysokość* liczonych w europaletach (np. 3x2). Dаты wyjazdu i przyjazdu są konwertowane do formatu daty.
4. **Walidacja adresów** - wykorzystuje osobny schemat AddressSchema dla weryfikacji danych adresowych.

Schemat używa funkcji tłumaczącej do generowania komunikatów błędów, co zapewnia internacjonalizację komunikatów walidacyjnych.

Jeżeli wprowadzone dane zostaną pozytywnie rozpatrzone przez walidację, następuje dodanie ogłoszenia o planowanej trasie do bazy danych.

Listing 3.3: Implementacja dodawania ogłoszenia do bazy danych

```
export async function addAnnouncement( state:
  NewAnnouncementFormState, formData: FormData) {

  // Powyzej opisana walidacja danych

  const data = validatedFields.data;
  let [ size_x, size_y ] = data.maxSize.split(' x').map(
    Number);
  const { userId } = await verifySession();

  const from_address_id = await addAddress( data.from);
  const to_address_id = await addAddress( data.to);

  await sql(
    ' INSERT INTO announcements ( title, description, start_
    date, arrive_date, max_weight, size_x, size_y, max_height,
    author_id, is_accepted, vehicle_brand, vehicle_model, from_
    address_id, to_address_id, road_color) VALUES( $1, $2, $3, $4,
    $5, $6, $7, $8, $9, $10, $11, $12, $13, $14, $15)',
    [
      data.title,
      data.desc,
      data.departureDate,
      data.arrivalDate,
      data.maxWeight,
      size_x,
      size_y,
      data.maxHeight,
      userId,
      false,
      data.brand,
      data.model,
      from_address_id,
      to_address_id,
      '#' + (( Math.random() * 0xffffffff) << 0).toString(16),
    ],
  );
  redirect({ locale: 'pl', href: '/ announcements' });
}
```

Powyższy fragment funkcji `addAnnouncement` realizuje proces zapisu nowego ogłoszenia transportowego do bazy danych. Wykonwane są w nim następujące czynności:

1. **Przygotowanie danych** - parsuje rozmiar towaru na osobne wartości `size_x` i `size_y`, a także weryfikuje czy użytkownik w rzeczywistości jest zalogowany. Następnie do tabeli `addresses` dodawane są rekordy z adresem początkowym oraz końcowym.
2. **Zapis do bazy danych** - dodaje rekord do tabeli `announcements` używając wszystkich przesłanych przez użytkownika danych. Odbywa się to poprzez używanie zmiennych (`$1`, `$2`, ...), uodparnia to aplikację na atak typu `SQL injection`.
3. **Automatyczne przekierowanie do listy ogłoszeń po pomyślnym dodaniu.**

Domyślnie kolumna `is_accepted` jest ustawiona na `false` ze względu na to, że ogłoszenie nie powinno być było widoczne publicznie natychmiast po dodaniu. Stanie się ono widoczne dopiero kiedy jeden z moderatorów je zatwierdzi. Wartość kolumny `road_color` jest generowana losowo, kolor ten będzie używany do rysowania trasy przewozu na mapie.

### 3.3.2. Implementacja wizualizacji tras na mapie

W aplikacji użyto darmową open-source'ową bibliotekę `Leaflet`, pozwala ona na wygenerowanie interaktywnej mapy, na której wyświetlane będą punkty oraz trasy. Wyświetlenie rzeczywistej trasy między punktami wymaga posiadania bazy danych ze współrzędnymi geograficznym wszystkich dróg na świecie. W aplikacji posłużono się gotowym darmowym rozwiązaniem w formie API. `Project OSRM` (ang. *Open Source Routing Machine*) pozwala na wygenerowanie macierzy współrzędnych geograficznych, wskazujących na najkrótszą drogę między dwoma punktami.

Listing 3.4: Argumenty komponentu mapy

```
export type Road = {
  from?: GeoPoint;
  to?: GeoPoint;
  postId?: string;
  color?: string;
};

type MapProps = {
  zoom?: number;
  className?: string;
  roads?: Road[];
};

export default function Map({ zoom = 13, className,
  roads = [] }: MapProps) {
  // Dalszy fragment komponentu mapy
}
```

Komponent `Map` przyjmuje w argumencie wartość `zoom`, czyli poziom przybliżenia mapy, domyślnie w aplikacji ta wartość została ustawiona na 13, co daje nam widok na całą Europę. Argument `className` to klasy `Tailwind`, pozwala na stylizację komponentu w zależności od potrzeb. Ostatni argument `roads` to tablica typu `Road`, typ ten zawiera informacje o trasie, współrzędnych geograficznych punktu początkowego oraz końcowego, identyfikator powiązanego z nią ogłoszenia oraz kolor rysowanej na mapie trasy.



Listing 3.5: Implementacja pobierania najkrótszych tras między punktami

```

export default function Map({ zoom = 13, className,
  roads = [] }: MapProps) {

  const [ routesPoints, setRoutesPoints] = useState<[ number,
number][][]>([]);
  const [ selectedRoute, setSelectedRoute] = useState< number |
null>( null);
  const t = useTranslations(' map');

  useEffect(() => {
    const fetchRoutes = async () => {
      const newRoutes: [ number, number][][] = [];

      for ( const road of roads) {
        if (! road. from?. coordinates || ! road. to?.
coordinates) continue;

        try {
          const response = await fetch(
            ' https:// router. project- osrm. org/ route/ v1/
driving/ ${ road. from. coordinates[1]}, ${ road. from.
coordinates[0]}; ${ road. to. coordinates[1]}, ${ road. to.
coordinates[0]}? overview= full& geometries= geojson',
          );

          const data = await response. json();

          if ( data. routes?.[0]?. geometry?. coordinates) {
            const points = data. routes[0]. geometry.
coordinates. map(
              ( coord: [ number, number]) => [ coord[1],
coord[0]] as [ number, number],
            );
            newRoutes. push( points);
          }
        } catch ( error) {
          console. error(' Bład_ podczas_ pobierania_ trasy:',
error);
          newRoutes. push([]);
        }
      }

      setRoutesPoints( newRoutes);
    };

    fetchRoutes();
  }, [ roads]);
}

```

Powyższy fragment kodu ukazuje implementację pobierania danych o najkrótszej trasie z API OSRM. Hak Reacta (ang. *React Hook*) o nazwie `useEffect()` powoduje, że kod zawarty w tym haku, wykona się już po załadowaniu komponentu na stronie. Fragment ten spełnia następujące funkcje:

1. **Sprawdzanie poprawności argumentów** - zanim API OSRM zacznie być odpytywane, argument `roads` sprawdzany jest pod kątem posiadania w sobie pola ze współrzędnymi geograficznymi.
2. **Wysyłanie zapytania do API OSRM** - zapytanie używa metody `GET` i są w nim zawarte informacje dotyczące współrzędnych geograficznych punktu początkowego oraz końcowego. Zauważyć można, że współrzędne podane są w odwrotny sposób niż ogólna przyjęta konwencja (najpierw szerokość, później długość), są to wymagania tego API.
3. **Zamiana szerokości geograficznej z długością** - oprócz tego, że wymagane jest podawanie współrzędnych w odwróconej formie, to są one również w takiej formie zwracane. Po uzyskaniu odpowiedzi od API należy ją odwrócić.
4. **Obsługa błędów** - jeżeli odpowiedź nie zostanie uzyskana lub zapytanie zwróci błąd, wówczas do konsoli aplikacji wypisywany jest komunikat o niepowodzeniu.
5. **Ustawienie tras w stanie komponentu** - uzyskane macierze ze współrzędnymi geograficznymi tras, są ustawiane jako aktualny stan komponentu.

Listing 3.6: Implementacja rysowania tras na mapie

```

export default function Map({ zoom = 13, className,
roads = [] }: MapProps) {
// Fragment kodu opisany powyzej
return (
  < MapContainer
    center={ roads[0]?. from?. coordinates ? [ Number( roads[0].
from. coordinates[0]), Number( roads[0]. from.
coordinates[1])] : undefined}
    zoom={ zoom}
    scrollWheelZoom={ true}
    className={ ` ${ className} z-10 h- screen w- full`}
  >

    { routesPoints. map(
      ( points, index) =>
        points &&
        points. length > 0 && (
          < div key={ index}>
            < Polyline
              key={ index}
              positions={ points}
              color={ roads[ index]. color}
              weight={ 3}
              opacity={ 0.5}
              eventHandlers={{
                click: () => {
                  setSelectedRoute( index);
                },
              }}
            />
            { selectedRoute === index && (
              < Popup
                position={ points[ Math. floor( points.
length / 2)] }
                eventHandlers={{
                  remove: () => setSelectedRoute( null),
                }}
              >
                < div>
                  < Link href={ ` / announcements / ${ roads[ index].
postId }` }>
                    Przejdź do ogłoszenia
                  < / Link>
                < / div>
              < / Popup>
            ) }
          < / div>
        ),
      ) }
  < / MapContainer>
);
}

```

Na powyższym fragmencie kodu zauważyć można renderowanie komponentu mapy z wykorzystaniem biblioteki Leaflet. Implementacja łączy pobrane wcześniej dane tras z interaktywną wizualizacją na mapie, umożliwiając użytkownikowi przeglądanie i eksplorację tras transportowych. Powyższy kod obejmuje następujące kluczowe funkcjonalności:

1. **Inicjalizacja kontenera mapy** - ustawienie centrum mapy na podstawie współrzędnych pierwszej trasy, dodanie możliwości przewijania mapy za pomocą rolki myszy.
2. **Renderowanie tras** - mapowanie pobranych wcześniej punktów tras. Rysowanie linii tras (Polyline) z indywidualnym kolorem dla każdej trasy.
3. **Obsługa interakcji** - zaznaczanie wybranej trasy, wyświetlanie wyskakującego okna (ang. *Popup*) po kliknięciu trasy. W oknie znajdują się odnośnik do powiązanego ogłoszenia.

Połączenie powyższych fragmentów kodu pozwala na wyświetlenie interaktywnej mapy z trasami transportowymi, co stanowi kluczową funkcjonalność aplikacji.

### 3.3.3. Implementacja algorytmu rekomendacji ogłoszeń

Rekomendowanie ogłoszenia odbywa się w funkcji `addAnnouncement` 3.3.1 jeszcze przed dodaniem ogłoszenia do bazy danych. Algorytm rekomendacji polega na znalezieniu ogłoszeń, które są najbardziej zbliżone do dodawanego ogłoszenia. W tym celu wykorzystywane są współrzędne geograficzne punktu początkowego i końcowego dodawanego ogłoszenia. Następnie obliczana jest odległość od punktów początkowych i końcowych innych ogłoszeń. Ogłoszenie, które wykaże odległość od punktu początkowego i punktu końcowego nie większą niż 50km oraz zawiera się w odpowiednim zakresie właściwości fizycznych towaru, jest zwracane jako rekomendacja.

Listing 3.7: Implementacja rekomendacji ogłoszeń

```
export async function tryLinkToPost( post: ErrandProps |
AnnouncementProps): Promise< string | null> {
  if ('ware' in post) {
    const announcements = await getAnnouncements( SortDirection.
ByNewest);
    const match = announcements. find(( announcement) =>
      isMatchingDelivery(
        post. from. geography!,
        announcement. from. geography!,
        post. to. geography!,
        announcement. to. geography!,
        post. ware,
        announcement. carProps,
      ),
    );
    return match?. id ?? null;
  } else {
    const errands = await getErrands( SortDirection. ByNewest);
    const match = errands. find(( errand) =>
      isMatchingDelivery(
        post. from. geography!,
        errand. from. geography!,
        post. to. geography!,
        errand. to. geography!,
        errand. ware,
        post. carProps,
      ),
    );
    return match?. id ?? null;
  }
}
```

Powyższy fragment kodu przedstawia implementację algorytmu rekomendacji ogłoszeń. Funkcja `tryLinkToPost` przyjmuje jako argument ogłoszenie, które ma zostać zarekomendowane. W zależności od tego czy ogłoszenie jest ogłoszeniem o planowanej trasie czy zleceniem transportowym, pobierane są odpowiednie ogłoszenia z bazy danych. Następnie dla każdego ogłoszenia sprawdzane jest czy spełnia ono warunki rekomendacji.

Listing 3.8: Funkcja sprawdzająca warunki powiązania ogłoszeń

```
export function isMatchingDelivery (  
  fromPoint1: GeoPoint,  
  fromPoint2: GeoPoint,  
  toPoint1: GeoPoint,  
  toPoint2: GeoPoint,  
  ware: ErrandProps['ware'],  
  carProps: AnnouncementProps['carProps']  
): boolean {  
  const isStartPointValid = arePointsInRange( fromPoint1,  
    fromPoint2, 50);  
  const isEndPointValid = arePointsInRange( toPoint1,  
    toPoint2, 50);  
  const areSizesValid = areDimensionsValid( ware, carProps);  
  
  return isStartPointValid && isEndPointValid && areSizesValid;  
}
```

Funkcja `isMatchingDelivery` sprawdza czy dane ogłoszenie spełnia warunki rekomendacji. W zależności od tego czy punkty początkowe i końcowe ogłoszeń znajdują się w odpowiedniej odległości od siebie oraz czy wymiary towaru mieszczą się w zakresie wymiarów pojazdu, funkcja zwraca wartość `true` lub `false`.

Listing 3.9: Funkcje obliczająca odległość między punktami

```

export function calculateDistance( point1:  GeoPoint,  point2:
GeoPoint):  number {
  const  R = 6371;

  const  lat1 = ( parseFloat( point1. coordinates[0]) *  Math.
PI) / 180;
  const  lon1 = ( parseFloat( point1. coordinates[1]) *  Math.
PI) / 180;
  const  lat2 = ( parseFloat( point2. coordinates[0]) *  Math.
PI) / 180;
  const  lon2 = ( parseFloat( point2. coordinates[1]) *  Math.
PI) / 180;

  const  dLat =  lat2 -  lat1;
  const  dLon =  lon2 -  lon1;

  const  a =
    Math. sin( dLat / 2) *  Math. sin( dLat / 2) +
    Math. cos( lat1) *  Math. cos( lat2) *  Math. sin( dLon / 2) *
    Math. sin( dLon / 2);

  const  c = 2 *  Math. atan2( Math. sqrt( a),  Math. sqrt(1 -
a));
  const  distance =  R *  c;

  return  Number( distance. toFixed(2));
}

export function arePointsInRange( point1:  GeoPoint,  point2:
GeoPoint,  maxDistance:  number):  boolean {
  return  calculateDistance( point1,  point2) <=  maxDistance;
}

```

W powyższym fragmencie kodu znajdują się funkcje pomocnicze wykorzystywane w algorytmie rekomendacji ogłoszeń. Funkcja `calculateDistance` oblicza odległość między dwoma punktami geograficznymi. Funkcja `arePointsInRange` sprawdza czy odległość między dwoma punktami nie przekracza maksymalnej odległości. Funkcja `areDimensionsValid` sprawdza czy wymiary towaru mieszczą się w wymiarach pojazdu. Do obliczenia odległości między punktami wykorzystany został wzór Haversine’a [7].

$$dystans = 2R \arcsin \left( \sqrt{\sin^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right) \quad (3.1)$$

Gdzie:  $R$  - promień Ziemi  $\phi_1, \phi_2$  - szerokości geograficzne  $\lambda_1, \lambda_2$  - długości geograficzne

Wzór ten jest matematyczną metodą obliczania odległości między dwoma punktami na sferze (w tym przypadku na kuli ziemskiej) przy użyciu ich współrzędnych geograficznych (szerokości i długości). Jest on powszechnie używany w nawigacji GPS oraz w systemach geolokacyjnych. Wzór ten wymaga podania promienia sfery (dla Ziemi jest to 6371km), następnie przekształca stopnie na radiany oraz używa funkcji trygonometrycznych do obliczenia różnicy



współrzędnych. Końcowy wynik jest zaokrąglany do dwóch miejsc po przecinku, co daje odległość liczoną w kilometrach.

Listing 3.10: Funkcja sprawdzająca czy ogłoszenie zawiera się w odpowiednim zakresie właściwości fizycznych towaru

```
export function areDimensionsValid( ware: ErrandProps['ware'],
  carProps: AnnouncementProps['carProps']): boolean {
  return (
    ware.weight <= carProps.maxWeight &&
    ware.size.height <= carProps.maxSize.height &&
    ware.size.x * ware.size.y <= carProps.maxSize.x *
    carProps.maxSize.y
  );
}
```

Funkcja `areDimensionsValid` sprawdza czy wymiary towaru mieszczą się w wymiarach pojazdu. Wartości te są porównywane z wartościami maksymalnymi pojazdu, jeżeli wszystkie wartości są mniejsze lub równe wartościom maksymalnym, to funkcja zwraca wartość `true`, w przeciwnym wypadku `false`.

## 3.4. Opis testów aplikacji

Aplikacja w trakcie implementacji była testowana na bieżąco. Testy obejmowały zarówno testy jednostkowe i testy end-to-end. Testy jednostkowe sprawdzały poprawność działania poszczególnych komponentów, a testy end-to-end sprawdzały poprawność działania aplikacji jako całości. W pełni wykonaną aplikację przetestowano również pod kątem wydajności.

### 3.4.1. Testy jednostkowe

Testy jednostkowe oraz integracyjne napisano w języku JavaScript przy użyciu biblioteki Jest. Sprawdzają one poprawność działania poszczególnych komponentów aplikacji. W aplikacji testowano następujące komponenty:

- Pasek nawigacyjny, w kodzie aplikacji nazwany `Nav`,
- Formularz rejestracji, w kodzie aplikacji nazwany `SignupForm`,
- Wyświetlanie zlecenia, w kodzie aplikacji nazwany `Errand`,
- Mapę z narysowanymi trasami, w kodzie aplikacji nazwany `AllRoadsMap`,

Przykładowy test jednostkowy sprawdzający poprawność działania komponentu `Nav`. Komponent ten został użyty między innymi na stronie przeglądarki ogłoszeń o planowanych trasach 2.3.1. W aplikacji przyjmuje on dwie formy, zależnie od szerokości przeglądarki. Gdy szerokość ekranu jest większa niż 768px, pasek nawigacji jest stale widoczny po lewej stronie. Jeżeli ekran jest węższy niż ta wartość graniczna, pasek nawigacji domyślnie jest ukryty, a pokazuje się dopiero po kliknięciu w ikonę hamburgera.

Listing 3.11: Przykładowy test jednostkowy paska nawigacyjnego

```

it(' opens mobile menu when hamburger icon is clicked', () => {
  render(
    < NextIntlClientProvider messages={ messages} locale=" pl">
      < Nav />
    </ NextIntlClientProvider>,
  );

  const menuButton = screen.getByTestId(' menu- button');
  fireEvent.click( menuButton);

  const sideMenu = screen.getByTestId(' side- menu');
  const menuOverlay = screen.getByTestId(' menu- overlay');

  expect( sideMenu). not. toHaveClass(' hidden');
  expect( sideMenu). toHaveClass(' fixed');
  expect( menuOverlay). toBeInTheDocument();
});

```

Powyższy test pokazuje sprawdzanie funkcjonalności otwierania menu mobilnego po kliknięciu w ikonę hamburgera. Sprawdza on czy, menu mobilne zostaje poprawnie otwarte, czy posiada odpowiednie klasy oraz czy nagłówek menu został wyrenderowany.

Inne testy jednostkowe dla tego komponentu sprawdzały poprawność następujących funkcjonalności:

1. Renderowanie nagłówka z nazwą aplikacji.
2. Czy mobilne menu jest domyślnie zamknięte.
3. Czy menu mobilne jest poprawnie zamykane po kliknięciu ikony hamburgera.
4. Czy nazwa obecnej strony jest poprawnie wyświetlana.
5. Czy chowanie i pokazywanie menu nawigacji będzie poprawnie działać po kliknięciu dwukrotnie w ikonę hamburgera.

```

● > npm run test

> cargolink@0.1.0 test
> jest

PASS  __tests__/Errand.test.tsx
PASS  __tests__/Nav.test.jsx
PASS  __tests__/Signup.test.jsx
PASS  __tests__/AllRoadsMap.test.tsx

Test Suites: 4 passed, 4 total
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        1.343 s
Ran all test suites.

```

Rys. 3.2: Wyniki uruchomienia testów jednostkowych

Wszystkie napisane testy jednostkowe pomyślnie się wykonują.

### 3.4.2. Testy end-to-end

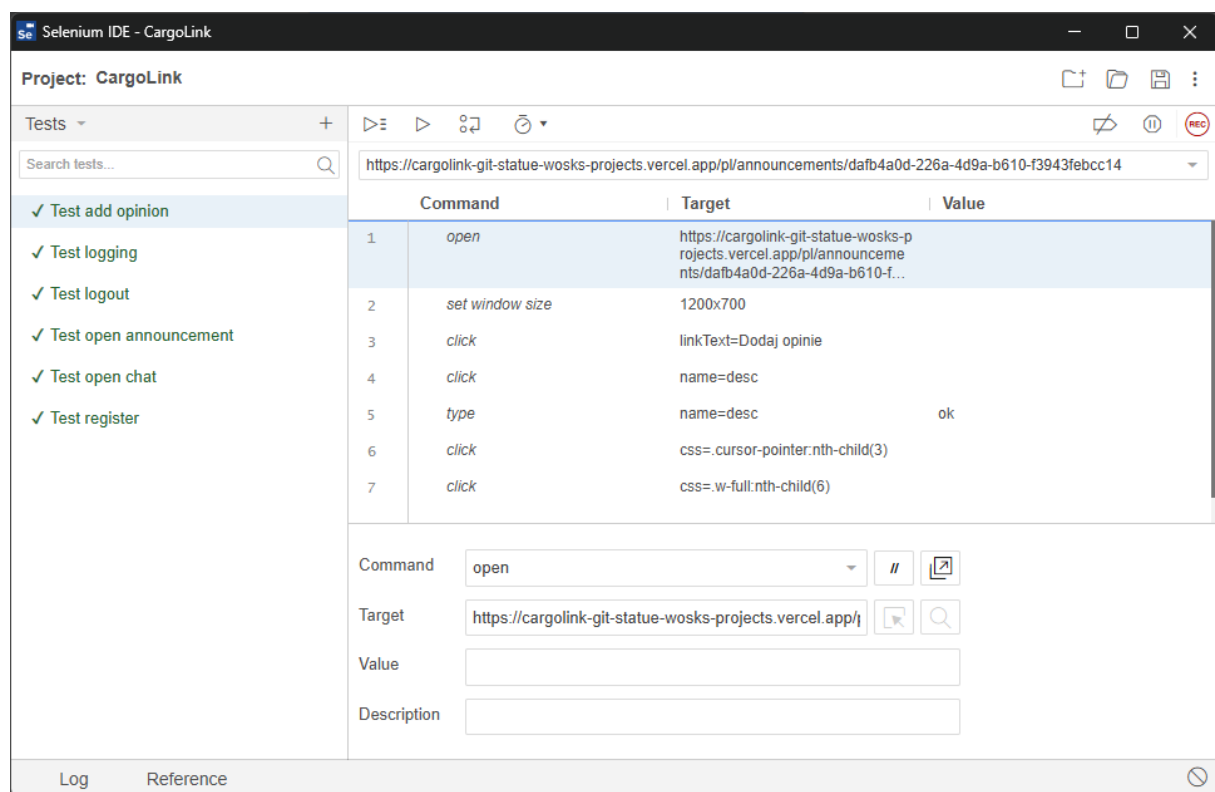
Testy end-to-end wykonano przy użyciu narzędzia Selenium IDE, rozszerzenia do przeglądarki Google Chrome. Narzędzie pozwala na nagrywanie sekwencji interakcji użytkownika z aplikacją, które następnie służą jako scenariusze testowe. Testy przeprowadzono w środowisku maksymalnie zbliżonym do produkcyjnego. Firma Vercel, twórca frameworka Next.js, udostępnia darmowy serwer do wdrażania aplikacji. Mechanizm ciągłego wdrażania (ang. *continuous deployment*) automatycznie buduje aplikację po każdym wysłaniu kodu do repozytorium:

Kod wysłany do gałęzi głównej (*master branch*) uruchamia budowanie wersji produkcyjnej. Kod wysłany do innych gałęzi tworzy środowisko podglądu (*preview*), dostępne tylko dla uprawnionych osób.

Testy Selenium IDE wykonano w środowisku podglądu, które niemal całkowicie odwzorowuje warunki środowiska produkcyjnego.

Zostały przygotowane następujące scenariusze testowe:

- rejestracja nowego użytkownika,
- logowanie do aplikacji,
- wylogowanie z aplikacji,
- otwieranie ogłoszenia i przechodzenie do profilu,
- dodawanie opinii o użytkowniku,
- otwieranie czatu z użytkownikiem



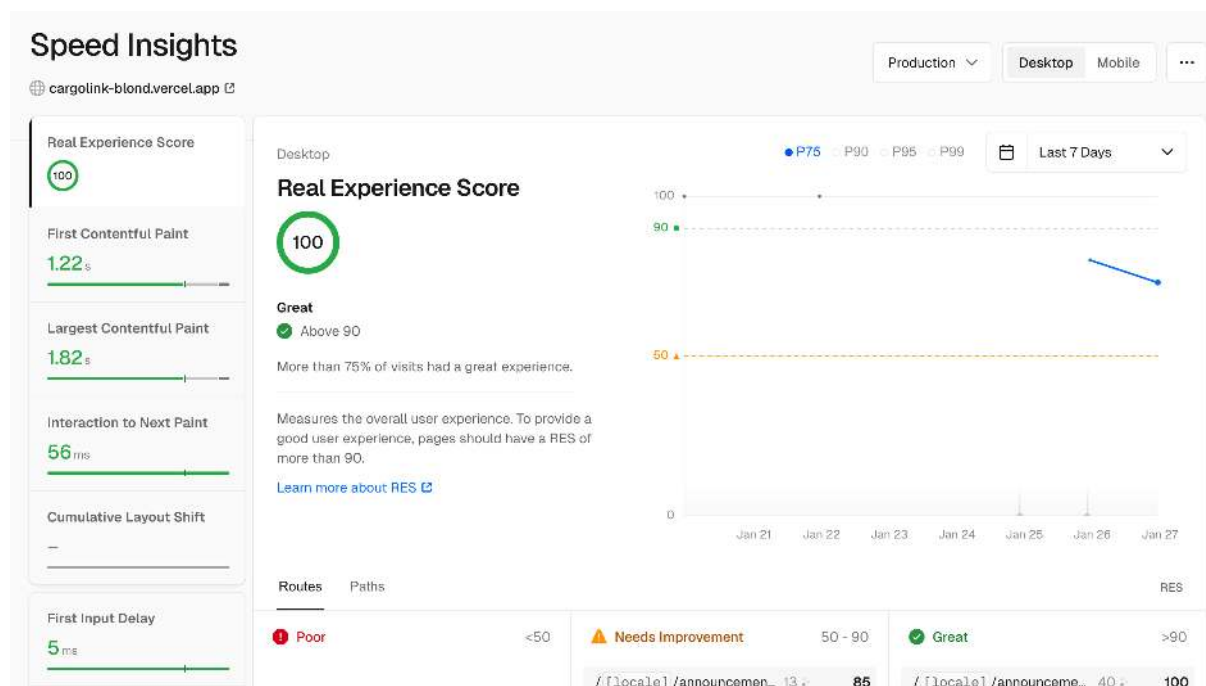
Rys. 3.3: Wyniki uruchomienia testów end-to-end

Początkowo test otwierający czat z użytkownikiem (*Test open chat*) zakończył się niepowodzeniem. W trakcie analizy okazało się, że przyczyną błędu była możliwość utworzenia czatu z ogłoszeniem o planowanej trasie będąc zalogowanym jako przewoźnik. Przewoźnik nie powinien mieć możliwości tworzenia czatu z ogłoszeniem o planowanej trasie, ponieważ również świadczy on usługi przewozowe. Po zmianie polegającej na tym aby, przycisk otworzenia czatu był dostępny tylko dla ogłoszeń o zleceniach transportowych, test zakończył się sukcesem.

Końcowo wszystkie testy end-to-end zakończyły się sukcesem. Testy potwierdziły poprawność działania aplikacji oraz jej zgodność z założeniami funkcjonalnymi.

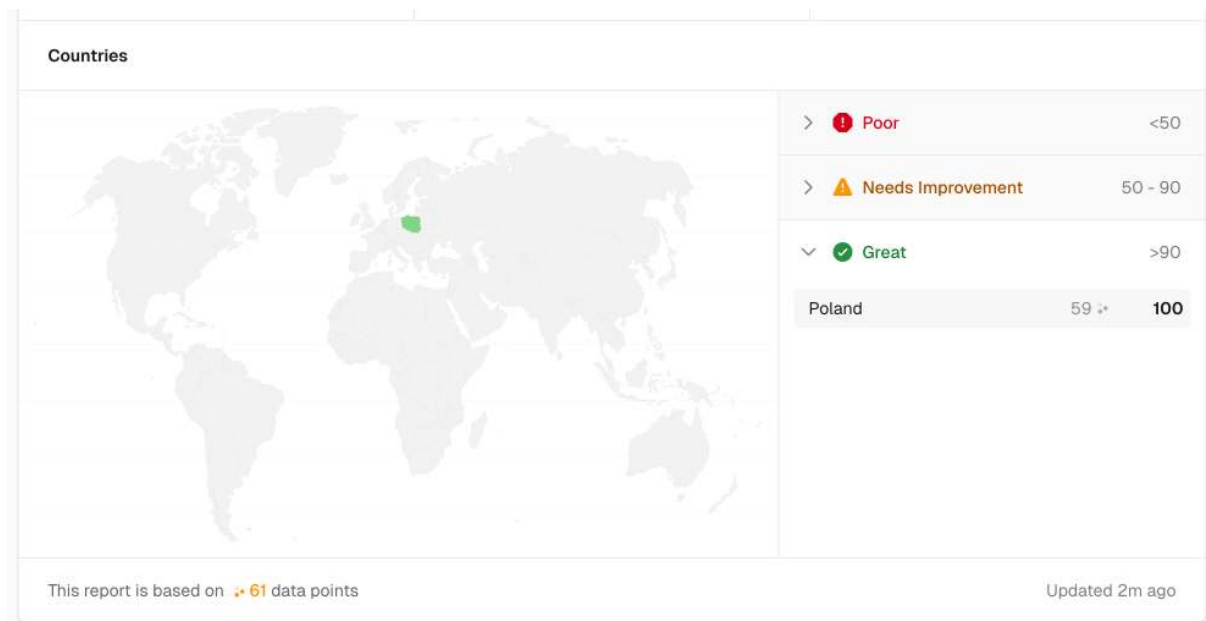
### 3.4.3. Testy wydajnościowe

Wydajność aplikacji mierzono za pomocą biblioteki `@vercel/speed-insights`, które jest zintegrowane z platformą Vercel. Narzędzie to automatycznie zbiera metryki wydajnościowe podczas rzeczywistego użytkowania aplikacji, w tym czas ładowania strony (LCP), opóźnienie interaktywności (FID) oraz przesunięcie układu wizualnego (CLS). Dane te są agregowane i prezentowane w panelu administracyjnym Vercel, co umożliwia monitorowanie wydajności w czasie rzeczywistym oraz identyfikację potencjalnych problemów wydajnościowych. Wszystkie metryki pozostawały w granicach zalecanych przez Web Vitals. Poniżej zamieszczony został zrzut ekranu panelu administracyjnego Vercel ukazujący uzyskane wyniki.



Rys. 3.4: Wyniki testów wydajnościowych

Zebrane dane wydajnościowe pochodzą z 61 rzeczywistych odczytów zarejestrowanych od użytkowników z Polski. Dostarczyły one reprezentatywnej próbki dla analizy zachowania aplikacji w warunkach lokalnych, potwierdzając jej stabilną wydajność.



Rys. 3.5: Pochodzenie danych do testów wydajnościowych

# Rozdział 4

## Podsumowanie i wnioski

### 4.1. Napotkane problemy

Początkowo projekt zakładał przechowywanie nazw miast początkowych i końcowych wraz ze współrzędnymi geograficznymi bezpośrednio w tabelach `announcements` i `errands`. Podczas implementacji odkryto jednak lukę: brakowało informacji o państwie i stanie lokalizacji.

Rozważano ręczne wprowadzanie tych danych przez użytkownika, ale mogłoby to negatywnie wpłynąć na wygodę korzystania z aplikacji. Alternatywą było utworzenie osobnej tabeli `addresses` przechowującej pełne informacje o adresach. Tym samym w tabelach `announcements` i `errands` pozostawiono jedynie klucze obce do tabeli `addresses`.

Zmiana wymusiła modyfikację formularzy dodawania ogłoszeń i zleceń - zamiast ręcznego wprowadzania, użytkownicy zyskali możliwość wyboru adresu z listy.

### 4.2. Możliwości rozwoju

Aplikacja została stworzona z myślą o rozwoju. W przyszłości można by dodać nowe funkcjonalności, takie jak:

- Możliwość dodawania zdjęć do ogłoszeń i zleceń.
- Dodanie możliwości ingerencji w spór między użytkownikami.
- Rozbudowanie systemu oceniania użytkowników, możliwość oceniania dopiero po zakończeniu zlecenia.
- Zgłaszanie nieprawidłowości w ogłoszeniach i zleceniach.
- Dodanie możliwości wyboru dokładnej trasy przejazdu (jeżeli użytkownik nie chce korzystać z proponowanej trasy).

### 4.3. Wnioski

Projekt CargoLink stanowi innowacyjne rozwiązanie w obszarze transportu okazjonalnego, które przy wykorzystaniu nowoczesnych technologii (TypeScript, Next.js, Tailwind CSS, Node.js, PostgreSQL) oferuje kompleksową platformę łączącą przewoźników ze zleceniodawcami. Aplikacja przyczyni się do optymalizacji procesów logistycznych poprzez inteligentny system rekomendacji, bezpośrednią komunikację między użytkownikami, graficzną prezentację tras oraz wielojęzyczny interfejs, co pozwoli na redukcję kosztów transportu i eliminację nieefektywnego wykorzystania zasobów transportowych. Realizacja projektu stanowi odpowiedź na rosnące zapotrzebowanie rynku na narzędzia usprawniające współpracę w sektorze transportowym.

# Literatura

- [1] Dokumentacja Docker. <https://docs.docker.com/>. [dostęp: 19 stycznia 2025].
- [2] Dokumentacja Jest. <https://jestjs.io/docs/getting-started>. [dostęp: 18 stycznia 2025].
- [3] Dokumentacja Next.js. <https://nextjs.org/docs>. [dostęp: 20 października 2024].
- [4] Dokumentacja PostgreSQL. <https://www.postgresql.org/docs/current/index.html>. [dostęp: 20 października 2024].
- [5] Dokumentacja React.js. <https://react.dev/reference/react>. [dostęp: 20 października 2024].
- [6] Dokumentacja Tailwind. <https://tailwindcss.com/docs/installation>. [dostęp: 18 stycznia 2025].
- [7] G. V. Brummelen. *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2013.



# Dodatek A

## Instrukcja wdrożeniowa

Aplikacja jest konteneryzowana przy użyciu Docker, co zapewnia kompatybilność międzyplatformową. Baza danych jest hostowana na darmowym serwerze Neon, co upraszcza konfigurację. Wymagania:

- Zainstalowane oprogramowanie Docker
- Dostęp do internetu
- Uprawnienia administratora

W celu wdrożenia aplikacji na środowisko produkcyjne należy wykonać następujące kroki:

1. Sklonować repozytorium z GitHub.
2. Otworzyć terminal w katalogu projektu.
3. Zbudować obraz aplikacji za pomocą komendy `docker build -t cargolink ..`
4. Uruchomić kontener z aplikacją za pomocą komendy `docker run -p 80:3000 cargolink`.

Po wykonaniu powyższych kroków aplikacja będzie dostępna pod adresem `http://localhost:80`. Port na którym działa aplikacja można zmienić w modyfikując argument w `-p` w komendzie `docker run -p 80:3000 cargolink`. Port 80 jest domyślnym portem dla protokołu HTTP, więc zaleca się pozostawienie go bez zmian. Jeżeli serwer, na którym wykonane zostały powyższe kroki, jest dostępny z zewnątrz, aplikacja będzie dostępna pod adresem `http://<adres_serwera>:80`.