POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Wiktor Czech - s21291
Stanisław Godwod – s21227
Borys Kotnowski – s20610
Aleksander Kozakowski – s19068
Aleksandra Mira – s20383
Rafał Osica – s21290
Bartosz Semeniuk – s20462
Adam Szerszenowicz – s20967
Julia Urbaniak – s20726
Jan Zwolan – s20963

Gary - Kompleksowy System Obsługi Incydentów Ratunkowych

Praca inżynierska napisana pod kierunkiem:
dr. inż. Mariusza Trzaski

Warszawa, luty 2023

Streszczenie

Celem niniejszej pracy inżynierskiej było stworzenie systemu wspierającego udzielanie pierwszej pomocy oraz zarządzanie pracą karetek. Niniejsza aplikacja ma za zadanie zwiększyć skuteczność udzielanej pierwszej pomocy oraz podnieść szanse na przeżycie poszkodowanego.

System Gary składa się z serwera napisanego w języku Java wspartego frameworkiem Spring, aplikacji mobilnej napisanej w języku Kotlin na platformę Android oraz aplikacji webowej napisanej przy pomocy TypeScript z użyciem biblioteki React.

Słowa kluczowe

Pierwsza pomoc, karetka, dyspozytor, resuscytacja, android, java, kotlin, intellij, spring, hibernate, react, typescript, sql

Spis treści

1. Wstęp

W samej Polsce wypadkom ulega kilkanaście tysięcy ofiar każdego dnia, w wyniku których tracą zdrowie lub życie. Szybkie przybycie służb ratunkowych oraz prawidłowe udzielenie pierwszej pomocy zwiększa, a często umożliwia przeżycie ofiar wypadku. Dlatego wiedza, jak udzielić pierwszej pomocy i możliwość szybkiego wezwania służb ratunkowych, jest istotna niezależnie od wieku, czy lokalizacji.

Zagadnienie to jest niezwykle złożone, gdyż rozwiązanie musi wspierać co najmniej trzy grupy docelowe: ratowników medycznych, dyspozytorów i ludzi będących świadkami wypadku. W niniejszej pracy postanowiliśmy rozszerzyć funkcjonowanie systemu o rolę kierownika, którego zadaniem jest dbanie o wyposażenie karetek, czy układanie grafiku służb ratunkowych. Całość pozwala stworzyć kompleksowy system, który zwiększa szanse na przeżycie. Dalsze rozdziały przedstawiają cel, sposób organizacji oraz zakres pracy w toku trwania projektu.

1.1. Cel pracy

Celem niniejszej pracy inżynierskiej było stworzenie kompleksowego systemu do obsługi incydentów ratunkowych, składającego się z aplikacji webowej oraz mobilnej. System ma pomagać użytkownikom wezwać służby ratunkowe oraz udzielić prawidłowo pierwszej pomocy, a także informować dyspozytorów o nowych incydentach i wspierać ich w pracy. Projekt jest realizowany w ramach specjalizacji Inżynieria Oprogramowania i Baz Danych w Polsko – Japońskiej Akademii technik komputerowych.

1.2. Organizacja pracy

Niniejsza praca stanowi dokumentację procesu tworzenia systemu, ale również działalności zespołu. W rozdziale drugim można zapoznać się z kontekstem prawnym oraz specyfiką działania przy incydentach ratunkowych. Co wpływa na funkcjonowanie naszej aplikacji. W rozdziale trzecim przedstawiono istniejące na rynku systemy, zarówno od strony użytkownika, jak i pracownika medycznego.

Kolejne rozdziały pokazują proces tworzenia aplikacji. Poczynając od rozdziału czwartego, gdzie udokumentowano fazę analityczną wraz z wymaganiami funkcjonalnymi, diagramami klas, przypadków użycia oraz scenariuszami wybranych przypadków. Rozdział piąty poświęcony jest technologiom wykorzystywanym w niniejszej pracy. W rozdziale szóstym oraz siódmym pokazane zostały szczegóły implementacji wraz z interfejsem użytkownika. Przedostatni rozdział opisuje testy, a po nim następuje zwieńczenie całej pracy w podsumowaniu. Oprócz tego zostały zawarte dodatki w postaci słownika pojęć oraz podziału pracy ze względu na osoby uczestniczące w niej.

1.3. Zakres pracy

Praca była tworzona w dziesięcioosobowym zespole, który składał się z czterech podzespołów:

1. Analizy
2. Implementacji
3. Testowania
4. Dokumentacji

Każdy członek zespołu uczestniczył intensywnie w pracy przynajmniej jednego podzespołu.

Planowanie odbywało się podczas cotygodniowych spotkań, gdzie przydzielane były zadania na kolejny tydzień oraz każdy zdawał raport z postępów. Głównym kanałem komunikacyjnym był serwer na Discordzie.

Tabela 1 pierwsza przedstawia składy zespołów wraz z podziałem zadań. W dodatku A można zobaczyć podział pracy ze względu na osoby. Każdy zespół składał się z kierownika zespołu oraz jego członków. Dodatkowo w zespole implementacji mieliśmy liderów każdego z modułów systemu, czyli

aplikacji webowej, mobilnej oraz *backend*. Liderzy odpowiadali za jakość oraz poprawność kodu, a także odpowiadali na pytania związane z technologiami i przeprowadzali z nich szkolenia.

Tabela 1. Skład podzespołów i przydział zadań w projekcie Gary. Źródło: Opracowanie własne

Kierownik projektu Gary
Aleksandra Mira

Zespół analityczny	
Kierownik	Jan Zwolan
Członkowie Zespołu	Wiktor Czech Stanisław Godwod Aleksander Kozakowski Aleksandra Mira Rafał Osica Julia Urbaniak
Podział zadań	
Stan Sztuki	Jan Zwolan Julia Urbaniak
Funkcjonalności systemu	Wiktor Czech Stanisław Godwod Aleksander Kozakowski Aleksandra Mira Rafał Osica Julia Urbaniak
Diagram przypadków użycia	Wiktor Czech Stanisław Godwod Rafał Osica Aleksandra Mira Julia Urbaniak
Diagram klas	Wiktor Czech Stanisław Godwod Aleksandra Mira Rafał Osica Julia Urbaniak
Słownik	Julia Urbaniak Jan Zwolan
Diagram aktywności	Wiktor Czech Stanisław Godwod Aleksandra Mira Rafał Osica Julia Urbaniak
Scenariusze	Jan Zwolan
Tutoriale pierwszej pomocy	Jan Zwolan Julia Urbaniak

Zespół implementacji	
Kierownik	Bartosz Semeniuk (lider backend)

Członkowie Zespołu	Wiktor Czech Rafał Osica Aleksander Kozakowski Stanisław Godwod Borys Kotnowski (lider aplikacji mobilnej) Aleksandra Mira Adam Szerszenowicz (lider aplikacji webowej)
Podział Zadań	
Konfiguracja serwera uczelnianego oraz wdrożenie projektu	Bartosz Semeniuk
Lokalizacja	Wiktor Czech Rafał Osica
Automatyzacja budowania serwera	Bartosz Semeniuk
Implementacja klas z diagramu projektowego wraz z odpowiednimi końcówkami REST	Bartosz Semeniuk Aleksandra Mira Aleksander Kozakowski
Implementacja aplikacji webowej	Adam Szerszenowicz Rafał Osica
Implementacja aplikacji mobilnej	Borys Kotnowski Wiktor Czech
Testy jednostkowe	Bartosz Semeniuk Stanisław Godwod Aleksander Kozakowski Aleksandra Mira

Zespół testowania	
Kierownik	Stanisław Godwod
Członkowie Zespołu	Julia Urbaniak
Podział zadań	
Testy manualne	Julia Urbaniak Stanisław Godwod

Zespół dokumentacji	
Kierownik	Julia Urbaniak
Członkowie Zespołu	Jan Zwolan Aleksandra Mira Stanisław Godwod Aleksander Kozakowski Bartosz Semeniuk Wiktor Czech Rafał Osica
Podział zadań	
Wstęp	Aleksandra Mira
Rozdział 2	Aleksandra Mira
Rozdział 3	Julia Urbaniak Wiktor Czech Aleksander Kozakowski
Rozdział 4	Jan Zwolan

Rozdział 5	Julia Urbaniak Rafał Osica Wiktor Czech Bartosz Semeniuk
Rozdział 6	Julia Urbaniak Aleksandra Mira Rafał Osica Wiktor Czech
Rozdział 7	Jan Zwolan
Rozdział 8	Stanisław Godwod Julia Urbaniak
Podsumowanie	Aleksandra Mira

2. Kontekst prawny i codzienne funkcjonowanie

Poniższe rozdziały opisują obecne przepisy dotyczące udzielania pierwszej pomocy oraz ochrony danych osobowych. Obydwa zagadnienia są istotne we współczesnym prawie oraz stanowią znaczącą część naszej pracy.

2.1. Kontekst prawny

W Polsce istnieje obowiązek wezwania służb ratunkowych oraz udzielenia pierwszej pomocy. Odstępstwem jest jedynie sytuacja, gdzie próba jej udzielenia spowoduje uszczerbek na zdrowiu lub życiu osoby ratującej albo osób trzecich. Odstąpienie od tego obowiązku bez wyżej wymienionej przyczyny grozi pozbawieniem wolności do lat trzech na podstawie art. 162 Kodeksu Karnego.

W momencie udzielania pomocy jesteśmy objęci ochroną równą tej, co posiadają funkcjonariusze publiczni. Wynika to z art. 5 z ustawy o Ratownictwie Medycznym. Dodatkowo osoby w Polsce nie powinny obawiać się o konsekwencje prawne w wyniku błędów popełnionych podczas akcji ratunkowej, jeśli wykażemy się dobrą wolą.

2.2. Ochrona danych osobowych

Na autorach i dostawcach ciąży obowiązek administracyjno-prawny chronić dane osobowe zarejestrowanych użytkowników oraz informować o zakresie, jak i sposobie przetwarzania.

Z uwagi na charakter aplikacji dane osobowe zgłaszającego incydent są niezbędne do egzekwowania wykroczenia, w przypadku nieuzasadnionego wezwania służb. Przekazywane dane są identyczne jak te przekazywane podczas telefonicznego zgłoszenia na numer alarmowy. Okres przechowywania danych osobowych osób, które dokonały zgłoszenia lub były poszkodowanymi są zgodne z ustawą o prawach pacjenta i Rzeczniku Praw Pacjenta, czyli przez 20 lat. Dane pracowników są przechowywane zgodnie z kodeksem pracy, czyli 10 lat od momentu wygaśnięcia stosunku pracy.

Użytkownicy naszej aplikacji, którzy nie zgłosili żadnego incydentu lub nie byli poszkodowanymi mają prawo do usunięcia konta (przy usunięciu wymazywane są wszelkie dane udostępnione naszej aplikacji).

Użytkownicy są informowani o tym, jakie dane są przetwarzane, jak jest powód ich udostępniania. W przypadku udostępniania danych w momencie, gdy znajdą się w roli poszkodowanego sami decydują o liczbie udostępnianych danych.

2.3. Codzienne funkcjonowanie

W przypadku incydentu, podczas którego dochodzi do zatrzymania akcji serca poszkodowany nie przeżyje bez rozpoczęcia RKO. Po około 4 minutach mózg obumiera na skutek braku tlenu. Według danych GUS w roku 2022 w miastach powyżej 10 tys. mediana czasu przybycia karetki nie przekroczyła 8 minut. Jest to zdecydowanie za długo dla osób w nagłych wypadkach.

Udzielanie pierwszej pomocy jest niezbędne, jednak wiele osób odstępuje od tego obowiązku. Bardzo często przyczyną jest strach przed zrobieniem krzywdy poszkodowanemu lub brak wiedzy jak udzielić ją poprawnie. Dlatego nasza aplikacja wszystkie poradniki o tym, jak udzielać pierwszej pomocy udostępnia za darmo dla wszystkich, którzy wejdą na stronę, czy ściągną aplikację. W nagłym wypadku osoba może od razu przystąpić do działania, nie tracąc czasu na uzupełnianie danych do rejestracji, czy logowanie.

3. Istniejące systemy pierwszej pomocy i/lub zarządzających służbami ratunkowymi

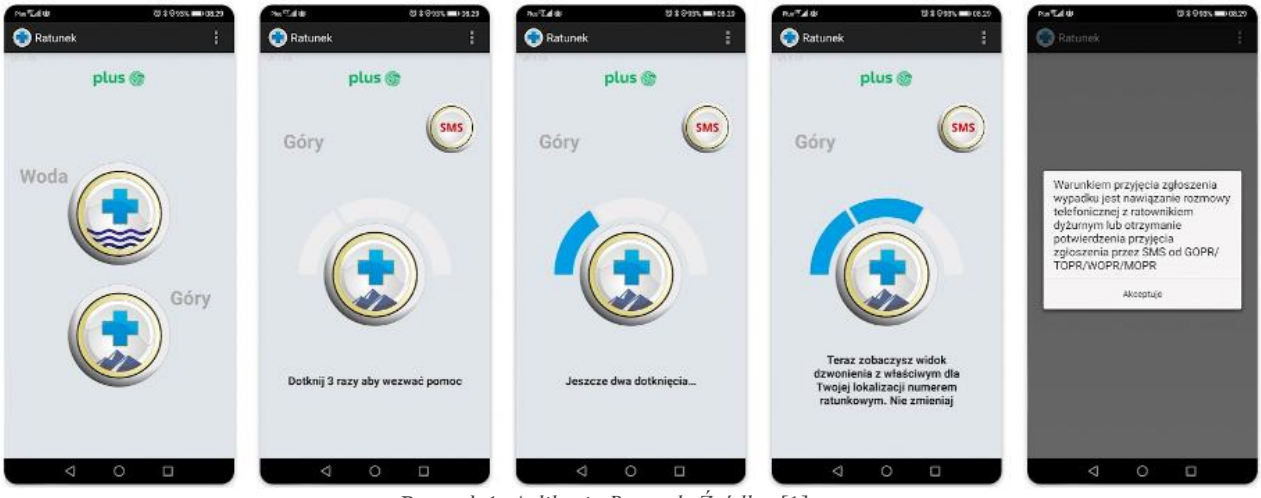
Rynek oferuje nam rozmaite aplikacje medyczne, które stanowią źródło informacji dla pacjenta jak i pracownika ochrony zdrowia oraz takie, które mają na celu zapoznanie użytkownika z pierwszą pomocą w razie zagrożenia. Przed przystąpieniem do projektowania i implementacji systemu będącego tematem niniejszej pracy, nasz zespół przeprowadził analizę rynku konkurencyjnych rozwiązań. Celem tej analizy było zapoznanie się z funkcjonalnościami tych aplikacji, budową oraz przedstawienie ich mocnych i słabych stron. Dalsze rozdziały są rezultatem przeprowadzonej analizy.

3.1. Ratunek – aplikacja przeznaczona głównie dla terenów górskich i morskich.

Aplikacja Ratunek została stworzona przez firmę Polkomtel Sp. z.o.o. służy do zgłaszania sytuacji zagrożenia życia spowodowanych zaginięciem w górach lub wypadku nad wodą. Główną jej funkcjonalnością jest możliwość wezwania pomocy i połączeniem się z odpowiednim numerem służb ratunkowych (GOPR, TOPR, WOPR, MOPR). Ponadto aplikacja umożliwia użytkownikowi:

- Automatyczne udostępnienie lokalizacji użytkownika - w trakcie rozmowy z ratownikiem (w systemie Android) lub bezpośrednio po niej (w systemie Windows, iOS), służby ratunkowe otrzymują SMS z informacją o lokalizacji dzwoniącego z dokładnością do 3 metrów,
- Prowadzenie książeczki medycznej – aplikacja przekazuje ratownikom wpisane przez użytkownika ważne informacje o stanie zdrowia oraz kontakt do zaufanej osoby w razie wypadku.

Ponadto Ratunek jest również udogodnieniem dla ratowników medycznych, którzy dostają informacje o dokładnej lokalizacji zgłaszającego jak również aktualizacje, gdy się przemieszcza. Innymi funkcjami dla ratowników jest również udostępnienie stanu baterii osoby wzywającej pomoc oraz możliwość kontaktu poprzez SMS lub połączenie.



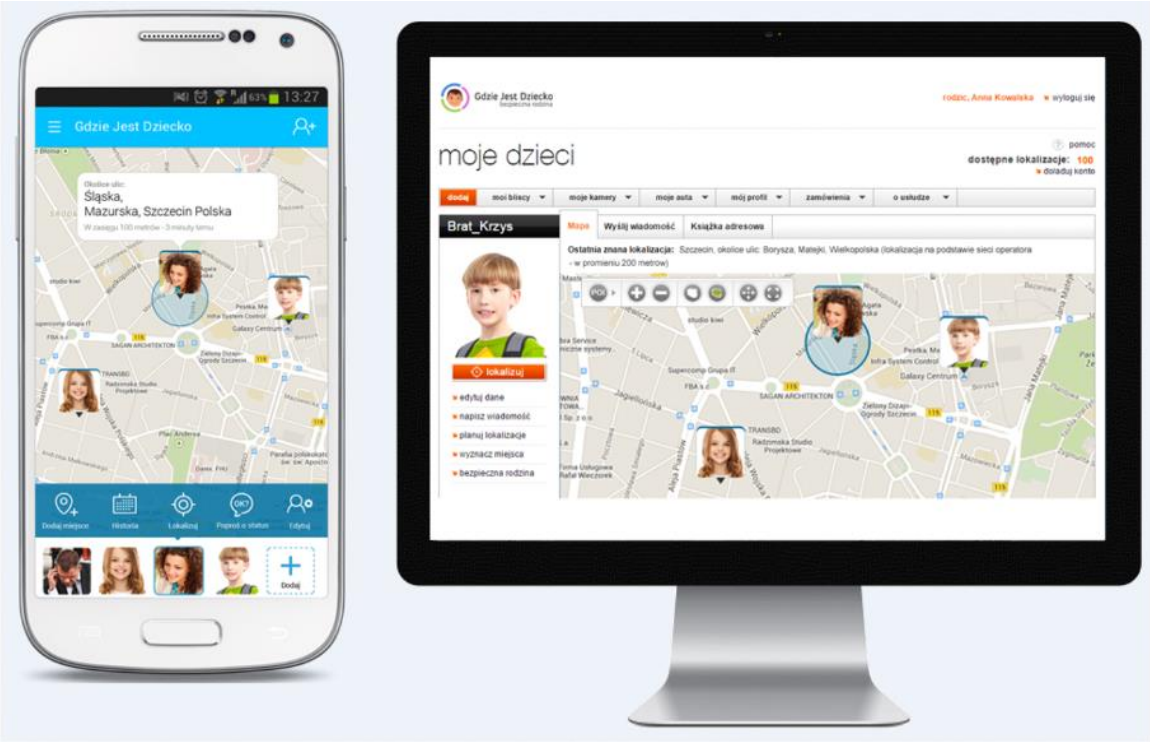
Rysunek 1. Aplikacja Ratunek. Źródło: [1].

Rysunek 1 przedstawia przebieg funkcjonalności - wezwanie pomocy. Użytkownik wybiera odpowiednie służby WOPR/TOPR w celu wezwania pomocy, musi 3 razy potwierdzić swój wybór, aby następnie zostać połączonym z ratownikiem.

3.2 S.O.S. / OK – aplikacja umożliwiająca bardzo szybkie wezwanie pomocy

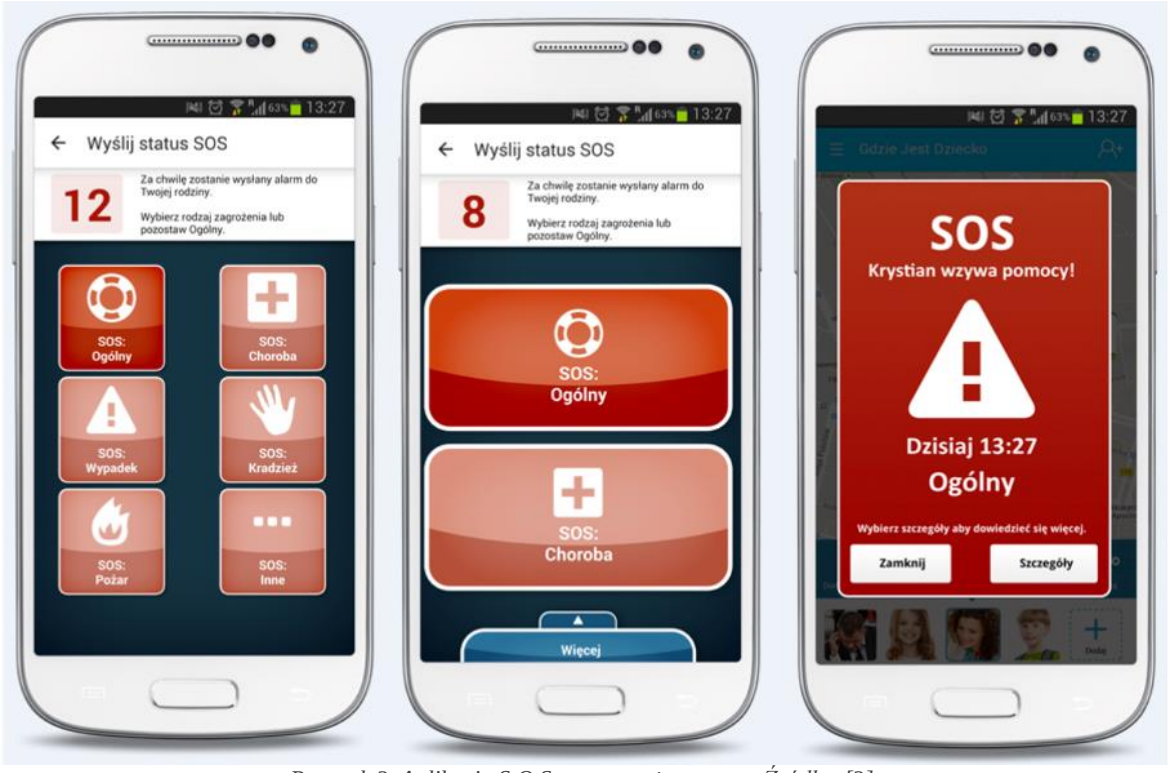
Twórcy aplikacji oferują szybkie wezwanie pomocy w nagłych wypadkach “w ciągu 3 sekund”. S.O.S. można zainstalować na telefonie dziecka lub bliskiej osoby. Dzięki tej aplikacji chroniona osoba może szybko skontaktować się z Rodzicem wysyłając odpowiednie powiadomienia, gdy będzie czuła się zagrożona. Aplikacja oferuje nam wiele funkcjonalności, oto najważniejsze z nich:

- Udostępnianie dokładnej lokalizacji użytkownika aplikacji - dzięki temu Rodzic zawsze wie, gdzie są ich dzieci. Lokalizację Bliskich można oglądać na cyfrowych mapach w aplikacji i na www, co zostało przedstawione na Rysunku 2.



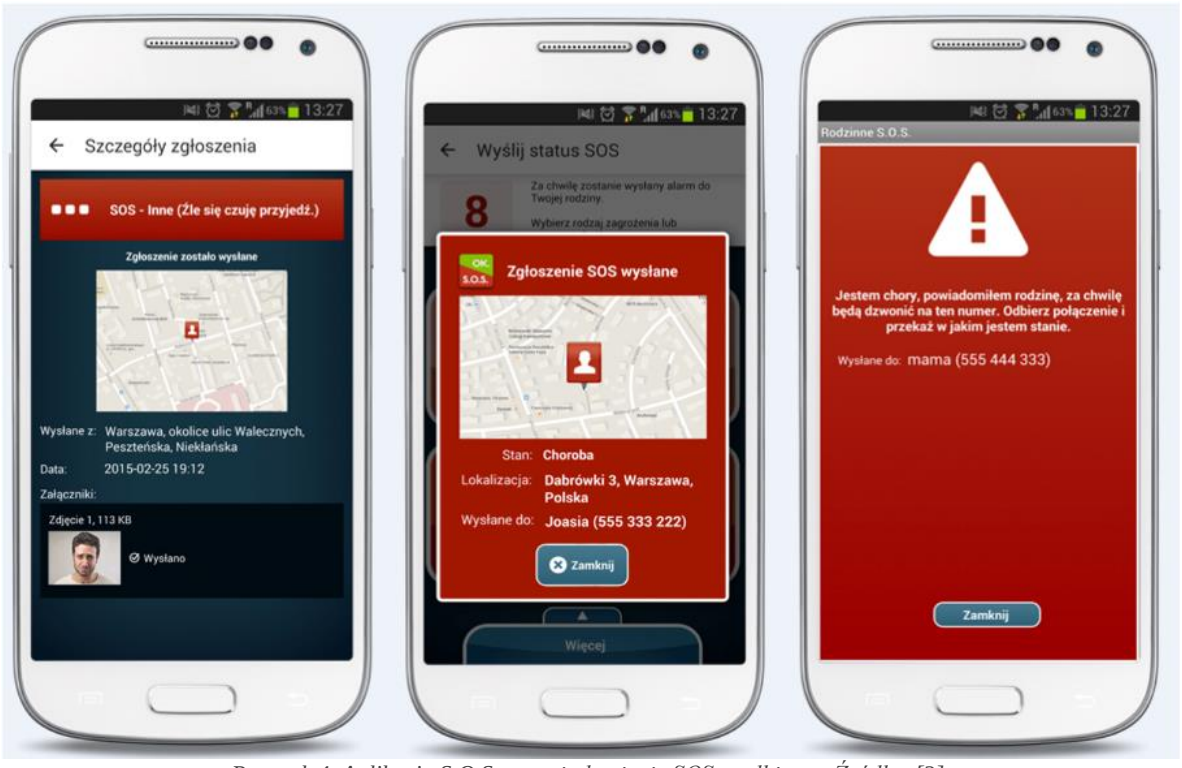
Rysunek 2. Aplikacja S.O.S. – lokalizacja. Źródło: [2].

- Łatwa obsługa aplikacji - użytkownik za pomocą tylko 1 przycisku poinformuje o zagrożeniu, poprzez wysłanie powiadomienia S.O.S, na Rysunku 3 przedstawiony jest schemat tworzenia powiadomienia.



Rysunek 3. Aplikacja S.O.S. - wezwanie pomocy. Źródło: [3].

- Wysłanie sygnału SOS do wielu odbiorców - Rodzica oraz wszystkich osób wskazanych, jako przypisanych odbiorców, a także na numer 112, na Rysunku 4 pokazane jest jak wygląda przykładowe zgłoszenie SOS w aplikacji u odbiorcy,



Rysunek 4. Aplikacja S.O.S. - powiadomienie SOS u odbiorcy. Źródło: [3].

- Podwójne zabezpieczanie zgłoszeń - w celu zapewnienia bezpieczeństwa wszystkie informacje zapisywane są w systemie. Dzięki temu wezwanie pomocy trafi do odbiorców, nawet w

przypadku utraty telefonu przez użytkownika. Rysunek nr 5 przedstawia schemat przechowywania danych.

Rysunek 5. Aplikacja S.O.S. - przechowywanie danych. Źródło: [3].

- Wyświetlanie ważnych miejsc na mapie: użytkownik (tzw. Bliski) uzyskuje dostęp do mapy w aplikacji, na której znajdują się znaczniki z ważnymi punktami pod względem bezpieczeństwa (Policja, Straż Miejska, Pogotowie Ratunkowe, Szpital). Rysunek 6 przedstawia fragment przykładowej mapy

Rysunek 6. Aplikacja S.O.S. – mapa. Źródło [3].

Ponadto aplikacja S.O.S. oferuje nam szereg innych funkcjonalności, które są pomocne dla użytkownika. Jest to możliwość włączenia lokalizacji telefonu 24h na dobę, możliwość lokalizowania i ochrony do 5 osób, łatwość i prostota włączenia usługi lokalizacji – w każdym momencie, za pomocą jednego przycisku w aplikacji mobilnej, gromadzenie danych z lokalizacji. Podsumowując aplikacja S.O.S. została przede wszystkim stworzona do monitorowania miejsca pobytu użytkowników i bez wątpienia ta funkcjonalność jest najbardziej dopracowana.

3.3. Alarm 112 – aplikacja stworzona z myślą o osobach niepełnosprawnych

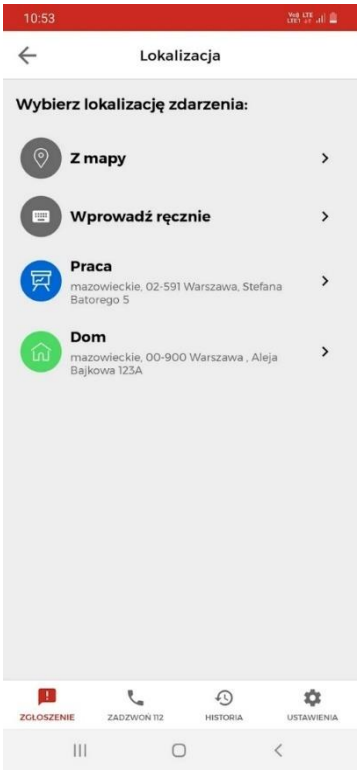
Aplikacja mobilna Alarm112 umożliwia w sytuacji zagrożenia życia, zdrowia, mienia, środowiska, bezpieczeństwa i porządku publicznego przekazanie zgłoszenia alarmowego do Centrum Powiadamiania Ratunkowego. Twórcy tej aplikacji skupili się głównie na pomocy osobom głuchym i niedosłyszącym, bądź też czasowo pozbawionym możliwości mówienia, jak również osobom znajdującym się w sytuacji zagrożenia. Dzięki aplikacji takie osoby będą mogły w sposób dyskretny skontaktować się z Centrum Powiadamiania Ratunkowego (CPR). Głównym atutem aplikacji jest prostota jej obsługi, użytkownik ma możliwość:

- Wyboru odpowiednich piktogramów w celu utworzenia zgłoszenia, które następnie jest generowane i przekazywane odpowiednim służbom ratunkowym wraz z określeniem lokalizacji i rodzaju zdarzenia, schemat działania aplikacji przedstawiony jest na Rysunku 7 i 8.

Rysunek 7. Aplikacja Alarm 112 – tworzenie zgłoszenia (1). Źródło [3].

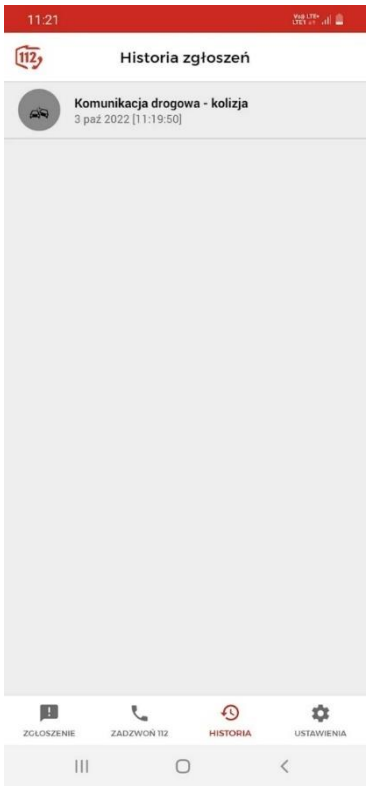
Rysunek 8. Aplikacja Alarm 112 – tworzenie zgłoszenia (2). Źródło [3].

- Wpisanie swojej lokalizacji ręcznie lub wybór z mapy zostało pokazane na Rysunku 9.



Rysunek 9. Aplikacja Alarm 112 - wybór lokalizacji. Źródło [3].

- Przechowywania informacji o zgłoszeniach w historii zgłoszeń aplikacji. Zostało to przedstawione na Rysunku 10.



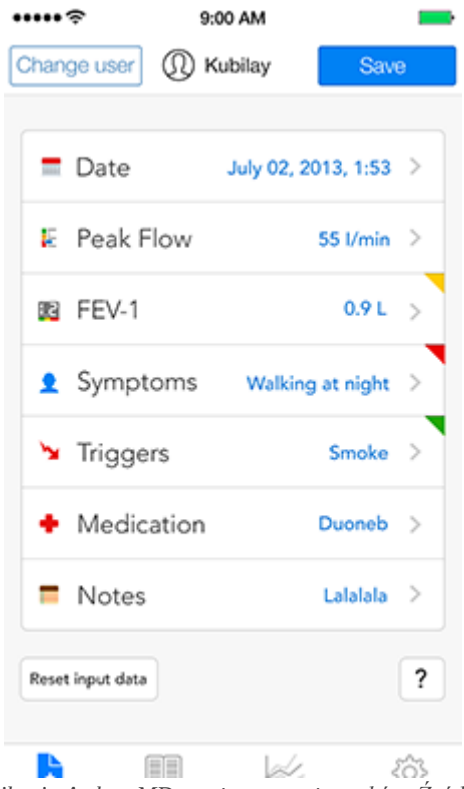
Rysunek 10. Aplikacja Alarm 112 – historia zgłoszeń. Źródło [3].

Ponadto po dokonaniu zgłoszenia możliwa jest dwustronna komunikacja pomiędzy operatorem numeru alarmowego i użytkownikiem za pomocą SMS. Z poziomu aplikacji użytkownik może również zadzwonić na numer alarmowy 112.

3.4. AsthmaMD – aplikacja dla osób chorych przewlekle na astmę i POChP

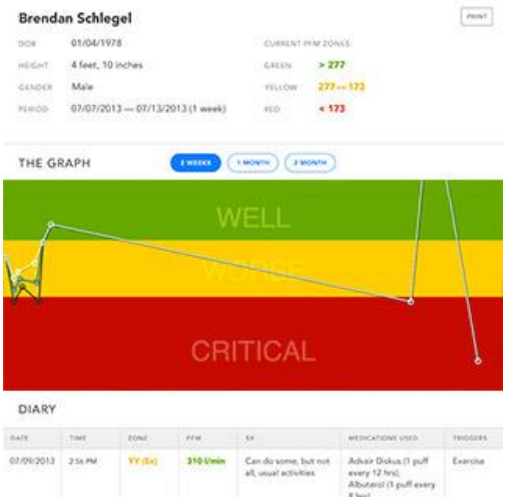
Aplikacja stworzona dla astmatyków i osób chorych na POChP pomagająca w analizie ich choroby – kiedy był atak, co go spowodowało, jakie lekarstwa przyjął chory, jaki był ich efekt. AsthmaMD posiada funkcjonalności takie jak:

- Śledzenie wyzwalaczy ataków, alergeny i zanieczyszczenia, zmiany klimatyczne i pogodowe, które stoją za tymi problemami, patrz Rysunek 11.



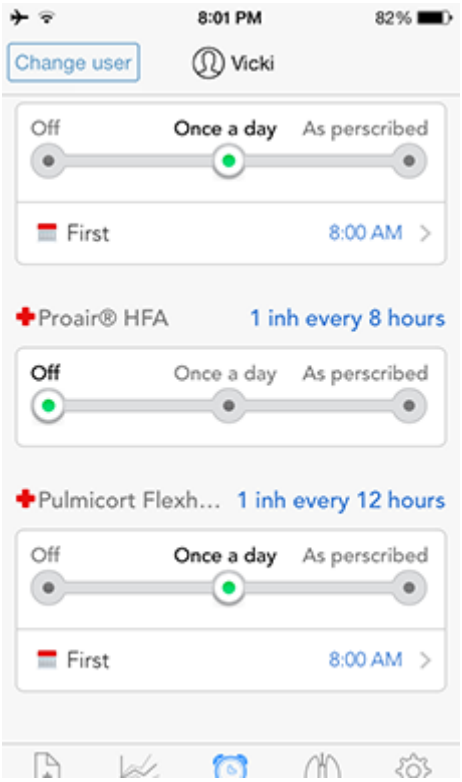
Rysunek 11. Aplikacja AsthmaMD – rejestrowanie ataków. Źródło [4].

Przesyłanie danych chorobowych lekarzowi – co ułatwi postawienie diagnozy i dalsze leczenie, patrz Rysunek 12.



Rysunek 12. Aplikacja AsthmaMD – raport. Źródło [4].

- Możliwość ustawienia przypomnienia na daną godzinę, aby zażyć lekarstwa, patrz Rysunek 13.



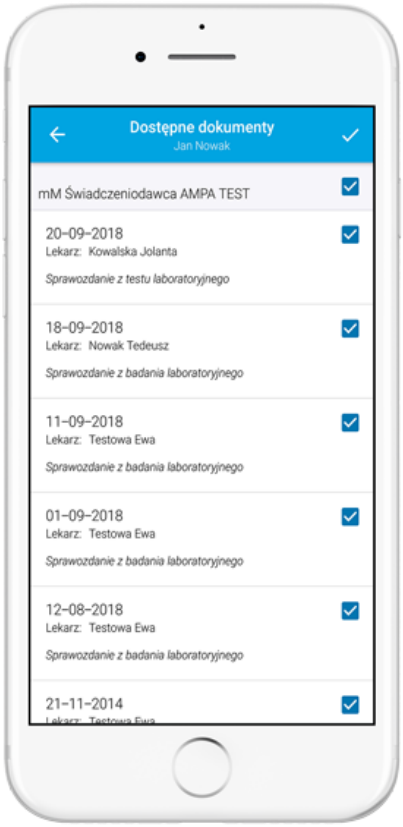
Rysunek 13. Aplikacja AsthmaMD – przypomnienie o zażyciu leków. Źródło [4].

- Anonimowe udostępnianie danych do badań i statystyk.

3.5. Dane medyczne - dostęp do danych medycznych pacjenta

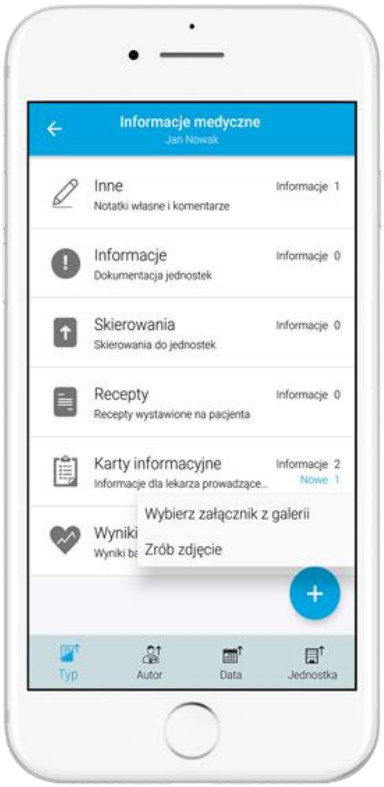
Aplikacja Informacje Medyczne daje dostęp do wyników badań oraz aktualnej dokumentacji pacjenta, opisującej leczenie w szpitalu lub przychodni. Bez potrzeby wizyty w placówce, można mieć także dostęp do informacji dotyczących leczenia bliskich. Główne funkcjonalności aplikacji:

- Pobieranie informacji i dokumentów medycznych z systemów informatycznych jednostek medycznych (mMedica i AMMS produkcji Asseco) zostało przedstawione na Rysunku 14.



Rysunek 14. Aplikacja Informacje medyczne – pobieranie dokumentów. Źródło [5].

- Tworzenie własnych informacji i dokumentów medycznych przez pacjenta, patrz Rysunek 15.



Rysunek 15. Aplikacja Informacje medyczne – tworzenie dokumentacji. Źródło [5].

- Zarządzanie listą poradni, gabinetów szpitali i innych jednostek medycznych, patrz Rysunek 16.

Rysunek 16. Aplikacja Informacje medyczne – placówki. Źródło [5].

- Przekierowanie na strony e-Rejestracji jednostek medycznych z automatycznym logowaniem, patrz Rysunek 17.

Rysunek 17. Aplikacja Informacje medyczne – rejestracja. Źródło [5].

- Integracja z aplikacją Apteczka Domowa w zakresie przypomnień o podaniach leków i terminarzy, patrz Rysunek 18.

Rysunek 18. Aplikacja Informacje medyczne – apteczka. Źródło [5].

3.6. Aplikacje dla ratowników medycznych i dyspozytorów

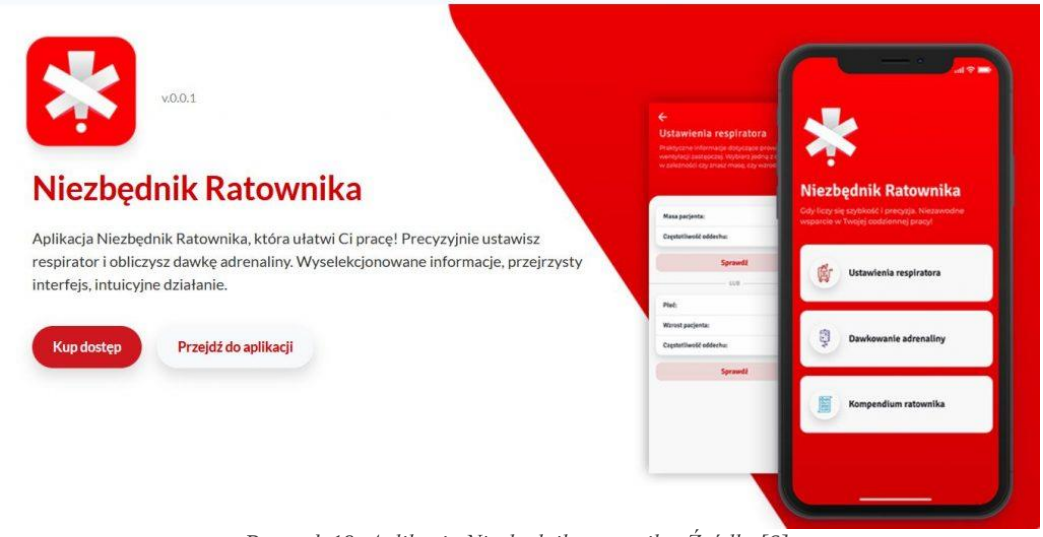
Powyższe aplikacje zostały głównie stworzone dla prywatnych użytkowników, którzy nie pracują w branży medycznej oraz nie posiadają żadnego doświadczenia w ratownictwie medycznym. Nasza aplikacja posiada również wersje dla dyspozytorów i ratowników medycznych. W tym celu przeszukaliśmy dodatkowo takie oferty aplikacji, które posiadają funkcjonalności dopasowane do potrzeb pracowników służby zdrowia.

3.6.1. Niezbędnik Ratownika - aplikacja pod patronatem Na Ratunku i serwisu ratownicy24.pl

Niezbędnik Ratownika (patrz Rysunek 19) jest zbiorem poradników oferujących rozmaite informacje, które są przydatne ratownikom w trakcie ich służby. Jest również źródłem materiałów szkoleniowych dla nowego lub przyszłego pracownika służby zdrowia. Aplikacja zawiera m.in.:

- Precyzyjny przelicznik ustawień respiratora, pod względem wagi i wzrostu pacjenta.
- Informacje o dawkowaniu adrenaliny w zależności od użytego sprzętu.
- Informacje o ustawianiu wartości PEEP na początek i w przypadku obrzęku płuc.

oraz wiele innych informacji z zakresu ratownictwa medycznego.



Rysunek 19. Aplikacja Niezbędnik ratownika. Źródło [6].

Aplikacja Niezbędnik Ratownika posiada wersję dostępną na iOS i Android oraz działa w trybie offline. Minusem aplikacji jest płatna subskrypcja na 12 miesięcy oraz możliwość używania jej tylko na jednym urządzeniu.

3.6.2. Ratownictwo medyczne algorytmy – Bit Med. Marcin Andrzejczak

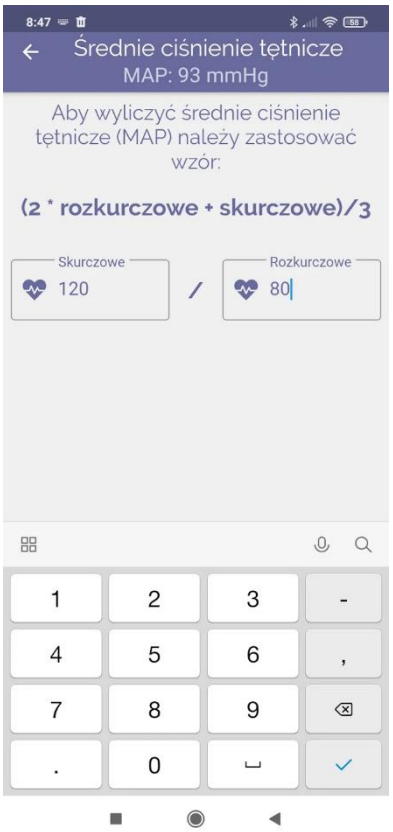
Aplikacja Ratownictwo medyczne algorytmy została stworzona przez Marcina Andrzejczaka. Jest ona zbiorem podstaw ratownictwa medycznego oraz algorytmów postępowania w nagłych stanach opisanych krok po kroku. Każdy przedstawiony w niej algorytm postępowania jest oparty o odpowiednie prace naukowe. Jest ona dostępna w wersji na Android oraz iOS. Główna strona aplikacji została przedstawiona na Rysunku 20.

Rysunek 20. Aplikacja Ratownictwo medyczne algorytmy – strona główna. Źródło [7].

- Przeglądanie poradników np. poradnik wieszania kroplówki, patrz Rysunek 21.

Rysunek 21. Aplikacja Ratownictwo medyczne algorytmy - poradnik. Źródło [7].

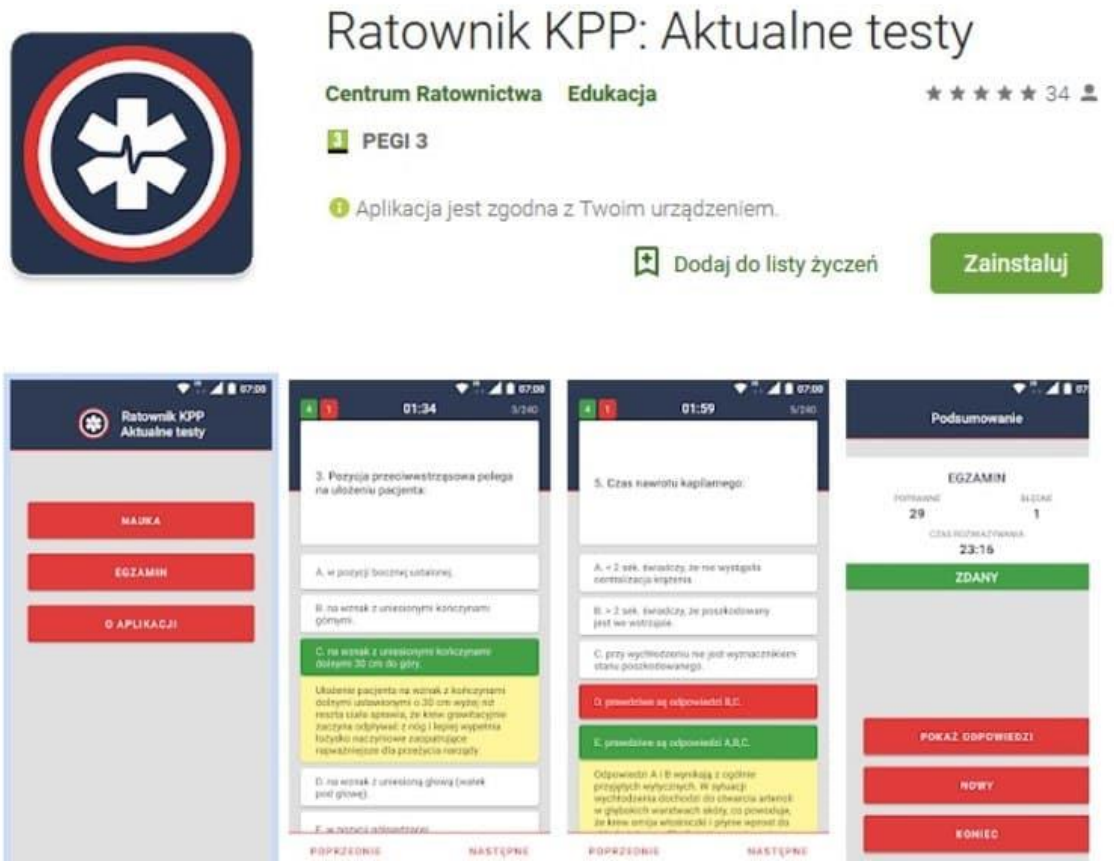
- Przelicznik średniego ciśnienia tętniczego, patrz Rysunek 22.



Rysunek 22. Aplikacja Ratownictwo medyczne algorytmy - przelicznik. Źródło [7].

3.6.3. Ratownik KPP: aktualne testy – Centrum Ratownictwa

Aplikacja Ratownik KPP: aktualne testy została stworzona przez Centrum Ratownictwa i jest dostępna dla systemów Android oraz iOS. Aplikacja składa się z dwóch modułów – NAUKA i EGZAMIN. Moduł NAUKA zawiera pytania wraz ze wskazanymi prawidłowymi odpowiedziami i komentarzami do nich. Natomiast w module EGZAMIN użytkownik ma za zadanie wskazać prawidłową odpowiedź do losowo wybranych pytań.



Rysunek 23. Aplikacja Ratownik KPP: aktualne testy. Źródło: [8].

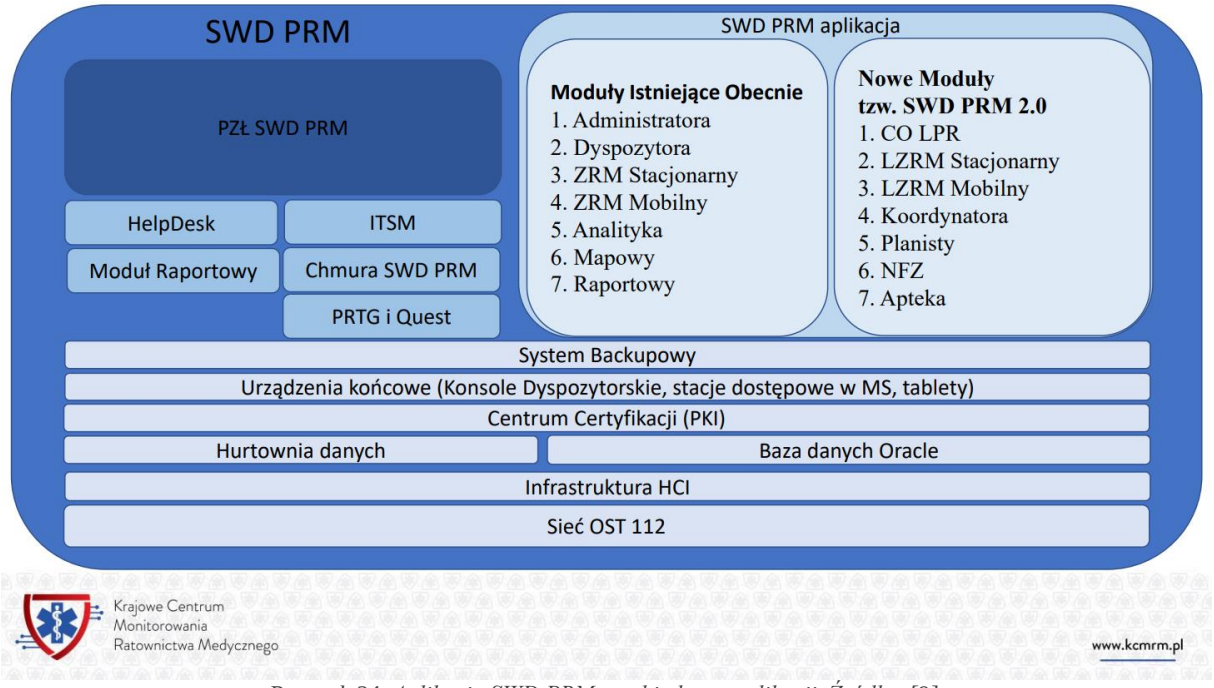
3.7. System Wspomagania Dowodzenia Państwowego Ratownictwa Medycznego (SWD PRM)

Główną funkcjonalnością tego systemu jest przyjmowanie zgłoszeń alarmowych i powiadomień o zdarzeniach z numerów alarmowych takich jak 112 i 999. Ponadto umożliwia:

- zarządzanie zgłoszeniami i zdarzeniami,
- dysponowanie zespołów ratownictwa medycznego,
- rejestrowanie zdarzeń medycznych,
- lokalizację poszczególnych zdarzeń, miejsc pobytu ZRM i ich statusów na mapie,
- monitorowanie i zarządzanie siłami i środkami,
- prowadzenie dokumentacji medycznej.

SWD PRM obsługuje 27 dyspozytorni medycznych, 1703 zespołów ratownictwa medycznego, 216 dysponentów ZRM, 1091 miejsca stacjonowania ZRM, 28 196 użytkowników.

Aplikacja SWD PRM obejmuje moduły: administratora, dyspozytora, ZRM stacjonarny, ZRM mobilny, analityka, mapowy, raportowy. Ponadto istnieją nowe moduły aplikacji dodane w wersji SWD PRM 2.0. Są to: CO LPR, LZRM Stacjonarny, LZRM Mobilny, Koordynatora, Planisty, NFZ, Apteka. Schemat architektury przedstawia Rysunek 24.



Rysunek 24. Aplikacja SWD PRM: architektura aplikacji. Źródło: [9].

3.7.1. Moduł administratora

Umożliwia administrowanie danymi poprzez zarządzanie kontami i uprawnieniami użytkowników, siłami i środkami dysponentów zespołów ratownictwa medycznego, konfiguracją dostępów do poszczególnych modułów. Ponadto administrator ma uprawnienia do modyfikacji parametrów systemu, a także zarządzanie aktualizacjami pozostałych modułów.

3.7.2. Moduł dyspozytora

Dyspozytor posiada uprawnienia do obsługi zgłoszeń alarmowych oraz powiadomień o zdarzeniach przez dyspozytorów medycznych oraz wojewódzkiego koordynatora ratownictwa medycznego. Moduł ten oferuje funkcjonalności takie jak:

- przyjmowanie zgłoszeń alarmowych z centrów powiadamiania ratunkowego oraz bezpośrednio z infolinii numeru alarmowego,
- dysponowanie zespołów ratownictwa medycznego do konkretnych zdarzeń,
- prezentacje miejsca zdarzenia i pozycjonowanie ZRM z wykorzystaniem Uniwersalnego Modułu Mapowego.

3.7.3. Moduł ZRM stacjonarny

Został stworzony z myślą o Zespołach Ratownictwa Medycznego, aby usprawnić przyjmowanie i wykonywanie zleceń oraz aktualizacji i wydruku dokumentów medycznych. Umożliwia zatwierdzanie dokumentacji medycznej po obsłudze zdarzenia, rejestrowanie początku i końca dyżuru oraz informowanie o gotowości ZRM do przyjmowania zleceń wyjazdów.

3.7.4. Moduł ZRM mobilny

Dostosowany do pracy na terminalu mobilnym w karetce. Jego najważniejsze funkcjonalności to:

- przyjmowanie i realizacja zdarzeń przekazywanych przez dyspozytornie medyczne,
- przyjmowanie i realizację zdarzeń przekazywanych przez dyspozytornie medyczne oraz jej wydruk,
- wyświetlanie trasy dojazdu do miejsca zdarzenia przy użyciu nawigacji.

3.7.5. Moduł Analityka

Został stworzony do przeglądania zdarzeń, Kart Zlecenia Wyjazdu oraz Kart Medycznych Czynności Ratunkowych. Ponadto jest możliwość śledzenia trasy pojazdów oraz modyfikacji danych KZW i KMCR.

3.7.6. Moduł Raportowy

Moduł Raportowy - Apex, wykorzystywany jest do generowania raportów z aktywności operacyjnej dysponentów zespołów ratownictwa medycznego. Główną funkcjonalnością aplikacji jest generowanie raportów w zależności od uprawnień użytkownika.

3.7.7. Moduł Mapowy

Moduł mapowy usprawnia pracę dyspozytora oferując:

- wizualizację miejsca zdarzenia,
- wizualizację przybliżonej lokalizacji zgłaszającego z wykorzystaniem danych pobranych z Platformy Lokalizacyjno–Informacyjnej z Centralną Bazą Danych,
- wizualizację w czasie rzeczywistym położenia ZRM-ów oraz ich status,
- wizualizację w czasie rzeczywistym położenia lotniczych zespołów ratownictwa medycznego,
- oznaczanie miejsc zdarzenia.

3.8. Podsumowanie

Wszystkie wyżej wymienione przykłady istniejących aplikacji posiadają znaczącą wadę. Są one głównie przeznaczone do użytku prywatnego, amatorskiego, dla pojedynczych jednostek. Ponadto w wersji dla prywatnego użytkownika brakuje im funkcjonalności tj.: tworzenie konta pacjenta z wyszczególnieniem chorób i alergii, możliwości kategoryzacji zdarzeń i ich dokładnego opisu wraz z liczbą poszkodowanych oraz możliwości dodania osoby zaufanej.

Aplikacja Gary posiada wersję bogatą w funkcjonalności dla dyspozytora i ratownika medycznego, co jest niespotykane w systemach dostępnych na rynku. W wersji dla dyspozytorów lub ratowników medycznych niniejsza aplikacja oferuje pełny system zarządzania karetkami i zgłoszeniami, czego nie znajdziemy w aplikacjach już dostępnych.

4. Propozycja systemu obsługi incydentów ratunkowych

Niniejszy rozdział dotyczy omówienia rozwiązania do wdrożenia systemu “Gary”, wymagań funkcjonalnych oraz niefunkcjonalnych. Jego treść została wsparta diagramami przypadków użycia, stanów oraz diagramem klas. Rozdział ten nie zawiera opisów źródeł danych poszczególnych pól i zasad walidacji - te informacje zostaną zawarte w Fazie projektowania. Gary ma być aplikacją pomagającą w ratowaniu życia - narzędziem, które w chwili kryzysowej umożliwia płynne przejście do wykonania kroków niezbędnych dla udzielenia pomocy osobie poszkodowanej

4.1. Cele

Celem systemu jest stworzenie narzędzia wspierającego wszystkich uczestników zdarzenia, w którym został poszkodowany człowiek, a jego życie i zdrowie jest zagrożone.

4.2. Założenia

System Gary w swoim założeniu ma być kompleksowym systemem do ratowania życia. Do jego głównych założeń należy: rejestrowanie w jednym systemie zdarzeń od początku do końca (tj. od zgłoszenia wypadku do wypisania raportu i przewiezienia poszkodowanego do szpitala). Dzięki temu, cała procedura pomocy poszkodowanym, wraz ze wszystkimi danymi, znajdować się będzie w jednym miejscu.

W założeniach System uwzględnia 3 kluczowe elementy wpływające na jego sprawne funkcjonowanie:

1. Działanie pod presją czasu.
2. Wysoki stres wynikający z sytuacji kryzysowej.
3. Zróżnicowanie użytkowników zarówno pod względem kompetencji jak i odporności na sytuacje kryzysowe.

Działania w sytuacjach kryzysowych oznaczają, że czas jest niezwykle istotnym elementem – szybkość reakcji, właściwa kolejność zdarzeń może decydować o zdrowiu lub życiu poszkodowanych, dlatego założono, że podstawową cechą systemu będzie intuicyjność i czytelność poleceń. System będzie podpowiadał użytkownikom kolejne kroki, a tym samym wymuszał optymalną sekwencję zdarzeń.

Grupy użytkowników aplikacji będą bardzo zróżnicowane, dlatego w tworzeniu systemu zwrócono uwagę na kilka cech:

- symplifikacja przekazu zarówno w odniesieniu do grafiki jak i zastosowanych poleceń oraz instrukcji,
- profilowanie treści dostosowanie do roli użytkowników,
- kompleksowość informacji w jednym miejscu.

Takie podejście gwarantuje nie tylko sprawne działanie, ale i odpowiednią logikę działań, niezwykle istotną w sytuacjach zagrożenia.

4.3. Opis

Główną funkcjonalnością systemu będzie zgłaszanie do dyspozytora wypadków. Następnie wypadki są przez niego analizowane, co umożliwia podjęcie decyzji o wezwaniu służb medycznych. Kluczową różnicą między utworzeniem zgłoszenia w aplikacji, a zwykłą rozmową telefoniczną jest to, że użytkownik dodatkowo wypełnia ankietę z najbardziej istotnymi informacjami (w tym lokalizacją, którą aplikacja potrafi sama odczytać). Dzięki temu dyspozytor od razu posiada wszystkie potrzebne informacje i może natychmiast wysłać najodpowiedniejszą dla tego konkretnego przypadku karetkę (system pod uwagę dla oceny bierze położenie oraz typ i status karetki).

Każdy użytkownik w systemie Gary będzie miał możliwość dołączenia do swojego profilu konkretnych informacji medycznych, takich jak: grupa krwi, przewlekłe choroby, alergie oraz numer

osoby zaufanej. Dzięki temu, w przypadku niefortunnego zdarzenia, inny użytkownik będzie mógł zeskanować indywidualną opaskę poszkodowanego i załączyć ją w ankiecie. To wszystko ma na celu przekazanie ratownikom medycznym dodatkowych informacji o osobie poszkodowanej Dzięki numerowi osoby zaufanej, odpowiednie osoby zostaną poinformowane w razie wypadku.

Jeśli chodzi o funkcjonalności dodatkowe, każdy użytkownik będzie miał możliwość przeglądania różnego rodzaju poradników, dzięki którym dowie się, jak reagować w konkretnych sytuacjach zagrożenia życia oraz poradników udzielających porad w zakresie profilaktyki.

Funkcjonalności kierownika karetek ograniczać się będą w systemie do modyfikowania informacji o karetkach i ratownikach medycznych, przydzielaniu ratowników do karetek oraz do tworzenia list wyposażenia dla poszczególnych typów karetek.

Dla ratownika medycznego przewidziana jest aplikacja mobilna, zaś dla dyspozytora i kierownika karetki przewidziana jest aplikacja webowa, użytkownik natomiast będzie posiadał zarówno aplikację mobilną, jak i aplikację webową.

4.4. Słownik pojęć

Ankieta
Zestaw informacji podawanych przez osobę zgłaszającą zdarzenie; uwzględnia informacje o tym czy poszkodowany jest przytomny, czy oddycha, datę i godzinę oraz krótki opis zdarzenia. Możliwość dodania załącznika: np. zdjęć.

Dodatkowa pomoc
Wezwane w trakcie realizacji dodatkowe służby: policja, straż pożarna, pogotowie energetyczne, karetka, helikopter.

Dokumentacja medyczna
Informacje medyczne o użytkowniku systemu: grupa krwi, choroby przewlekłe, alergie.

Dostępność karetki
Jedna z cech karetki, uwzględnia informacje: typ, przypisanych ratowników oraz dyżur. Pozwala określić gotowość karetki do prowadzenia akcji ratunkowych. Uwzględniamy typy dostępności:

- dostępna (uwzględniania w przypisywaniu do zgłoszenia),
- tankowanie (chwilowo niedostępna, w trakcie uzupełniania paliwa),
- zajęta (przypisana do zgłoszenia),
- awaria (brak możliwości przypisania do karetki z powodów technicznych),
- przerwa (przerwa w pracy).

Dyspozytor
Osoba zarządzająca zgłoszeniami i pracą karetek. Jej głównym zadaniem jest przyjęcie zgłoszenia od użytkownika i przypisanie karetki do zgłoszenia pod względem jej typu i odległości od miejsca zgłoszenia. Nadzoruje cały proces przebiegu zgłoszenia.

Dyżur
Przedział pracy uwzględniający godzinę początku i końca pracy Pracowników.

Gość
Osoba niezalogowana do systemu (brak własnego konta), posiadająca ograniczone funkcjonalności systemu.

Grupa krwi
Wyróżniamy: A+, AB+, 0+, B-, B+, A-, AB-, 0-, nieznana.

Karetka

Uprzywilejowany środek transportu dysponowany na miejsce zgłoszenia, przeznaczony do udzielania pomocy, przewozu chorych lub rannych z miejsca zdarzenia do szpitala.
Wyróżniamy 5 rodzajów karetek (określające ich przeznaczenie):

- rodzaj P - karetka podstawowa (dostosowana do podstawowych wypadków, urazów i zachorowań, zespół składa się z co najmniej dwóch ratowników lub pielęgniarek),
- rodzaj S - karetka specjalistyczna (karetka specjalistyczna, używana w stanach zagrożenia życia, 3 osobowy zespół w tym co najmniej 1 lekarz, zaawansowana aparatura i sprzęt medyczny),
- rodzaj N - karetka neonatologiczna (przystosowana do transportu noworodków i niemowląt do 1. roku życia),
- rodzaj T - karetka transportowa (do transportu poszkodowanych bez użycia zaawansowanego wyposażenia, zespół składa się z kierowcy i ratownika medycznego,
- rodzaj Covid - przystosowana do transportu poszkodowanych zakażonym wirusem SARS-Cov-2 lub z podejrzeniem zakażenia.

Wyróżniamy 3 typy karetki (określające stopień zaawansowania świadczonych usług):

- typ A - transport poszkodowanych,
- typ B - karetka ratunkowa (posiada odpowiednie wyposażenie do podstawowego udzielenia pomocy i monitorowania stanu zdrowia poszkodowanego oraz możliwość transportu poszkodowanego),
- typ C - najbardziej zaawansowany typ karetki - możliwość transportu pacjentów, zaawansowanego leczenia i monitorowania stanu poszkodowanego.

Do każdej karetki przypisanie jest wyposażenie:
zużywające się stopniowo rzeczowe składniki majątku karetki. Określamy ich ilość, datę oraz jednostkę miary. Istnieje możliwość uzupełnienia wyposażenia lub sprawdzenia jego zużycia.

Kierownik karetek
Osoba zarządzająca karetkami i ratownikami. Jej zadaniem jest zarządzanie listą wyposażenia karetki i oraz przypisywanie ratowników medycznych do karetki.

Lokalizacja
Określa adres lub współrzędne geograficzne placówki.

Niebezpieczny pacjent
Osoba poszkodowana, która wykazuje zachowanie utrudniające pracę pracownikom służby zdrowia (awanturuje się, jest agresywny itd.).

Opaska identyfikująca
Opaska przeznaczona dla użytkownika systemu, z unikalnym kodem użytkownika, umożliwia skanowanie w celu identyfikacji użytkownika.

Osoba Zaufana
Wybrana przez użytkownika aplikacji osoba, mająca zgodnie z prawem możliwość uzyskania informacji o użytkowniku w razie zgłoszenia.

Placówka
Miejsce świadczące pomoc osobom poszkodowanym - szpital, policja, straż pożarna. W przypadku szpitala uwzględniamy czy dostępne są wolne miejsca.

Położenie
Określa współrzędne geograficzne poszkodowanych i karetek.

Poradnik

Krótki film instruktażowy dostępny dla użytkownika systemu, mówiący o postępowaniu w sytuacjach kryzysowych i udzielaniu pierwszej pomocy.

Poszkodowany
Osoba, która potrzebuje udzielenia pomocy.

Poziom zagrożenia
Uwzględnia stopień zagrożenia życia (1-10, 10 - maksymalny), sytuacje wymagające zaawansowanych świadczeń medycznych. Ma na celu jak najszybsze reagowanie na sytuacje wymagające pilnej pomocy i dopasowanie personelu i wyposażenia do konkretnej sytuacji.

Pracownik
Osoba posiadająca rolę w systemie taką jak: "Dyspozytor", "Kierownik Karetek", "Ratownik".

Raport o poszkodowanym
Generowany przez ratownika medycznego, zbiór informacji o poszkodowanym, wypełniany przez ratownika medycznego, uwzględniający czy poszkodowany oddycha, czy jest przytomny, czy ma puls, datę, podane medykamenty i wykonane czynności oraz ewentualny opis. Po wypełnieniu, dodane zostają informacje z dokumentacji medycznej pacjenta, po czym raport zostanie zapisany w bazie danych naszego systemu, a następnie wysłany do szpitala.

Ratownik medyczny
Osoba udzielająca pomocy poszkodowanemu. W skład zadań wchodzi: sporządzenie raportu o poszkodowanym, kontakt ze szpitalem, oznaczenie poszkodowanego jako niebezpieczny pacjent.

Recenzja
Analiza i ocena poradnika sporządzana przez użytkownika, obejmuje ocenę w skali 1-5 i krótki opis. Możliwość zmiany oceny.

System
Zbiór funkcjonalności zawartych w aplikacji webowej i mobilnej.

Użytkownik
Osoba zalogowana do systemu, posiadająca określony zakres funkcjonalności w zależności od pełnionej przez niego roli w systemie.

Zgłoszenie
Jest ono generowane w aplikacji przez użytkownika. Zawiera zbiór informacji opisujących przyczyny medyczne zgłoszenia i uwzględniające:

- datę,
- poziom zagrożenia,
- lokalizację,
- liczbę poszkodowanych,
- ankietę

Rozróżniamy trzy statusy zgłoszenia, które mogą zostać zmienione przez dyspozytora:

- otwarte,
- zamknięte,
- odrzucone,
- przyjęte

Do zgłoszenia na zlecenie dyspozytora, mogą zostać wezwane dodatkowe służby.
Po obsłużeniu zgłoszenia jest możliwość jego zamknięcia.
Po obsłudze generowany jest raport o poszkodowanych.

4.5. Użytkownicy systemu

1. Użytkownik
   - 1.1. Przechowywane dane:
     - 1.1.1. Imię,
     - 1.1.2. Nazwisko,
     - 1.1.3. Rok urodzenia,
     - 1.1.4. Dodatkowe informacje:
       - 1.1.4.1. grupa krwi,
       - 1.1.4.2. historia alergii,
       - 1.1.4.3. choroby przewlekłe.
     - 1.1.5. Numer telefonu,
     - 1.1.6. Email,
     - 1.1.7. Hasło,
     - 1.1.8. Numer opaski.
   - 1.2. Dostępne funkcjonalności:
     - 1.2.1. Zarządzaj informacjami podanymi na profilu,
       Użytkownik ma możliwość edycji informacji podanych na profilu - dotyczy się to zarówno danych osobowych, danych kontaktowych, dodatkowych informacji jak i hasła do procesu logowania w aplikacji.
     - 1.2.2. Utwórz zgłoszenie,
       Aby utworzyć zgłoszenie, użytkownik musi wejść w zakładkę “Utwórz zgłoszenie”. Następnie wyświetla się ankieta z polami. Użytkownik musi wypełnić ankietę w której istnieją różne rodzaje pól. Są nimi zarówno *checkboxy* jak i pola tekstowe. Cała funkcjonalność kończy się wraz z kliknięciem opcji “prześlij”
     - 1.2.3. Oglądaj poradnik,
       W aplikacji, dostępne są dla użytkownika różnego rodzaju poradniki, po przyswojeniu których, użytkownik będzie posiadał wiedzę, jak radzić sobie w kryzysowych sytuacjach. Aby zapoznać się z poradnikiem, użytkownik musi wejść w sekcję poradników, a następnie wybrać jedną z dostępnych tam opcji. Na końcu poradnika, jest opcja pozostawienia oceny, na którą składa się ocena od 1 do 5 oraz opcjonalne miejsce na komentarz.
     - 1.2.4. Przeglądaj mapę.
       Użytkownik ma możliwość w aplikacji przeglądania mapy, na której znajdują się zarówno placówki takie jak: szpital, posterunek policji, straż pożarna jak i miejsca, które zostały oznaczone jako niebezpieczne sytuacje.

2. Dyspozytor
   - 2.1. Przechowywane informacje:
     - 2.1.1. Imię,
     - 2.1.2. Nazwisko,
     - 2.1.3. Email,
     - 2.1.4. Hasło,
     - 2.1.5. Rok urodzenia,
     - 2.1.6. Dyżury.
   - 2.2. Dostępne funkcjonalności:
     - 2.2.1. Rozpocznij dyżur,

Po zalogowaniu się do aplikacji przez dyspozytora, pojawia się okno z zapytam czy rozpocząć dyżur. Kliknięcie w opcję “Tak”, oznacza, rozpoczęcie dyżuru. Informacja o rozpoczęciu dyżuru zapisywana jest w bazie danych.

2.2.2. Zakończ dyżur,
Przy wylogowywaniu się z aplikacji przez dyspozytora, pojawia się okno z zapytaniem, czy zakończyć dyżur. Kliknięcie w opcję “Tak”, oznacza, zakończenie dyżur, a kliknięcie w opcję “Nie”, oznacza zmianę statusu na “w przerwie”

2.2.3. Zarządzaj zgłoszeniami,
Główną funkcjonalnością dyspozytora, jest zarządzanie zgłoszeniami. Po kliknięciu w opcję “zgłoszenia”, wyświetla się lista zgłoszeń w widoku tabeli. Po wybraniu jednego ze zgłoszenia pojawiają się dostępne opcje do zarządzania danym, konkretnym zgłoszeniem. Pierwszą czynnością, którą dyspozytor musi zrobić podczas zarządzanie zgłoszeniem, jest zdecydowanie, czy zgłoszenie powinno zostać zaakceptowane i czy status jego powinien zmienić się na “otwarte”. W przypadku braku akceptacji, zgłoszenie jest zamykane i nie można zrobić z nim nic więcej. Następne czynności, które dyspozytor może zrobić dla zaakceptowanego zgłoszenia to, ustalenie poziomu zagrożenia.

2.2.4. Przeglądaj mapę.
Na mapie wyświetla się rozmieszczenie placówek, otwarte oraz zaakceptowane zgłoszenia oraz aktualne pozycje wszystkich karetek.

3. Ratownik medyczny

3.1. Przechowywane informacje:

3.1.1. Imię,
3.1.2. Nazwisko,
3.1.3. Email,
3.1.4. Hasło,
3.1.5. Rok urodzenia,
3.1.6. Dyżury,
3.1.7. Lokalizacja - podczas dyżuru.

3.2. Dostępne funkcjonalności:

3.2.1. Rozpocznij dyżur,
Po zalogowaniu się do aplikacji przez ratownik, pojawia się okno z zapytam czy rozpocząć dyżur. Kliknięcie w opcję “Tak”, oznacza, rozpoczęcie dyżuru. Informacja o rozpoczęciu dyżuru zapisywana jest w bazie danych

3.2.2. Zakończ dyżur,
Przy wylogowywaniu się z aplikacji przez ratownika, pojawia się okno z zapytaniem, czy zakończyć dyżur. Kliknięcie w opcję “Tak”, oznacza, zakończenie dyżur, a kliknięcie w opcję “Nie”, oznacza zmianę statusu na “w przerwie”

3.2.3. Zmień status karetki,
Ratownik ma możliwość zmiany statusu karetki na jeden z: “Tankowanie”, “Awaria”, “Przerwa jedzeniowa”, “W akcji”, “Dostępna”

3.2.4. Wyświetl informacje o zgłoszeniu,
Aby wyświetlić informacje o zgłoszeniu, ratownik musi wejść w zakładkę w menu “zgłoszenia”. Następnie zostaną wyświetlone informacje o

poszkodowanym i przyciski związane z pozostałymi funkcjonalnościami 3.2.5 - 3.2.9

3.2.5. Wyświetl najszybszą trasę,
Wybranie tej opcji powoduje otworzenie aplikacji “Google Maps” i wyświetlenie tras przejazdu karetki do miejsca docelowego.

3.2.6. Przeglądaj dokumentację medyczną poszkodowanego,
Opcja ta jest dostępna tylko i wyłącznie wtedy, gdy poszkodowany został zidentyfikowany przy tworzeniu zgłoszenia za pomocą opaski.

3.2.7. Utwórz raport o poszkodowanym,
Po zakończeniu akcji ratunkowej, wyświetlany jest raport do wypełnienia, w którym ratownik opisuje wszystkie czynności oraz medykamenty, które zostały podane poszkodowanemu. Istnieje też możliwość dodania numeru jego opaski. Następnie raport wysyłany jest do szpitala.

3.2.8. Ostrzegaj o niebezpiecznym pacjencie,
Zaznaczenie tej opcji wiąże się z wysłaniem informacji o zachowaniu poszkodowanego do pozostałych służb powiązanych ze zgłoszeniem, oraz o ile pacjent jest w bazie użytkowników i został zidentyfikowany, zapisaniem informacji w bazie danych

3.2.9. Poproś o dodatkowe służby,
W razie problemów, ratownik może wezwać dodatkowe służby poprzez wybranie określonej jednostki służby oraz opcjonalnym podaniem przyczyny tej prośby.

3.2.10. Zarządzaj wyposażeniem
Ratownik ma możliwość podejrzenia w systemie ile wyposażenia posiada przypisana do niego karetka i ewentualnej zmiany ilości medykamentów

4. Kierownik karetki

4.1. Dostępne funkcjonalności:

4.1.1. Zarządzanie listą wyposażenia,
Kierownik ma możliwość tworzenia i modyfikacji już istniejących list wyposażenia, które powinny mieć karetki poszczególnych typów.

4.1.2. Zarządzanie dyżurami ratowników w karetkach,
Kierownik tworzy dyżury ratownikom medycznym poprzez przypisanie ich do odpowiednich karetek.

4.1.3. Przeglądaj informacje o ratownikach
Kierownik ma możliwość wejścia w szczegółowe informacje na temat każdego ratownika, aby podejrzeć jego dane, kwalifikacje oraz dyżury jakie odbył.

4.1.4. Zarządzaj karetkami,
Kierownik ma możliwość dodawania, usuwania oraz modyfikowania każdej istniejącej karetki.

4.1.5. Zarządzaj ratownikami.
Kierownik ma możliwość dodawania, usuwania oraz modyfikowania każdego ratownika medycznego.

5. Gość

5.1. Dostępne funkcjonalności:

5.1.1. Oglądaj poradnik
Po otworzeniu aplikacji w trybie gościa wyświetlają się dostępne w aplikacji poradniki do oglądania.

4.6. Wymagania funkcjonalne

Tabela 2 zawiera wymagania funkcjonalne dotyczące zgłoszenia

Tabela 2. Wymagania funkcjonalne dotyczące zgłoszenia dla projektu Gary. Źródło: Opracowanie własne

L.P.	**Nazwa**	**Opis wymagania**	**Aktorzy**	**Warunki początkowe**
1	Utwórz zgłoszenie	Użytkownik musi wybrać opcję “Utwórz zgłoszenie”, a następnie wypełnić pola w ankiecie. Po ich wypełnieniu, należy kliknąć przycisk “Prześlij”	Użytkownik	
2	Wybierz zgłoszenie	Dyspozytor wybiera z tabeli zgłoszeń zgłoszenie, którym chce zarządzać	Dyspozytor	W systemie muszą znajdować się oczekujące zgłoszenia.
3	Zaakceptuj lub odrzuć zgłoszenie	Dyspozytor podejmuje decyzję, czy powinien zareagować na zgłoszenie, czy raczej powinien je odrzucić i potraktować jako nieuzasadnione wezwanie.	Dyspozytor	Dyspozytor uprzednio musi wybrać z listy zgłoszenie.
4	Ustal priorytet	Po wybraniu konkretnego zgłoszenia, dyspozytor może nadać zgłoszeniu odpowiedni priorytet	Dyspozytor	Zgłoszenie musi zostać wcześniej zaakceptowane.
5	Wybierz karetkę	Dyspozytor wybiera do danego zgłoszenia najlepszą karetkę z listy dostępnych karetek	Dyspozytor	Zgłoszenie musi zostać wcześniej zaakceptowane
6	Wezwij dodatkowe służby	Dyspozytor ma możliwość zadecydowania, czy chce wezwać dodatkowe służby do pomocy przy zgłoszeniu	Dyspozytor	Zgłoszenie musi zostać wcześniej zaakceptowane
7	Wyświetl informacje na temat poszkodowanego	Po kliknięciu w aktywne zgłoszenie, ratownik może przejrzeć informacje o poszkodowanym	Ratownik medyczny	Karetka ratownika musi być wcześniej przypisana do zgłoszenia

8	Wyświetl najszybszą trasę	Po kliknięciu w aktywne zgłoszenie, ratownik może zobaczyć najszybszą trasę do poszkodowanego	Ratownik medyczny	Karetka ratownika musi być wcześniej przypisana do zgłoszenia
9	Napisz raport o poszkodowanym	Przy zakończeniu zgłoszenia ratownik powinien wypełnić raport o poszkodowanym	Ratownik medyczny	Poszkodowany musi zostać przyjęty w szpitalu
10	Oznacz poszkodowanego jako osobę niebezpieczną	Po wejściu w aktywne zgłoszenie, ratownik może oznaczyć poszkodowanego jako osobę niebezpieczną	Ratownik medyczny	Karetka ratownika musi być wcześniej przypisana do zgłoszenia
11	Poproś o dodatkowe służby	Po wejściu w aktywne zgłoszenie, ratownik ma możliwość poproszenia o dodatkowe służby	Ratownik medyczny	Karetka ratownika musi być wcześniej przypisana do zgłoszenia

Tabela 3 przedstawia funkcjonalności systemu związane z dyżurami pracowników.

Tabela 3. Funkcjonalności związane z dyżurami. Źródło: Opracowanie własne.

L.P.	Nazwa	Opis wymagania	Aktorzy	Warunki początkowe
1	Rozpocznij dyżur	Pracownik po zalogowaniu się do aplikacji rozpoczyna dyżur klikając pole “Rozpocznij dyżur”	Ratownik medyczny, dyspozytor	Pracownik musi być zalogowany
2	Zakończ dyżur	Pracownik przy wylogowywaniu się, wybiera opcję “Zakończ dyżur”	Ratownik medyczny, dyspozytor	Pracownik musi być zalogowany oraz dyżur musi obecnie trwać.

Tabela 4 przedstawia funkcjonalności systemu związane z przeglądaniem poradników.

Tabela 4. Funkcjonalności związane z poradnikami. Źródło: Opracowanie własne.

L.P.	Nazwa	Opis wymagania	Aktorzy	Warunki początkowe
1	Przeglądaj listę poradników	Aktorzy mają możliwość przeglądania listy poradników po wejściu w sekcję poradniki.	Gość, Użytkownik	
2	Oglądaj poradnik	Użytkownik ma możliwość oglądania poradnika po wybraniu poradnika z listy	Gość, Użytkownik	
3	Oceń poradnik	Użytkownik ma możliwość ocenienia obejrzanego	Użytkownik	

		poradnika i zostawienia komentarza		

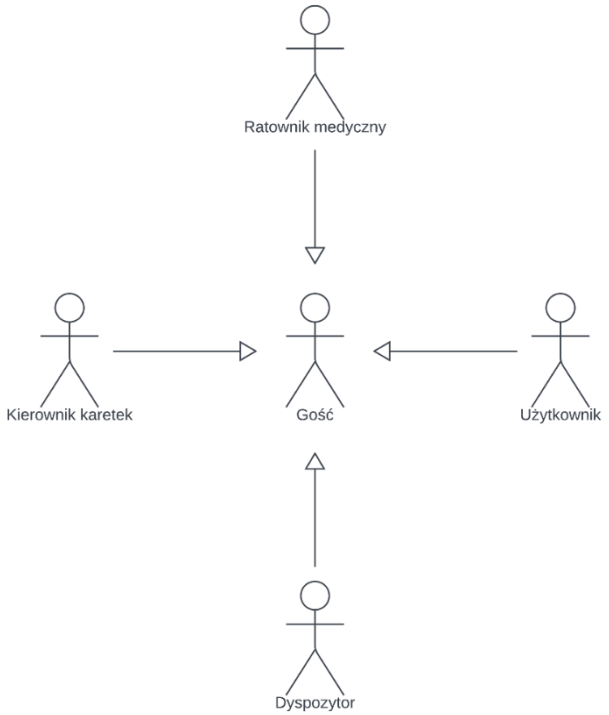
Tabela 5 przedstawia funkcjonalności systemu związane z modyfikacją karetek.

Tabela 5. Funkcjonalności związane z zarządzaniem karetkami. Źródło: Opracowanie własne.

L.P.	Nazwa	Opis wymagania	Aktorzy	Warunki początkowe
1	Przeglądaj listę karetek	Kierownik karetek może przeglądać listę karetek po wejściu w zakładkę “Zarządzaj karetkami”	Kierownik karetek	
2	Modyfikuj karetkę	Kierownik może modyfikować informacje o karetce	Kierownik karetek	Musi być co najmniej jedna karetka
3	Usuń karetkę	Kierownik może usunąć karetkę z systemu	Kierownik karetek	Musi być co najmniej jedna karetka
4	Dodaj karetkę	Kierownik może dodać karetkę do systemu. Wszystkie wymagane informacje muszą zostać podane.	Kierownik karetek	

4.5. Aktorzy systemu

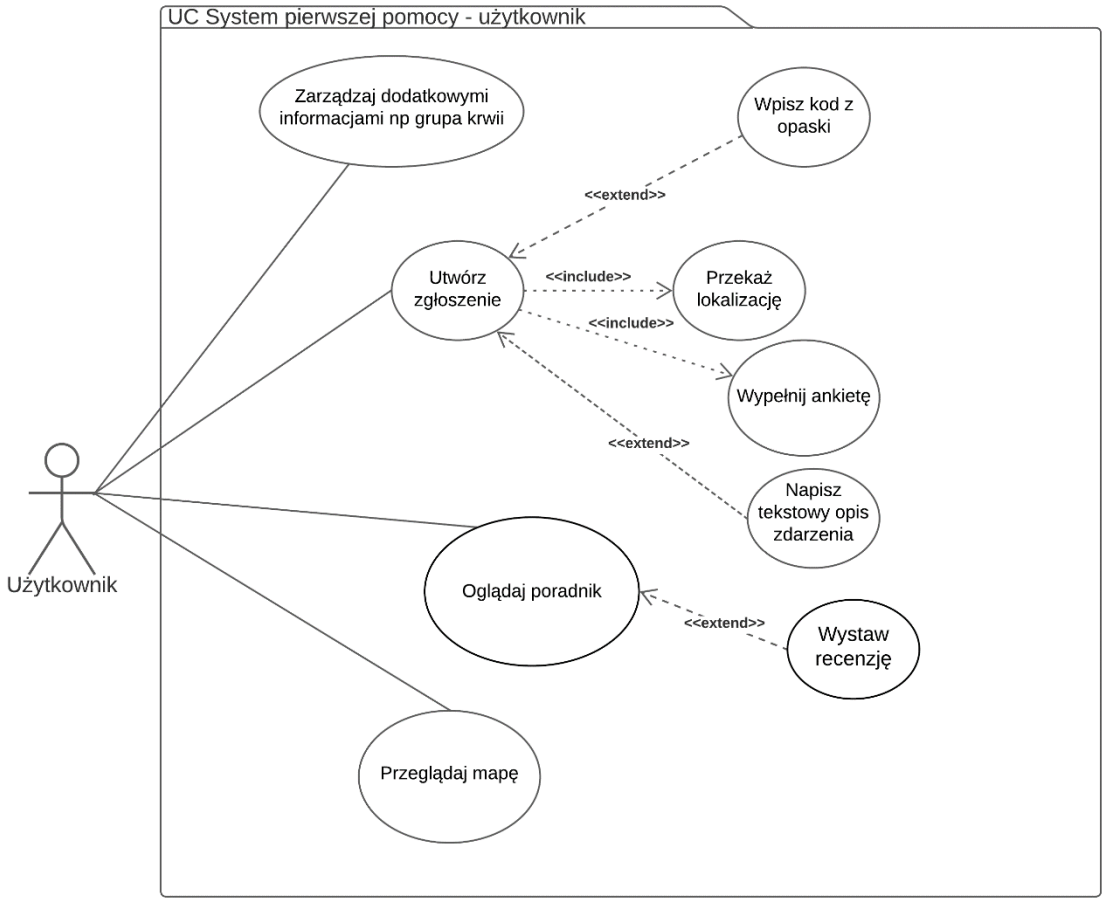
Rysunek 25 przedstawia aktorów w systemie Gary.



Rysunek 25. Diagram aktorów w systemie Gary. Źródło: Opracowanie własne

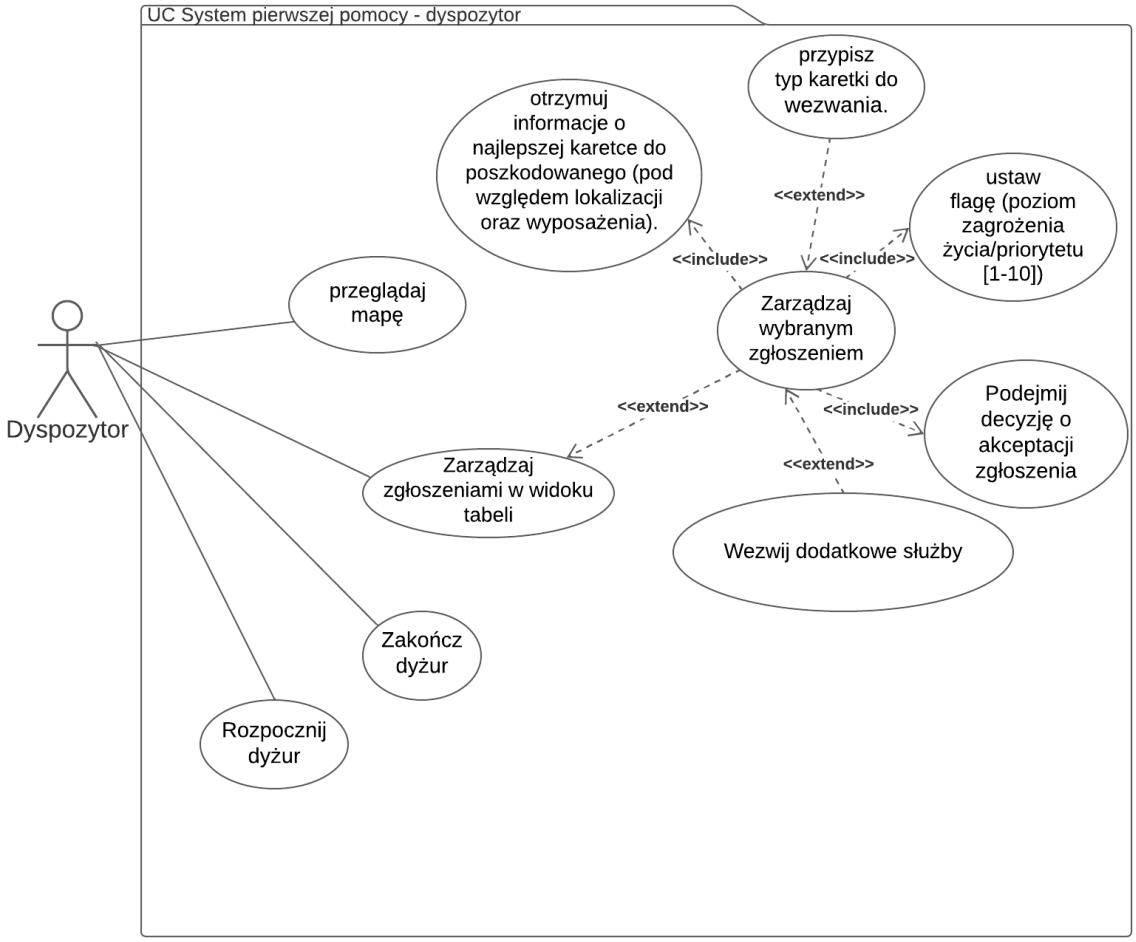
4.6. Przypadki użycia

Rysunek 26 przedstawia diagram przypadków użycia Użytkownika



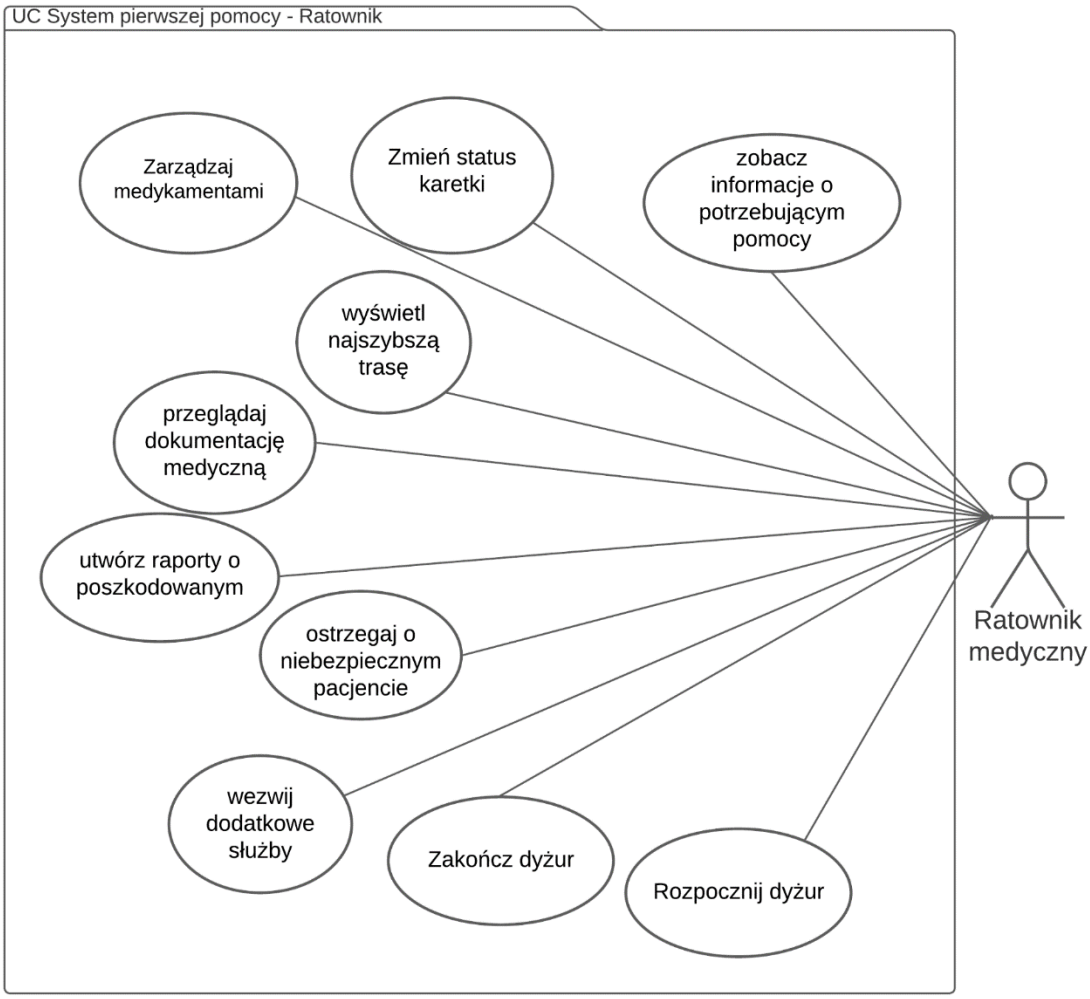
Rysunek 26. Diagram USE CASE użytkownika. Źródło: Opracowanie własne

Rysunek 27 przedstawia diagram przypadków użycia dyspozytora.



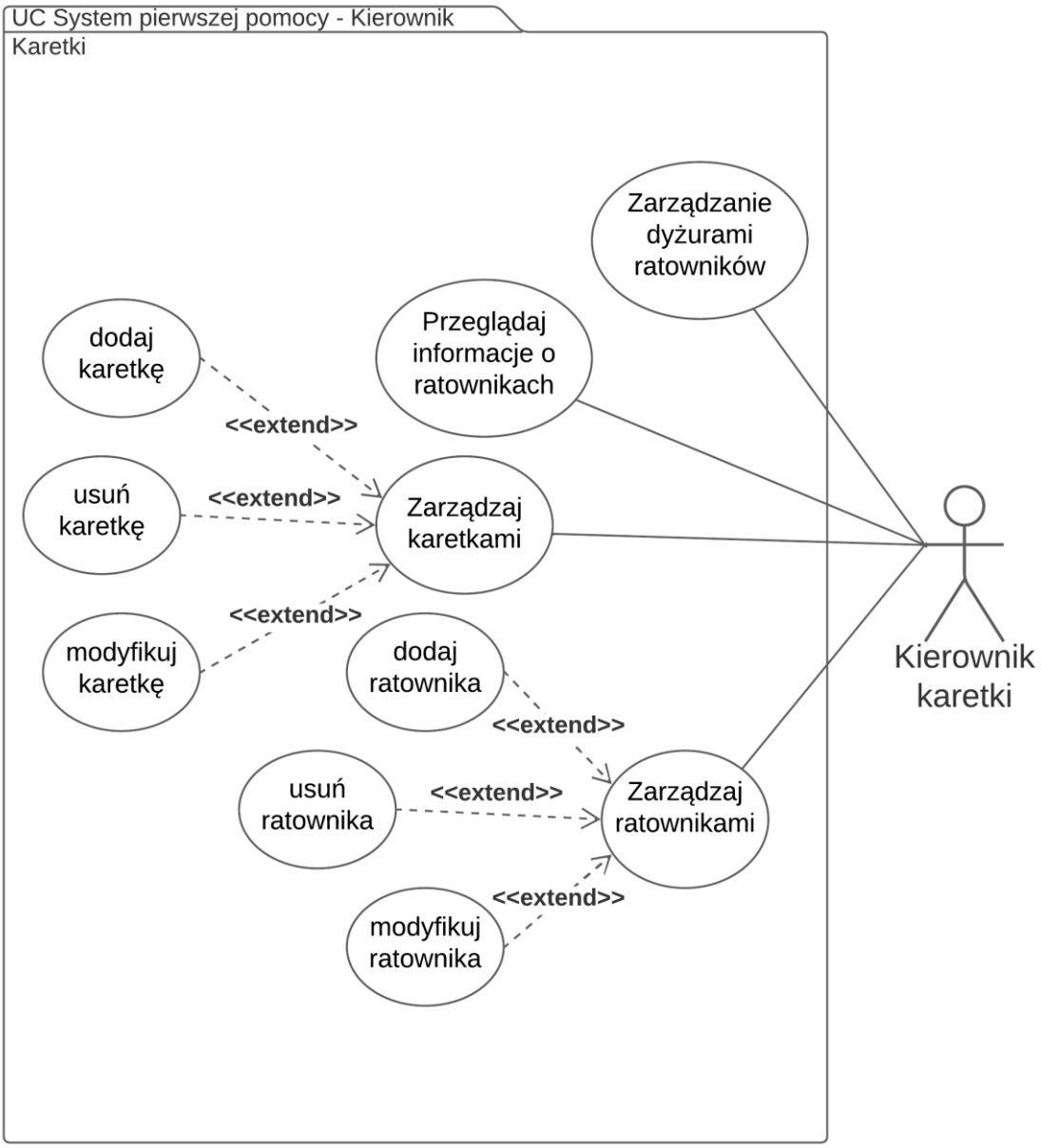
Rysunek 27. Diagram USE CASE dyspozytora. Źródło: Opracowanie własne.

Rysunek 28 przedstawia diagram przypadków użycia ratownika medycznego.



Rysunek 28. Diagram USE CASE ratownika medycznego. Źródło: Opracowanie własne.

Rysunek 29 przedstawia diagram przypadków użycia kierownika karetek.



Rysunek 29. Diagram USE CASE kierownika karetek. Źródło: Opracowanie własne.

4.7. Wymagania niefunkcjonalne

Tabela 6 przedstawia wymagania niefunkcjonalne systemu Gary.

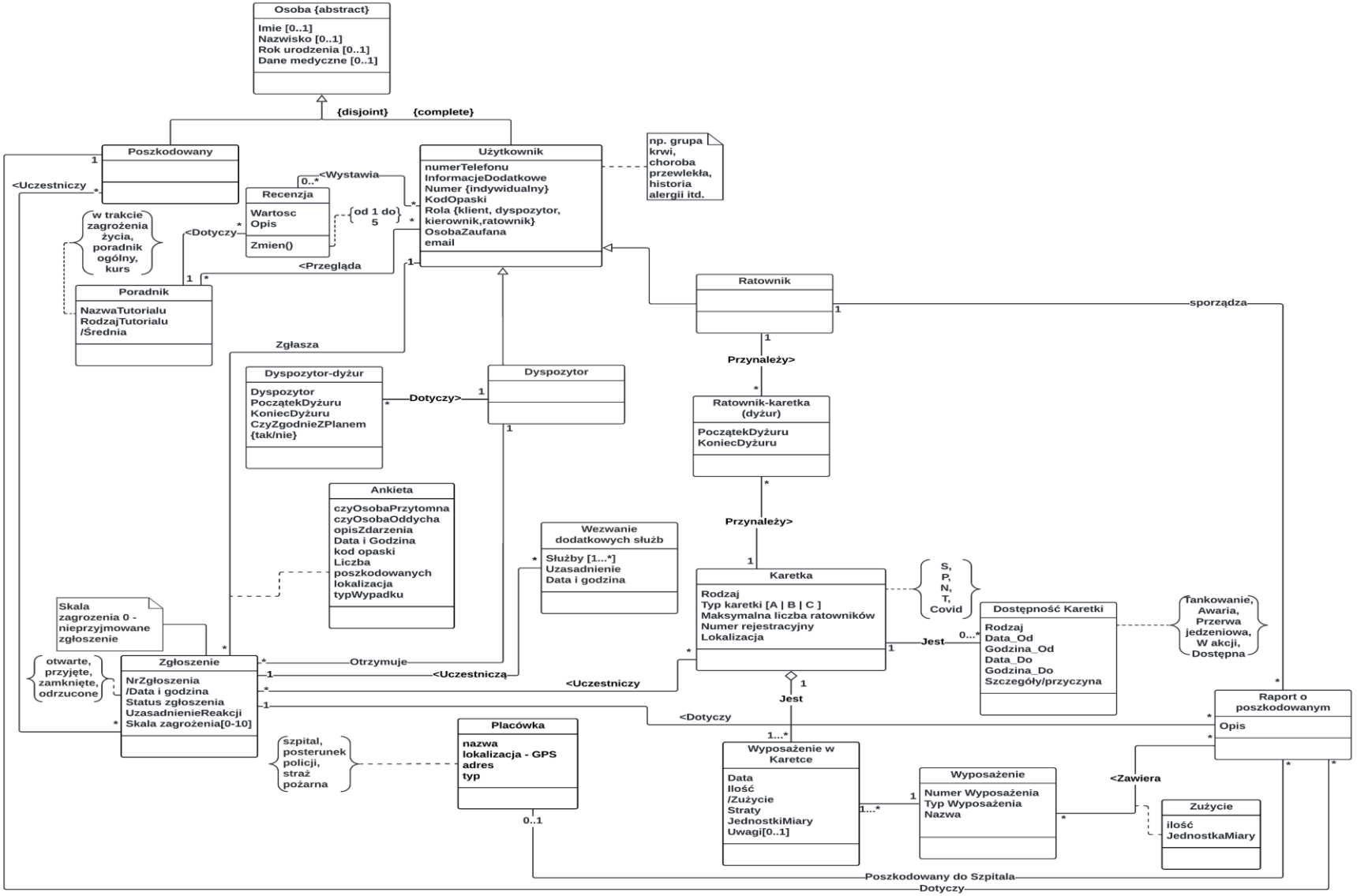
Tabela 6. Wymagania niefunkcjonalne. Źródło: Opracowanie własne

L.P.	Cecha	Priorytet	Opis
1	Użyteczność	Wymagany	System obsługuje język polski i angielski
2	Użyteczność	Wymagany	Odpowiednia przeglądarka internetowa - IE11, Firefox, Safari, Opera, Chrome, Edge

3	Użyteczność	Wymagany	Aplikacja mobilna działa w systemie operacyjnym Android
4	Użyteczność	Wymagany	System działa na systemach operacyjnych Windows i MacOS
5	Wydajność	Wymagany	Serwer przesyła zgłoszenia w czasie poniżej 10 sekund
6	Wydajność	Powinien	Serwer obsługuje 1000 użytkowników jednocześnie.
7	Niezawodność	Powinien	Wymagana pamięć dyskowa 100GB
8	Niezawodność	Powinien	Wymagana ilość RAM 4GB
9	Bezpieczeństwo	Wymagany	Hasła powinny być szyfrowane
10	Bezpieczeństwo	Powinien	Dane medyczne powinny być szyfrowane
11	Wydajność	Umiarkowany	Czas dojazdu karetki do poszkodowanego powinien być aktualizowany na bieżąco
12	Wydajność	Wymagany	Odpowiedź o zajętości szpitala powinna być automatyczna, natychmiastowa.

4.8. Analityczny diagram klas

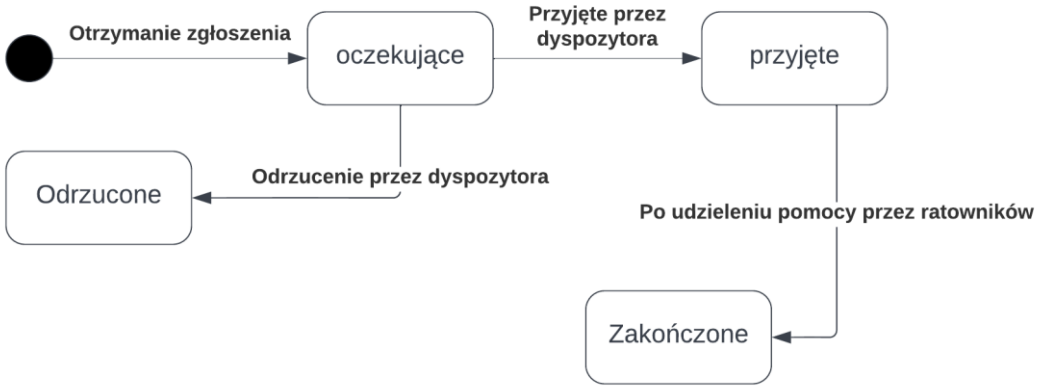
Rysunek numer 30 przedstawia analityczny diagram klas powstały w wyniku przeprowadzonej fazy analizy.



Rysunek 30. Analityczny diagram klas. Źródło: Opracowanie własne

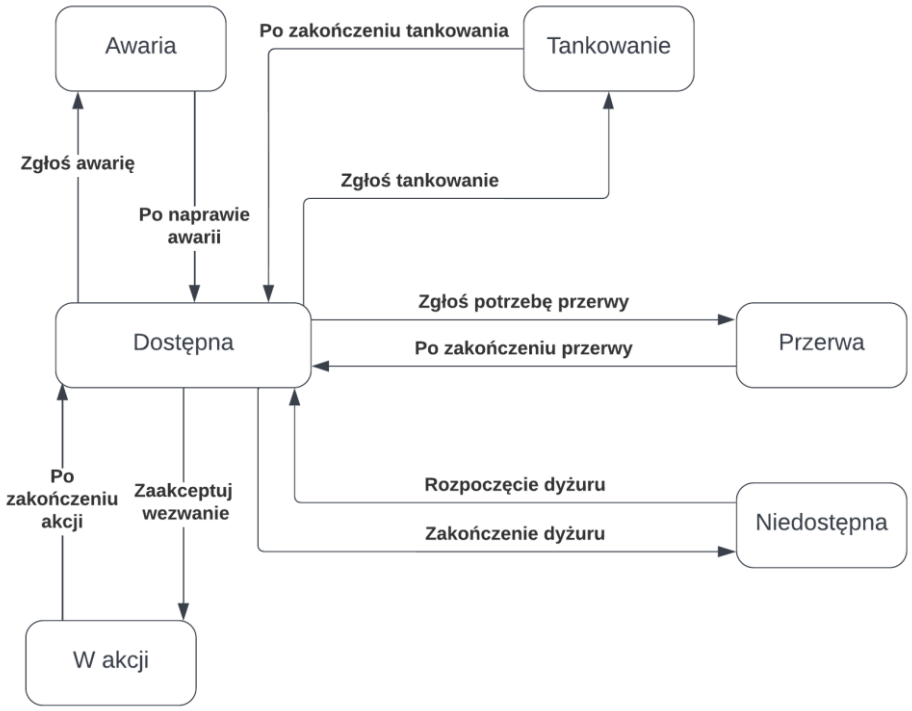
4.9. Diagram stanów

Rysunek 31 przedstawia diagram stanów zgłoszenia. Jak można zauważyć w naszym systemie zgłoszenie po przejściu do kolejnego stanu nie może wrócić do stanów poprzedzających.



Rysunek 31. Diagram stanów dostępności karetki. Źródło: Opracowanie własne

Rysunek 32 przedstawia diagram stanów dostępności karetki. Zależy nam na dokładnej znajomości stanu w czasie, by dyspozytor mógł bez problemu przydzielić dostępną karetkę.



Rysunek 32. Diagram stanów dostępności karetki. Źródło: Opracowanie własne

4.10. Scenariusze

W tym rozdziale można zapoznać się ze scenariuszami dla najważniejszych przypadków użycia powstałych podczas fazy analitycznej.

Tabela 7 przedstawia scenariusz przepływu zdarzeń przypadku „Utwórz zgłoszenie”.

Tabela 7. Scenariusz przepływu zdarzeń przypadku „Utwórz zgłoszenie”

Nazwa	Utwórz zgłoszenie	
Aktorzy:	Użytkownik	
Główny przepływ zdarzeń:		
L.P.	Nazwa aktywności	Numer następnej aktywności:
1	Użytkownik uruchamia aplikację i loguję się.	2
2	Użytkownik wybiera opcję utwórz zgłoszenie	3
3	Użytkownik ustala typ zgłoszenia jako “wypadek”	4
4	Użytkownik podaje liczbę poszkodowanych	5
5	Użytkownik odczytuje lokalizację za pomocą GPS.	6
6	Użytkownik odpowiada na pytania o stan zdrowia poszkodowanych	7
7	Użytkownik skanuje opaskę	8
8	Użytkownik przesyła zgłoszenie	
Alternatywny przebieg zdarzeń:		
L.p.	Nazwa aktywności	Numer następnej aktywności:
3a	Użytkownik ustala typ zgłoszenia jako “niebezpieczne zdarzenie”	3.1a
3.1a	Użytkownik podaje typ niebezpiecznej sytuacji	4
5a	Użytkownik wpisuje adres za pomocą klawiatury	6
5b	Użytkownik podaje lokalizację za pomocą pinezki	6
7a	Użytkownik pozostawia pole na numer opaski puste	8

Tabela 8 przedstawia scenariusz przepływu zdarzenia przypadku „Zarządzaj konkretnym zgłoszeniem”, gdzie warunkiem początkowym jest posiadanego przypisanego do aktora zgłoszenia.

Tabela 8. Scenariusz przepływu zdarzeń przypadku "Zarządzaj konkretnym zgłoszeniem". Źródło: Opracowanie własne

Nazwa	Zarządzaj konkretnym zgłoszeniem	
Aktorzy:	Dyspozytor	
Główny przepływ zdarzeń:		
L.P.	Nazwa aktywności	Numer następnej aktywności:
1	Dyspozytor uruchamia aplikację i loguję się.	2
2	Dyspozytor wybiera zgłoszenie z tabeli	3
3	Dyspozytor zatwierdza zgłoszenie	4
4	Dyspozytor ustawia poziom zagrożenia	5
5	Dyspozytor przypisuje typ karetki do wezwania	6
6	Dyspozytor zapisuje dokonane zmiany	
Alternatywny przebieg zdarzeń:		
L.p.	Nazwa aktywności	Numer następnej aktywności:
3a	Dyspozytor odrzuca zgłoszenie	3.1a
3.1a	Zgłoszenie kończy się	6

Kolejny scenariusz został pokazany w tabeli 9 dla przypadku „Zarządzaj karetkami”.

Tabela 9. Scenariusz przepływu zdarzeń przypadku "Zarządzaj karetkami". Źródło: Opracowanie własne.

Nazwa	Zarządzaj karetkami	
Aktorzy:	Kierownik karetek	
Główny przepływ zdarzeń:		
L.P.	Nazwa aktywności	Numer następnej aktywności:

1	Kierownik karetek uruchamia aplikację i loguję się.	2
2	Kierownik karetek otwiera zakładkę do zarządzania karetkami	3
3	Kierownik wybiera opcję modyfikuj karetkę	4
4	Kierownik pobiera listę karetek	5
5	Kierownik wybiera karetkę z listy do modyfikacji	6
6	Kierownik otwiera formularz do modyfikacji karetki	7
7	Kierownik wpisuje nowe informacje	8
8	Kierownik zapisuje zmiany	
Alternatywny przebieg zdarzeń:		
L.p.	Nazwa aktywności	Numer następnej aktywności:
3a	Kierownik wybiera opcję dodaj nową karetkę	3.1a
3.1a	Kierownik otwiera formularz dodania karetki	3.2a
3.2a	Kierownik wpisuje informacje o nowej karetce	8
3b	Kierownik wybiera opcję usuń karetkę	3.1b
3.1b	Kierownik wybiera karetkę do usunięcia	3.2b
3.2b	Kierownik usuwa karetkę	8

Tabela 10 przedstawia scenariusz przepływu zdarzeń przypadku „Zarządzaj dodatkowymi informacjami”. Aby przyśpieszyć i zapewnić jak najlepszą pomoc dla użytkowników przedstawionego systemu. dodatkowe informacje, będą udostępnione ratownikom udzielającym pomocy danemu użytkownikowi.

Tabela 10. Scenariusz przepływu zdarzeń przypadku "Zarządzaj dodatkowymi informacjami". Źródło: Opracowanie własne.

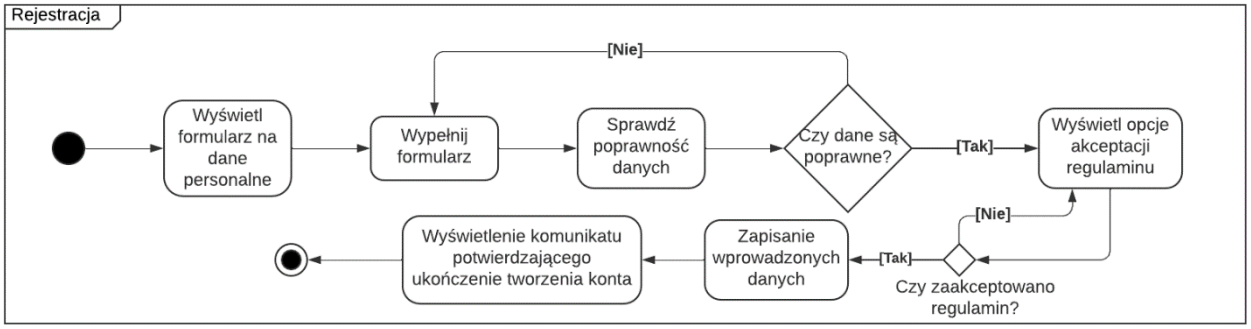
Nazwa	Zarządzaj dodatkowymi informacjami	
Aktorzy:	Użytkownik	
Główny przepływ zdarzeń:		
L.P.	Nazwa aktywności	Numer następnej aktywności:

1	Użytkownik uruchamia aplikację i loguje się.	2
2	Użytkownik otwiera zakładkę ‘Widok użytkownika’	3
3	Użytkownik kilka przycisk ‘Dodaj informacje’	4
4	Użytkownik wybiera grupę krwi jako informację do zmiany	5
5	Użytkownik wybiera daną grupę krwi	6
6	Użytkownik wybiera RH krwi	7
7	Użytkownik klika przycisk ‘Wyślij’	
Alternatywny przebieg zdarzeń:		
L.p.	Nazwa aktywności	Numer następnej aktywności:
2a	Użytkownik otwiera zakładkę ‘Widok użytkownika’	3a
3a	Użytkownik kilka przycisk ‘Dodaj informacje’	4a
4a	Użytkownik wybiera alergię	5
5a	Użytkownik wybiera rodzaj alergii	6a
6a	Użytkownik wpisuje dodatkowe informacje	7
7	Użytkownik kilka przycisk ‘Wyślij’	

4.11. Diagramy aktywności

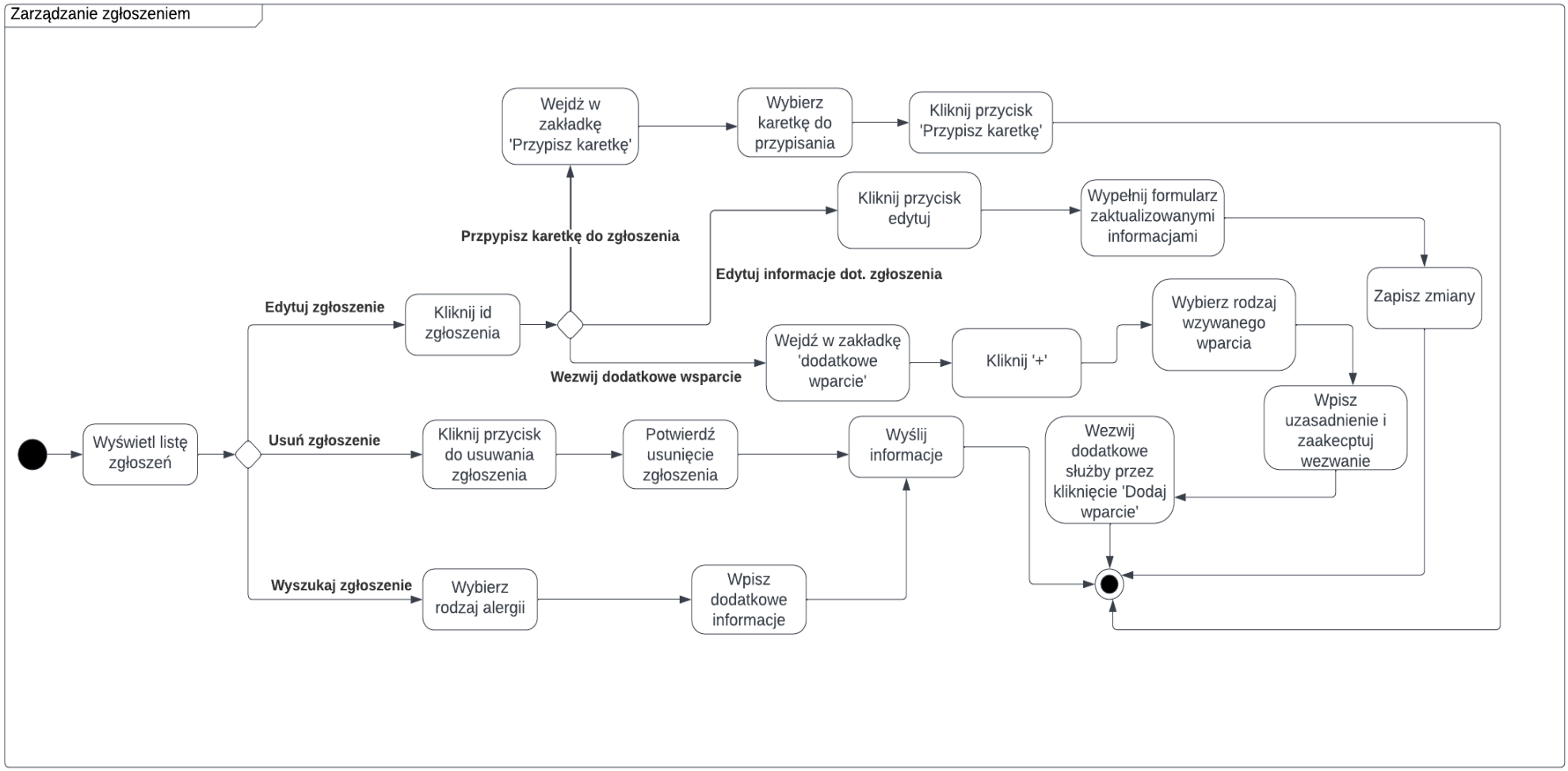
Niniejszy rozdział zawiera kilka przykładowych diagramów aktywności powstałych w wyniku fazy analizy.

Rysunek 33 przedstawia diagram aktywności z rejestracji nowego użytkownika. Przepływ jest identyczny zarówno dla aplikacji webowej, jak i mobilnej.



Rysunek 33. Diagram akrywności dla przypadku "Rejestracja". Źródło: Opracowania własne.

Rysunek 34 przedstawia diagram aktywności dla zarządzania zgłoszenia.



Rysunek 34. Diagram aktywności dla przypadku "Zarządzania zgłoszeniem". Źródło: Opracowanie własne.

5. Technologie i narzędzia wykorzystane w pracy

W trakcie tworzenia projektu zostały wykorzystane różne technologie i narzędzia. Zostały one przedstawione i opisane poniżej.

5.1. Kotlin

Według [10] jest to statycznie typowany język programowania działający na maszynie wirtualnej Javy. Został stworzony przez firmę JetBrains z myślą o pełnej interoperacyjności z językami działającymi na maszynie wirtualnej Javy. Rysunek 35 przedstawia przykładową aplikację napisaną w Kotlinie.

5.1.1. Historia

Nazwa tego języka programowania pochodzi od wyspy niedaleko Petersburga. Został oficjalnie przedstawiony w lipcu 2011 przez JetBrains, jako nowy język na JVM. Według programisty Dmitry Jemerova został on stworzony w celu odwzorowania cech języka Scala w połączeniu z czasem kompilacji na poziomie Javy. W lutym 2012 JetBrains otworzył kod projektu na licencji Apache 2.0.

5.1.2. Filozofia

Według programisty Andrieja Briesława, Kotlin został zaprojektowany jako przemysłowy, obiektowy język w pełni interoperacyjny z kodem napisanym w Javie. Pozwala to firmom na stopniową migrację bazy kodu z Javy do Kotlina oraz zapewnia m.in. eliminację błędów odwołania, funkcje rozszerzeń czy notację infiksową.

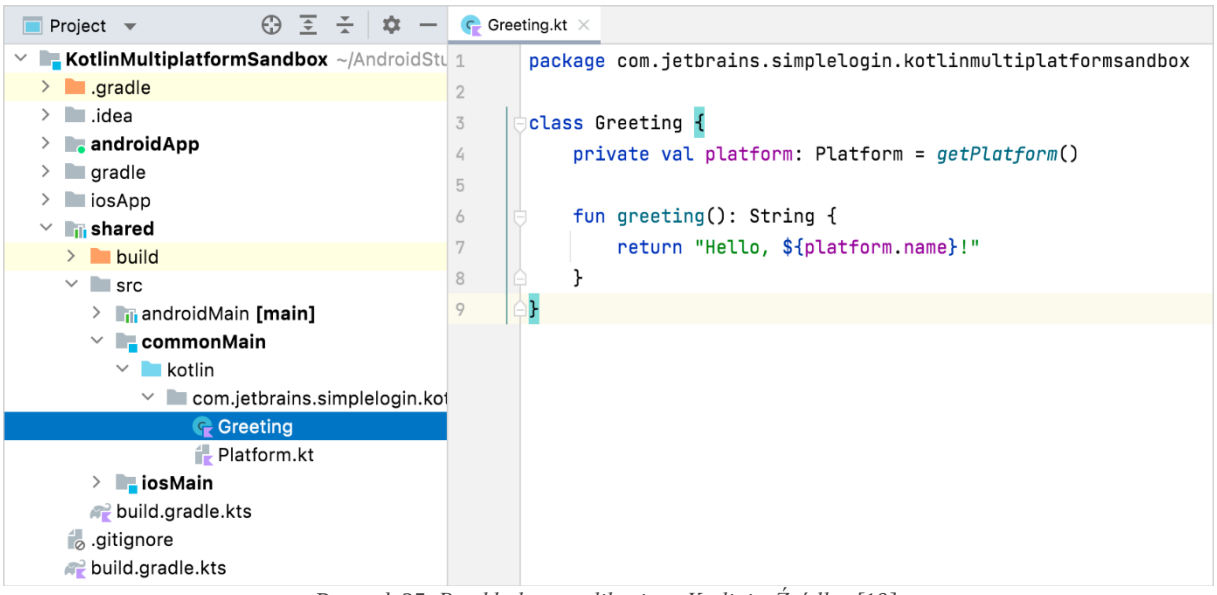
5.1.3. Składnia i semantyka

W deklaracjach zmiennych typ danych umieszczamy po nazwie, uprzednio oddzielając ją dwukropkiem. Średniki na końcu wiersza są opcjonalne.

W Kotlinie oprócz standardowych klas i metod możemy znaleźć elementy programowania proceduralnego – funkcje. Klasycznym punktem wejścia do programu w Kotlinie jest funkcja *main*, do której przekazywana jest tablica z argumentami podanymi w linii poleceń.

5.1.4. Narzędzia

W środowisku IntelliJ IDEA od wersji 15 możemy znaleźć funkcję wspierającą programowanie w Kotlinie. Natomiast w Eclipse znajdziemy odpowiadającą wtyczkę. Ponadto Kotlin współpracuje z takimi narzędziami jak Apache Maven, Apache Ant czy Gradle. Rysunek 31 przedstawia przykładowy projekt w Kotlinie, na którym widoczna jest struktura projektu i fragment kodu.



Rysunek 35. Przykładowa aplikacja w Kotlinie. Źródło: [10].

5.2. Android SDK (Android Software Development Kit)

Według [12] Android SDK jest to zestaw narzędzi dostępnych dla programistów służący do tworzenia aplikacji na platformę Android. W jego skład wchodzą dwie części:

- SDK Tools – wymagane do tworzenia aplikacji niezależnie od wersji Androida
- Platform Tools – narzędzia wyspecyfikowane pod względem konkretnych wersji systemu.

Środowisko programistyczne składa się z m.in.: dokumentacji, przykładowych programów, samouczków, bibliotek, emulatora opartego na QEMU, *debuggera* oraz wielu innych narzędzi.

5.2.1. SDK Tools

SDK Tools to narzędzia wymagane do tworzenia aplikacji na platformę Android. Wśród nich wyróżniamy:

- android – zarządzanie wirtualnymi maszynami (AVD Manager), projektami, oraz instalowanie i odinstalowywanie modułów SDK
- Dalvik Debug Monitor Server (ddms) – *debugger* aplikacji
- emulator – emulator urządzenia z Androidem oparty na QEMU, służący do projektowania, debugowania i testowania aplikacji pod różnymi wersjami systemu
- layoutopt – analiza *layoutu* (rozmieszczenie widżetów) aplikacji w celu zoptymalizowania ich pod kątem wydajności
- mksdcard – tworzenie obrazu dysku do użycia z emulatorem w celu zasymulowania obecności zewnętrznej pamięci
- ProGuard – zmniejszanie, optymalizacja i zaciemnianie kodu poprzez usuwanie nieużywanych fragmentów oraz zmianę nazw klas, metod i pól
- sqlite3 –dostęp do plików baz SQLite tworzonych przez aplikacje
- traceview – graficzna przeglądarka logów wykonania aplikacji
- zipalign – optymalizacja plików APK w taki sposób, by nieskompresowane dane były w konkretnej pozycji względem początku pliku

5.2.2. Platform Tools

Platform tools to narzędzia aktualizowane podczas instalacji każdej nowej platformy SDK. Każda aktualizacja Platform tools jest kompatybilna wstecznie z poprzednią wersją. Android Debug Bridge (*adb*) jest najczęściej używanym narzędziem, które pozwala kontrolować emulator lub urządzenie z Androidem. Używany jest do instalowania aplikacji i uruchamiania ich.

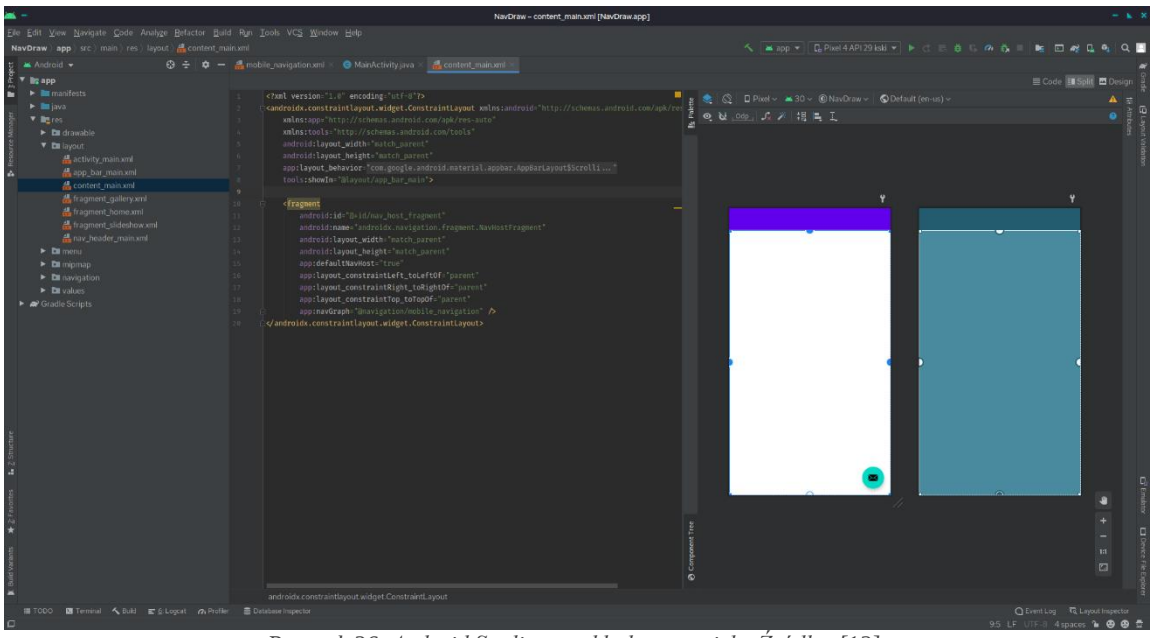
Drugim często używanym narzędziem jest fastboot. Służy on do wykonywania operacji tj. wczytywanie obrazu systemu na urządzenie, zarządzanie partycjami, czy odblokowywanie *bootloadera*.

5.2.3. IDE

W ramach Android SDK udostępniane jest ADT Bundle, czyli zestaw podstawowych narzędzi potrzebnych do rozpoczęcia pisania aplikacji. Wyróżniamy:

- SDK Tools
- Platform Tools
- Zestaw bibliotek do najnowszej wersji Androida
- Obraz systemu do emulatora

W maju 2013 wypuszczono wersję testową Android Studio, czyli środowiska opartego na IntelliJ IDEA. Głównymi jego zaletami w porównaniu do Eclipse'a z ADT jest użycie nowszego Gradle zamiast Anta do budowy oprogramowania, możliwość skonfigurowania kilku wariantów budowy programu dla jednego projektu, a także usprawnienia w kwestiach refaktoryzacji i uzupełniania kodu.
Rysunek 36 przedstawia przykładowy projekt w Android Studio.



Rysunek 36. Android Studio: przykładowa projekt. Źródło: [12].

5.3. Biblioteka Material Design

Jest to system projektowania stworzony przez Google, służący do tworzenia prostego, a równocześnie czytelnego i łatwo dostosowywalnego do różnych płaszczyzn stylu graficznego dla Androida, iOS, Fluttera i aplikacji webowych.

5.3.1. Trójwymiarowa płaszczyzna.

Istotną cechą Material Design jest jego położenie w trójmiarze. Polega ono na wyobrażeniu sobie odpowiedniej hierarchii elementów strony czy aplikacji i oznaczeniu tego za pomocą cieni. W Material Design wyróżnia się trzy podstawowe głębokości, które można stosować do wyróżniania elementów znajdujących się nad innymi.

5.3.2. Żywe kolory

Dużą rolę odgrywa tutaj odpowiedni dobór kolorów. Do stworzenia palet kolorystycznych dla aplikacji i stron internetowych wykorzystuje się dwa główne kolory. Podstawowy stanowi bazę do tworzenia kolejnych jego wariacji i oznaczony jest jako 500. Na jego podstawie powinno się

wygenerować kolory oznaczone od 50 – 900, gdzie te od 50 – 400 są od niego jaśniejsze, a 600 – 900 ciemniejsze.

Drugi kolor powinien zdecydowanie odznaczać się na tle podstawowego i być wykorzystywany do wyróżniania istotnych elementów interaktywnych projektu, takich jak przyciski akcji czy linki.

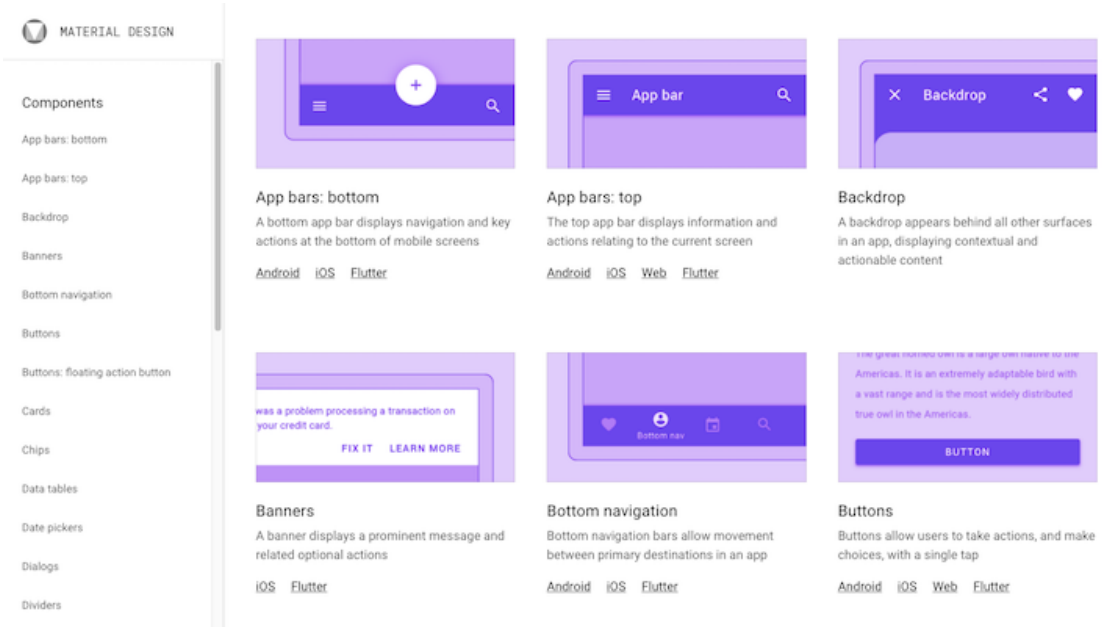
5.3.3. Komponenty

Według [13] Material Components są interaktywnymi blokami konstrukcyjnymi do tworzenia interfejsu użytkownika i zawierają wbudowany system stanów do komunikowania akcji skupienia, zaznaczenia, aktywacji, błędu, najechania kursorem, naciśnięcia, przeciągnięcia i wyłączenia. Biblioteki komponentów są dostępne dla systemów Android, iOS, Flutter i sieci.

Komponenty obejmują szereg potrzeb związanych z interfejsami, wyróżniamy m.in.:

- Wyświetlanie: umieszczanie i organizowanie zawartości za pomocą elementów, takich jak karty, listy i arkusze.
- Nawigacja: Umożliwienie użytkownikom poruszania się po produkcie przy użyciu komponentów, takich jak szuflady nawigacji i karty.
- Akcje: Zezwalanie użytkownikom na wykonywanie zadań przy użyciu komponentów, takich jak pływający przycisk akcji.
- Dane wejściowe: Zezwalanie użytkownikom na wprowadzanie informacji lub dokonywanie wyborów przy użyciu komponentów, takich jak pola tekstowe, elementy i kontrolki wyboru.
- Komunikacja: ostrzeganie użytkowników o kluczowych informacjach i wiadomościach za pomocą takich komponentów, jak snackbar, banery i okna dialogowe.

Rysunek 37 przedstawia przykładowe komponenty z tej biblioteki.



Rysunek 37. Przykładowe komponenty w Material Design. Źródło: [13].

5.3.4. Typografia

Material Design zapewnia 13 stylów typografii, których można użyć do wszystkiego, od nagłówków po tekst podstawowy i podpisy. Każdy styl ma jasne znaczenie i zamierzone zastosowanie w interfejsie.

Ważne atrybuty, takie jak krój pisma, grubość czcionki i wielkość liter, można modyfikować, aby pasowały do konkretnego projektu.

5.3.5. Kształt

Stosowanie różnych stylów kształtów może pomóc skierować uwagę lub zidentyfikować komponenty, zakomunikować ich stan oraz w odpowiedni sposób określić odbiór aplikacji przez użytkownika.

Wszystkie Material Components są pogrupowane w kategorie kształtów na podstawie ich rozmiaru (mały, średni, duży). Te globalne style umożliwiają szybką zmianę kształtu komponentów o podobnych rozmiarach. Jest możliwość generowania własnych styli kształtów za pomocą narzędzia do dostosowywania kształtów.

5.4. Retrofit

Według [41] Retrofit jest klientem HTTP zorientowanym na typowanie obiektów zapytań, używającym do żądań bibliotekę OkHttp. Służy do pobierania i przesyłania danych w formie obiektów lub schemacie JSON za pośrednictwem usługi internetowej opartej na REST. Umożliwia wykonywanie zapytań w sposób synchroniczny i asynchroniczny. Uwzględnia uwierzytelnianie i rejestrowanie stanu operacji.

Zaczynając pracę z tą biblioteką należy dokonać definicji klasy modelu danych, interfejsu deklarującego możliwe operacje HTTP oraz instancji klasy Retrofit.Builder, której zadaniem jest zbudowanie usługi w oparciu o wskazane zależności.

5.4.1. Model

Rolą Konwertera jest próba konwersji otrzymanego wyniku do zadeklarowanego typu klasy modelu o strukturze danych reprezentującej oczekiwany rezultat. Wartości zostają przypisane tylko do właściwości, które są zgodne z formatem odpowiedzi, czyli brakujące lub nadmiarowe pola zostaną zignorowane.

5.4.2. Interfejs

W interfejsie są deklarowane metody, które odwołują się do zasobów sieciowego API. Za pomocą adnotacji @GET, @POST, @PUT, @DELETE można określić rodzaju zapytania dla architektury REST. Zapytania mogą być parametryzowane przy użyciu argumentów metody oznaczonych jako @Path i @Query.

5.4.3. Budowniczy

Możliwość wykorzystania stworzonego API jest uzależniona od zbudowania instancji przy pomocy budowniczego Retrofit Builder. Wymagane jest podanie przynajmniej bazowego URL oraz opcjonalnie m.in. konwertera (np. Gson, Protobuf, Simple XML), adaptera i klienta HTTP. Następnie przy użyciu obiektu Retrofit należy utworzyć instancję wybranego interfejsu API. Przykładowe użycie znajduje się na Listingu 1.

Listing 1. Przykład zastosowania biblioteki Retrofit. Źródło: [14].

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .build();

GitHubService service = retrofit.create(GitHubService.class);
```

5.4.4. Zapytanie

Istnieje możliwość synchronicznego wykonania żądania sieciowego wykorzystywanego w metodzie zapytania typu *Call*. Należy wtedy użyć metody *execute* lub wykonać asynchronicznie metodą *enqueue* wraz z przekazaniem obiektu zwrotnego typu *Callback*.

5.5.5. RxJava

Według [14] Retrofit umożliwia współpracę z RxJava (metody mogą zwracać *Observable*) poprzez dodanie adaptera RxJava2CallAdapterFactory do konfiguracji budowniczego. Sprawia to, że tworzenie aplikacji z wykorzystaniem obu bibliotek staje się prostsze. Dzięki temu Retrofit jest nierzadko wybierany jako podstawowy klient sieciowy w aplikacji używających RxJava.

5.5 Osmdroid

Według [15] jest to biblioteka Androida wykorzystująca odpowiednie narzędzia i widoki umożliwiające użycie Open Source Maps dla aplikacji mobilnych. Jest to pełny i bezpłatny zamiennik klasy MapView (API v1) systemu Android. Oferuje obsługę nakładek z możliwością kreślenia ikon, śledzenia lokalizacji i rysowania kształtów. Jej najnowsza wersja została wypuszczona 22 sierpnia 2022r. (6.1.14). Rysunek 38 przedstawia przykładową mapę stworzoną w oparciu o bibliotekę Osmdroid.

Rysunek 38. Biblioteka Osmdroid: przykładowa mapa. Źródło: [15].

5.6. HTML5

Według [42] jest to język znaczników służący do tworzenia i prezentowania stron internetowych www. Rozwinął się z języka HTML 4 i jego XML-owej odmiany XML-owej odmiany (XHTML 1).

HTML5 precyzuje sposób obsługi błędów w kodzie HTML – taka sama we wszystkich przeglądarkach, nieprawidłowy element będzie działał w każdej przeglądarce albo żadnej.

Możemy zaobserwować również rozwinięcie semantyki. Element `<div>` został wyparty przez `<header>`, `<main>`, `<article>`, `<aside>`, `<footer>`, `<nav>`. Ponadto zostały dodane `<canvas>`, `<figure>`, `<details>`, `<summary>`. Element `<span>` traci na znaczeniu na rzecz `<mark>`, `<output>`, `<var>`, `<u>`, `<s>`.

5.6.1. Różnice w stosunku do HTML4

- Nowe tagi: `section`, `article`, `header`, `footer`, `nav`, `video`, `audio`, `mark`, `progress`, etc.
- Nowe typy pól "input": `tel`, `search`, `url`, `email`, `datetime`, `date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`, `color`.
- Nowe atrybuty elementów formularzy: `autofocus`, `required`, `autocomplete`, `min`, `max`, `multiple`, `pattern`, `step`, etc.
- W poprzednich wersjach HTML, tag `<meta>` nie miał atrybutu `charset`, definiującego standardowe kodowanie znaków dla strony internetowej.
- Możliwość osadzenia MathML i SVG bezpośrednio w dokumencie, podobnie jak w XHTML.

Nowości w API:

- Możliwość 2D z wykorzystaniem `canvas`,
- WebGL (port OpenGL dla przeglądarek),
- API dla odtwarzania audio i video,
- API dla aplikacji offline,
- API, pozwalające zarejestrować aplikację WEB jako protokół lub `media_type`,
- API edycji z atrybutem `contenteditable`,
- API przeciągnij i upuść, z atrybutem `draggable`,
- API do obsługi przycisku wstecz (History API),
- API pamięci pozwalające na przechowywanie danych pomiędzy przeładowaniami strony,
- Microdata – przechowywanie danych w atrybutach (prefix: data-),
- Geolokalizacja,
- Web Sockets (dwukierunkowa komunikacja z serwerem),
- Komunikacja między stronami,
- Nowa wersja XMLHttpRequest umożliwiająca załadowanie plików oraz monitorowanie postępu
- File API – dostęp do systemu plików po stronie klienta
- Baza danych SQL

Nowości w DOM:

- `getElementsByClassName`
- `activeElement, hasFocus`
- `getSelection`
- `classList` (*wrapper* dla className) z metodami: `has(), add(), remove(), toggle()`
- `relList` dla elementu a
- `innerHTML` dla `window` i `document`

5.6.2. Podstawowe elementy używane do tworzenia stron www.

Podstawowymi znacznikami jakie zawsze występują w stronie www stworzonej za pomocą kodu HTML są:

- `<!DOCTYPE>` - określa, że jest to dokument HTMl
- `<html> </html>` - w tych znacznikach znajduje się cała zawartość strony
- `<head> </head>` - w tej sekcji znajdują się informacje dla przeglądarki, które nie są widoczne dla użytkownika, np. przypięty arkusz styli
- `<link>` - znacznik do wstawiania plików np. CSS
- `<title> </title>` - tytuł wyświetlany na karcie przeglądarki
- `<meta>` - ustawia zestaw znaków i umożliwia poprzez kodowanie wyświetlać polskie znaki
- `<body> </body>` - umieszczona jest tutaj cała struktura strony widoczna dla użytkownika

Listing 2 przedstawia jak wygląda przykładowy kod HTML z zachowaniem odpowiedniej struktury.

Listing 2. Przykładowy kod HTML. Źródło: [16].

```
<!doctype html>

<html lang="pl">

<head>

  <meta charset="utf-8">

  <link rel="stylesheet" href="style.css">

  <title>Index – Moja strona</title>

</head>

  <body>

      <h1>Witaj na mojej stronie</h1>

  </body>

</html>
```

5.7. CSS - Cascading Style Sheets

Kaskadowe arkusze stylów służą do formatowania strony napisanej w języku HTML. Według [17] CSS został stworzony przez organizację W3C w 1996r. jako następca języka DSSSL przeznaczonego do używania w połączeniu z SGML-em. Miał on na celu zmniejszenie skompilowania dokumentu HTML. Z tego względu elementy służące do prezentacji strony WWW zostały przeniesione do osobnego pliku – arkusza stylów CSS. W pliku HTML za pomocą znacznika `<link></link>` określamy plik CSS do wyświetlania treści w odpowiedni sposób. Listing 3 przedstawia przykładowe użycie znacznika `<link>`.

Listing 3. Przykładowe użycie znacznika <link>. Źródło: Opracowanie własne.

```
<head>
       <link rel=”stylesheet” href=”style.css”>
</head>
```

Struktura pliku CSS składa się z selektorów i deklaracji. Rolą selektora jest wskazanie elementu np. `<h1>`, zaś deklaracja pozwala opisać ten znacznik poprzez m.in. zmianę koloru czcionki lub wielkości. Każdy selektor można rozróżnić za pomocą klasy, aby móc przyporządkować konkretne właściwości odpowiednim znacznikom. Główną zaletą arkusza stylów jest możliwość określenia cechy dla jednego znacznika na całej stronie lub ujednolicenie cech wszystkich znaczników danego rodzaju w obrębie całego pliku CSS. Listing 4 przedstawia przykładowy kod CSS.

Listing 4. Przykładowy plik CSS. Źródło: Opracowanie własne.

```
h1.welcome{
       color: burlywood;
       font-size: 76%;
}
body{
       background-color: azure;
}
```

Dodatkowo do dyspozycji jest biblioteka CSS – Bootstrap. Zawiera ona wiele gotowych rozwiązań służących do stylizacji tekstów, przycisków, formularzy i innych elementów wyświetlanych na stronie. Bootstrap wykorzystuje język JavaScript.

5.8. Biblioteka React

Według [23] jest to biblioteka języka programowania JavaScript. Stosowany jest do tworzenia interfejsów dla użytkownika. React powstał w 2013 roku w celu uproszczenia i zwiększenia wygody podczas tworzenia stron internetowych. Opiera się na budowie aplikacji przy użyciu komponentów, co pozwala na ponowne wykorzystanie poszczególnych elementów. Poniższy podrozdział przedstawia zastosowane biblioteki w ramach React.

5.8.1. Dodatkowe biblioteki

- `i18n` – jest to biblioteka wykorzystywana do tłumaczenia aplikacji. Pozwala on na zapewnienie kilka różnych wersji językowych. Umożliwia korzystanie z wielu plików z tłumaczeniami i załadowanie ich w dowolnym momencie. Do przechowywania tłumaczeń wykorzystuje się pliki json i są one zapisywane w postaci „klucz” : ”wartość”. Listing 5 przedstawia przykład użycia tej biblioteki.

Listing 5. Przykładowe użycie biblioteki 118n do tłumaczenia. Źródło: [18]

```
<script>
    let langResourcesArr = {
        "en": {
            "hello" : "Hello!",
            "welcome": "Welcome to my app!"
        },
        "fr": {
            "hello": "Bonjour!",
            "welcome": "Bienvenue sur mon appli!"
        },
        "it": {
            "hello": "Ciao!",
            "welcome": "Benvenuto nella mia app!"
        }
    };
</script>
```

Przykładowo możemy skorzystać z tłumaczeń angielsko-polskich, jeśli zostanie skonfigurowany plik do zarządzania tłumaczeniem. Następuje to w sposób przypisania tłumaczeń w języku angielskim (`resourcesEN`) do angielskiego skrótu „en”, analogicznie dla języka polskiego. Przykładowe użycie na Listingu 6.

Listing 6. Przykładowa konfiguracja pliku do tłumaczenia. Źródło: Opracowanie własne.

```
import resourcesPL from ”./locales/resources.pl.json”;
import resourcesEN from ”./locales/resources.en.json”;
const resources = {
     pl: {
          translation: resourcesPL
     },
     en: {
          translation: resourcesEN
     };
```

- `leaflet` – służy do obsługi map, np. OpenStreetMap. Pozwala na stylizowanie i tworzenie interaktywnych warstw, a także korzystanie z danych w standardzie GeoJson. Przykładowe użycie przedstawione na Listingu 7.

Listing 7. Przykładowe użycie biblioteki leaflet. Źródło: [19].

```
<script>
  // initialize Leaflet
  var map = L.map('map').setView({lon: 0, lat: 0}, 2);

  // add the OpenStreetMap tiles

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
{
    maxZoom: 19,
    attribution:                    '&copy;                    <a
href="https://openstreetmap.org/copyright">OpenStreetMap
contributors</a>'
  }).addTo(map);

  // show the scale bar on the lower left corner
  L.control.scale({imperial: true, metric: true}).addTo(map);

  // show a marker on the map
  L.marker({lon:  0,  lat:  0}).bindPopup('The  center  of  the
world').addTo(map);
</script>
```

- `recharts` – biblioteka służąca do tworzenia wykresów.
- `axios` - jest wykorzystywany jako klient HTTP, dzięki któremu można stworzyć zapytania w przeglądarce. Zwraca nam status pozytywny lub negatywny dla danego zapytania. Pozwala wysyłać zapytania typu Get i Post.
- `react-icons` - umożliwia dodawanie ikon do projektu. Pozwala je stylizować w różny sposób tak, aby były spójne z aplikacją. Importuje tylko te ikony, które są wykorzystywane w projekcie.

- `react-rating` - służy do dodania komponentu służącego do oceniania na stronie web, korzysta z niestandardowych wbudowanych styli lub też z narzędzi takich jak Bootstrap.

5.9. Java

Według [20] jest to wysokopoziomowy obiektowy język programowania stworzony przed firmę Sun Microsystems, a obecnie rozwijany przez Oracle. Od wielu lat jest to jeden z najpopularniejszych języków programowania. Jest niezawodna, bezpieczna i niezależna od architektury. Program napisany w Javie jest kompilowany do kodu bajtowego, który następnie jest wykonywany przez maszynę wirtualną. Obecnie najnowszą wersją Javy jest Java 19.

5.9.1. Główne koncepcje Javy

- Obiektowość – jest silnie ukierunkowana na programowanie obiektowe, wyjątkiem są typy proste takie jak `int` czy `float`
- Dziedziczenie – wszystkie obiekty dziedziczą po klasie `Object`. Dzięki temu mają wspólne podstawowe możliwości, takie jak ich: identyfikacja, porównywanie, kopiowanie, niszczenie czy wsparcie dla programowania współbieżnego
- Niezależność od architektury - kod źródłowy programów pisanych w Javie kompiluje się do kodu pośredniego. Dzięki temu jest on niezależny od systemu operacyjnego i procesora. Wykonywany jest poprzez wirtualną maszynę Javy, która tłumaczy kod uniwersalny na kod dostosowany do specyfiki konkretnego systemu operacyjnego i procesora.
- Sieciowość i obsługa programowania rozproszonego – Java posiada wyspecjalizowane funkcje umożliwiające programowanie rozproszone. Dodatkowo jest możliwość tworzenia apletów oraz serwletów.

Listing 8 przedstawia fragment kodu napisany w języku Java.

Listing 8. Przykładowy kod Java. Źródło: Opracowanie własne.

```
public class Main {
  public static void main(String[] args){
    System.out.print("Hello World");
  }
}
```

5.9.2. Java Development Kit (JDK)

Według [35] JDK to środowisko programistyczne służące do tworzenia apletów i aplikacji Java. Programiści Java mogą go używać w systemach Windows, macOS, Solaris i Linux. JDK pomaga im kodować i uruchamiać programy Java. Istnieje możliwość zainstalowania więcej niż jednej wersji JDK na tym samym komputerze.

JDK zawiera narzędzia wymagane do pisania programów Java i JRE do ich wykonywania. Zawiera kompilator, program uruchamiający aplikacje Java, *Appletviewer* itp. Kompilator konwertuje kod napisany w Javie na kod bajtowy. Program uruchamiający aplikacje Java otwiera środowisko JRE, ładuje niezbędną klasę i wykonuje swoją główną metodę.

5.9.3. Java Virtual Machine (JVM)

Wirtualna maszyna Javy (JVM) to silnik, który zapewnia środowisko wykonawcze do sterowania kodem Java lub aplikacjami. Konwertuje kod bajtowy Java na język maszynowy. JVM jest częścią środowiska Java Runtime Environment (JRE). W innych językach programowania kompilator tworzy kod maszynowy dla określonego systemu. W tym przypadku kompilator Java tworzy kod dla maszyny wirtualnej znanej jako wirtualna maszyna Java.

JVM zapewnia niezależny od platformy sposób wykonywania kodu źródłowego Java. Posiada liczne biblioteki, narzędzia i biblioteki. Po uruchomieniu programu Java możesz go uruchomić na dowolnej platformie i zaoszczędzić mnóstwo czasu. JVM jest dostarczany z kompilatorem JIT (Just-in-

Time), który konwertuje kod źródłowy Java na język maszynowy niskiego poziomu. Dlatego działa szybciej niż zwykła aplikacja.

5.9.4. Java Runtime Environment (JRE)

JRE to oprogramowanie przeznaczone do uruchamiania innego oprogramowania. Zawiera m.in. biblioteki klas i maszynę JVM. Mówiąc prościej, jeśli chcesz uruchomić program Java, potrzebujesz środowiska JRE. Jeśli nie jesteś programistą, nie musisz instalować JDK, wystarczy JRE, aby uruchamiać programy Java.

Środowisko JRE zawiera biblioteki klas, maszynę JVM i inne pliki pomocnicze. Nie zawiera żadnego narzędzia do programowania w języku Java, takiego jak debugger, kompilator itp. Wykorzystuje ważne klasy pakietów, takie jak biblioteki matematyczne, `swing`, `util`, `lang`, `awt` i `runtime`.

5.10. Spring

Jest jednym z najpopularniejszych bibliotek oraz kontenerem bazującym na zasadach odwrócenia sterowania (*Inversion-of-Control*) dla aplikacji Java według [21]. Często jest używany jako dodatek do modelu Enterprise JavaBeans (EJB). Framework Spring jest projektem *open source*. Listing 9 przedstawia przykładowy kod napisany z użyciem Springa.

Listing 9. Przykładowy kod z użyciem biblioteki Spring. Źródło: [21].

```
package com.example.springboot;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

	@GetMapping("/")
	public String index() {
		return "Greetings from Spring Boot!";
	}
}
```

5.10.1. Jak działa biblioteka Spring?

Według [36] aplikacja internetowa (architektura warstwowa) zwykle obejmuje trzy warstwy:

- Warstwa prezentacji/widoku (UI) — jest to najbardziej zewnętrzna warstwa, która obsługuje prezentację treści i interakcję z użytkownikiem.
- Warstwa logiki biznesowej — centralna warstwa zajmująca się logiką programu.
- Warstwa dostępu do danych — warstwa głęboka, która zajmuje się pobieraniem danych ze źródeł.

Każda warstwa jest zależna od drugiej oraz niezbędna dla poprawnego działania aplikacji. Innymi słowy, warstwa prezentacji komunikuje się z warstwą logiki biznesowej, która komunikuje się z warstwą dostępu do danych. Zależność jest tym, czego każda warstwa potrzebuje do wykonywania swojej funkcji. Typowa aplikacja ma tysiące klas i wiele zależności.

Bez biblioteki Spring kod aplikacji jest zwykle ściśle powiązany (współzależny), co nie jest uważane za dobrą praktykę kodowania. Luźne sprzężenie jest idealne, ponieważ luźno połączone elementy są niezależne, co oznacza, że zmiany w jednym nie będą miały wpływu na działanie innych.

Podstawową logiką Springa jest wstrzykiwanie zależności. Wstrzykiwanie zależności to wzorzec programistyczny, który umożliwia programistom tworzenie bardziej oddzielonych architektur. Wstrzykiwanie zależności oznacza, że Spring rozumie różne adnotacje Java, które programista

umieszcza na klasach. Spring wie, że programista chce stworzyć instancję klasy i że powinien nią zarządzać. Biblioteka ta również analizuje zależności i upewnia się, że wszystkie utworzone instancje mają odpowiednio je wypełnione.

Aby Spring tworzył instancje obiektów i wypełniał zależności, programista po prostu określa, którymi obiektami ma zarządzać i jakie są zależności dla każdej klasy. Deweloper robi to za pomocą adnotacji, takich jak:

- `@component` - pozwala Springowi wiedzieć, którymi klasami zarządzać (tworzyć). Oznacza komponenty bean (obiekty) jako zarządzane, co oznacza, że Spring automatycznie wykryje te klasy w celu wstrzyknięcia zależności.
- `@autowired` - mówi Springowi, jak obsługiwać instancję klasy (więc zaczyna szukać tej zależności między komponentami/klasami, aby znaleźć dopasowanie).

5.10.2. Ważna terminologia Spring

Autowiring — proces, w którym Spring identyfikuje zależności, dopasowuje je i zapełnia.

Bean — bean to obiekt, który jest tworzony i zarządzany przez kontener IoC. Jest on podstawą aplikacji.

Wstrzykiwanie zależności (*Dependency injection*) — wzorzec projektowy, który sprawia, że kod jest luźno powiązany, co oznacza, że jakakolwiek zmiana w zastosowaniu jednego nie wpłynie na drugi.

Inversion of control (IoC) — odebranie kontroli klasie i przekazanie jej bibliotece Spring.

Odwrócenie kontenera kontrolnego (*Inversion of control container*) — jest to rdzeń Spring Framework, w którym obiekty są tworzone, łączone ze sobą, konfigurowane i zarządzane przez cały cykl ich życia.

5.11. Hibernate

Według [23] jest to biblioteka wspomagająca dostęp do danych w Javie. Głównym celem jej użycia jest translacja danych pomiędzy relacyjną bazą danych a światem obiektowym. Opisuje strukturę danych za pomocą adnotacji, a dzięki temu umożliwia rzutowanie obiektów w obiektowych językach programowania takich jak Java. Na Listingu 10 znajduje się przykładowy test konfiguracji adnotacji Hibernate.

Listing 10. Przykładowy test konfiguracji adnotacji – Hibernate. Źródło: [22].

```
package com.journaldev.hibernate.main;

import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.journaldev.hibernate.model.Employee1;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateAnnotationMain {

     public static void main(String[] args) {
          Employee1 emp = new Employee1();
          emp.setName("David");
          emp.setRole("Developer");
          emp.setInsertTime(new Date());

          //Get Session
```

```
            SessionFactory sessionFactory =
HibernateUtil.getSessionAnnotationFactory();
            Session session = sessionFactory.getCurrentSession();
            //start transaction
            session.beginTransaction();
            //Save the Model object
            session.save(emp);
            //Commit transaction
            session.getTransaction().commit();
            System.out.println("Employee ID="+emp.getId());

            //terminate session factory, otherwise program won't end
            sessionFactory.close();
      }
}
```

5.11.1. Interfejs aplikacji

Interfejs org.hibernate.Session reprezentuje sesję Hibernate, czyli główny punkt manipulacji przeprowadzanej na obiektach bazy danych. Te ostatnie działania obejmują (między innymi) zarządzanie stanem trwałości obiektów, pobieranie utrwalonych obiektów z bazy danych oraz zarządzanie rozgraniczeniem transakcji.

Sesja ma trwać tak długo, jak logiczna transakcja w bazie danych. Ze względu na tę ostatnią funkcję implementacje sesji nie powinny traktowane jako bezpieczne dla wątków ani używane przez wielu klientów.

5.11.2. Mapowanie

Mapowanie klas Java na tabele bazy danych jest realizowane przez konfigurację pliku XML lub przy użyciu adnotacji Java. Używając pliku XML, Hibernate może generować szkielet kodu źródłowego dla klas trwałości. Jest to dużym udogodnieniem, gdy używane są adnotacje. Hibernate może używać pliku XML lub adnotacji Java do utrzymania schematu bazy danych.

Dostępne są narzędzia do przedstawienia relacji jeden-do-wielu i wiele-do-wielu między klasami. Oprócz zarządzania powiązaniami między obiektami, Hibernate może również zarządzać powiązaniami zwrotnymi, w których obiekt ma relację jeden-do-wielu z innymi instancjami klasowymi.

Hibernate obsługuje mapowanie niestandardowych typów wartości. Dzięki temu możliwe są następujące scenariusze:

- Zastępowanie domyślnego typu SQL podczas mapowania kolumny na właściwość,
- Mapowanie wyliczeń Java na kolumny tak, jakby były zwykłymi właściwościami,
- Mapowanie jednej właściwości na wiele kolumn.

Mapowanie informuje narzędzie ORM o tym, jaki obiekt klasy Java ma być przechowywany w której tabeli bazy danych.

5.11.2. Podprojekty

- Hibernate Core – czyli centralna część wszystkich projektów i umożliwia wykonywanie mapowania obiektowo-relacyjnego dla języka Java, w którym mapowania zdefiniowane są w dokumentach XML.
- Hibernate Annotations – rozszerzenie projektu, które umożliwia stosowanie adnotacji.
- Hibernate EntityManager – nakładka wprowadzająca EntityManager jako centralną klasę, z poziomu której wykonywana jest komunikacja z bazą danych.
- Hibernate Shards – ułatwia używanie Hibernate Core w przypadku dużej ilości baz danych.
- Hibernate Validator – rozszerza adnotacje o takie, które umożliwiają dodanie ograniczeń na wartości pól podlegających mapowaniu.
- Hibernate Search – rozszerza Hibernate Core o usługę pełnotekstowego wyszukiwania opierając się o bibliotekę Lucene.
- Hibernate Tools – jest to zestaw narzędzi ułatwiających wykorzystanie Hibernate Core w tworzeniu projektów w języku Java.

- NHibernate – implementuje usługę mapowania obiektowo-relacyjnego dla platformy .NET.

5.12. Lombok

Według [44] jest to biblioteka Javy, której głównym założeniem jest ułatwienie definiowania klas, szczególnie klas modelu zgodnych ze standardem JavaBeans lub klas niezmiennych. Lombok przede wszystkim poprzez generowanie kodu umożliwia ograniczenie konieczności pisania powtarzalnego kodu takiego jak np. gettery, settery, konstruktory, czy metody takie jak `hashCode()`, czy `equals(Object obj)`. W kodzie dodawana jest odpowiednia adnotacja, która sprawia, że jest on mniej obszerny i czytelniejszy. Listing 11 przedstawia fragment kodu z wykorzystaniem biblioteki Lombok.

Listing 11. Przykładowy kod z użyciem biblioteki Lombok. Źródło: [24].

```
@Entity
@Getter @Setter @NoArgsConstructor // <--- THIS is it
public class User implements Serializable {

    private @Id Long id; // will be set when persisting

    private String firstName;
    private String lastName;
    private int age;

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
}
```

5.12.1. Najważniejsze adnotacje Lombok

- **@Getter, @Setter przedstawione na Listingu 12**
  Według [37] kiedy pole jest opatrzone adnotacją `@Getter` i/lub `@Setter`, Lombok automatycznie wygeneruje odpowiednio domyślny getter i/lub setter. Domyślna implementacja getterów zajmuje się zwracaniem pola z adnotacją. Podobnie domyślna implementacja setterów przyjmuje jeden parametr tego samego typu co pole z adnotacjami i ustawia go z odebraną wartością. Kiedy pole o nazwie `value` ma adnotację zarówno `@Getter`, jak i `@Setter`, Lombok zdefiniuje metodę `getValue()` (lub `isValue()`, jeśli pole jest wartością logiczną) oraz metodę `setValue()`. Wygenerowana metoda pobierająca/ustawiająca będzie publiczna, chyba że zostanie określony określony poziom dostępu. Dozwolone wartości *AccessLevel* to PUBLIC, PROTECTED, PACKAGE i PRIVATE.

  Listing 12. Przykładowy getter i setter z użyciem biblioteki Lombok. Źródło: [37].

```
@Getter
@Setter
public class Author {
    private int id;
    private String name;
    @Setter(AccessLevel.PROTECTED)
    private String surname;
}
```

- **`@NoArgsConstructor`, `@RequiredArgsConstructor`, `@AllArgsConstructor` przedstawiona na Listingu 13**
  Gdy klasa jest opatrzona adnotacją `@NoArgsConstructor`, Lombok zajmie się automatycznym wygenerowaniem konstruktora bez parametrów. Podobnie, gdy zostanie opatrzony adnotacją `@AllArgsConstructor`, zostanie wygenerowany konstruktor z jednym parametrem dla każdego pola w klasie. `@RequiredArgsConstructor` prowadzi do konstruktora z parametrem dla każdego pola wymagającego specjalnej obsługi. W szczególności dotyczy to niezainicjowanych pól końcowych, a także wszelkich pól oznaczonych jako `@NonNull`, które nie są inicjowane w miejscu zadeklarowania. Adnotacje te ignorują pola statyczne.

Listing 13. Adnotacja @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor. Źródło: [37].

```
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
public class Author {
    private int id;
    private String name;
    private String surname;
    private final String birthPlace;
}
```

- **`@ToString` przedstawiona na Listingu 14**
  Kiedy klasa jest opatrzona adnotacją `@ToString`, Lombok zadba o wygenerowanie poprawnej implementacji metody `toString()`. Domyślnie zwrócony zostanie ciąg zawierający nazwę klasy, po której następuje wartość każdego pola oddzielona przecinkiem. Ustawienie parametru `includeFieldNames` na wartość `true` spowoduje umieszczenie nazwy każdego pola przed jego wartością. Domyślnie podczas generowania metody `toString()` zostaną uwzględnione wszystkie pola niestatyczne. Adnotowanie pola za pomocą `@ToString.Exclude`, sprawi, że Lombok je zignoruje.

Listing 14. Adnotacja @ToString. Źródło: [37].

```
@ToString(includeFieldNames=true)
public class Author {
    private int id;
    private String name;
    private String surname;
}
```

- **`@EqualsAndHashCode` przedstawiona na Listingu 15**
  Adnotacja klasy za pomocą `@EqualsAndHashCode` sprawi, że Lombok automatycznie zaimplementuje metody `equals()` i `hashCode()`. Domyślnie uwzględniane będą wszystkie niestatyczne pola. Można modyfikować, które pola są używane, dodając do nich adnotacje `@EqualsAndHashCode.Include` lub `@EqualsAndHashCode.Exclude`. Alternatywnie można dodać adnotację do klasy za pomocą `@EqualsAndHashCode(onlyExplicitlyIncluded = true)`, a następnie dokładnie określić, które pola lub metody mają być używane, oznaczając je za pomocą `@EqualsAndHashCode.Include`. Należy pamiętać, że metody `equals())` i `hashCode())` będą generowane przez Lombok bez zerwania umowy między nimi.

Listing 15. Adnotacja @EqualsAndHashCode. Źródło: [37].

```
@Getter
@Setter
@EqualsAndHashCode
public class Author {
    private int id;
    private String name;
    private String surname;
}
```

- **`@NonNull` przedstawiona na Listingu 16**
  Za pomocą `@NonNull` można dodać adnotację do komponentu rekordu, parametru metody lub całego konstruktora. W ten sposób Lombok wygeneruje odpowiednie zestawienia kontroli zerowej.

Listing 16. Adnotacja @NonNull. Źródło: [37].

```
public class Author {
    private int id;
    private String name;
    private String surname;

    public Author(
      @NonNull int id,
      @NonNull String name,
      String surname
    ) {
      this.id = id;
      this.name = name;
      this.surname = surname;
  }
}
```

- **`@Data` przedstawiona na Listingu 17**
  `@Data` to adnotacja skrótu, która łączy funkcje `@ToString`, `@EqualsAndHashCode`, `@Getter` `@Setter` i `@RequiredArgsConstructor` razem.

Listing 17. Adnotacja @Data. Źródło: [37].

```
@Data
public class Author {
    private final int id;
    private String name;
    private String surname;
}
```

- **`@Value` przedstawiona na Listingu 18**
  `@Value` to niezmienny wariant `@Data`. Oznacza to, że wszystkie pola są domyślnie ustawione jako prywatne i finalne przez Lombok. Ponadto setery nie zostaną wygenerowane, a sama klasa zostanie oznaczona jako finalna. W ten sposób klasa nie będzie dziedziczona. Podobnie jak w przypadku `@Data`, tworzone są również implementacje `toString()`, `equals()` i `hashCode()`.

Listing 18. Adnotacja @Value. Źródło: [37].

```
@Data
```

```
public class Author {
    int id;
    String name;
    String surname;
}
```

5.12.2. Wtyczka Lombok

Najpopularniejsze i najczęściej używane IDE są dostarczane z oficjalną wtyczką Lombok zaprojektowaną, aby pomóc programistom w korzystaniu z Lombok. W szczególności wspiera, oferując skróty do najpopularniejszych adnotacji Lombok. Ponadto sugeruje adnotacje, które są potrzebne, w zależności od tego, w jakim miejscu w kodzie się znajduje. Oficjalnie obsługiwane są IntelliJ IDEA, Eclipse, Spring Tool Suite, (Red Hat) JBoss Developer Studio, MyEclipse, Microsoft Visual Studio Code i Netbeans.

5.13. PostgreSQL

Jest jednym z otwartych systemów zarządzania relacyjną bazą danych zgodnym ze standardem SQL według [25]. Zalicza się do baz typu RDBMS z rozszerzeniami obiektowymi. Obsługuje zarówno zapytania SQL (relacyjne), jak i JSON (nierelacyjne). Listing 19 przedstawia przykład użycia komendy *update* w PostgreSQL.

Listing 19. Zastosowanie PostgreSQL. Źródło: [26].

```
postgres=#   update   dummy_table   set   name='GHI',age=54   where
address='location-D';
UPDATE 1
postgres=# select * from dummy_table;
 name |  address   | age
------+------------+-----
 XYZ  | location-A |  25
 ABC  | location-B |  35
 DEF  | location-C |  40
 GHI  | location-D |  54
(4 rows)

postgres=#
```

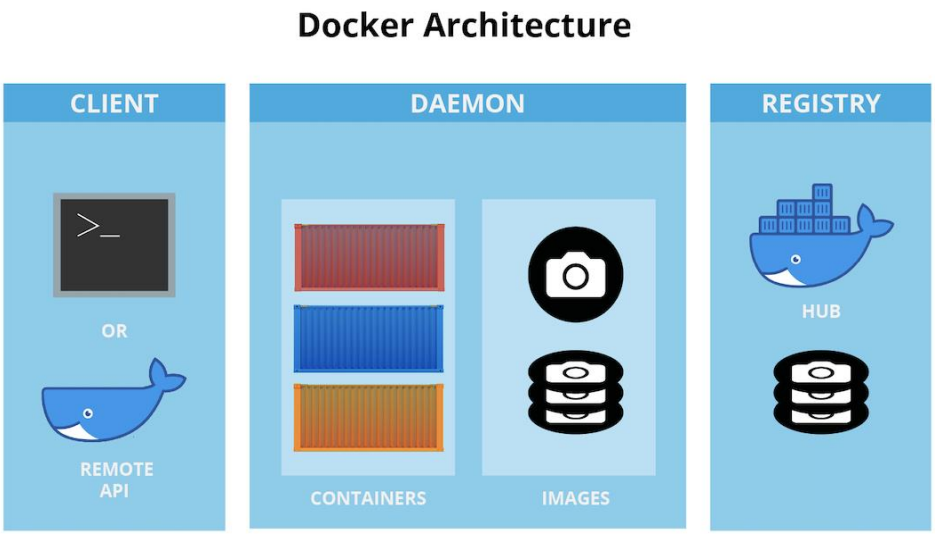
Podstawowe funkcje:

- Typy zdefiniowane przez użytkownika
- Dziedziczenie tabeli
- Mechanizm blokujący
- Integralność referencyjna klucza obcego
- Widoki, reguły, podzapytania
- Transakcje zagnieżdżone
- Kontrola współbieżności wielu wersji (MVCC)
- Replikacja asynchroniczna
- Natywna wersja Microsoft Windows Server
- Odzyskiwanie do punktu w czasie

5.14. Docker

Jest to platforma, która ułatwia programistom i administratorom tworzenie, wdrażanie i uruchamianie aplikacji rozproszonych. Docker jest to narzędzie, które służy do konteneryzacji, czyli

umieszczenia programu oraz jego zależności (biblioteki, pliki konfiguracyjne, lokalne bazy danych itp.) w lekkim, przenośnym, wirtualnym kontenerze. W przeciwieństwie do tradycyjnej wirtualizacji, kontener Docker działa na jądrze hostującego systemu operacyjnego. Rysunek 39 przedstawia architekturę Dockera.



Rysunek 39. Architektura Dockera. Źródło: [14].

Narzędzia Docker:

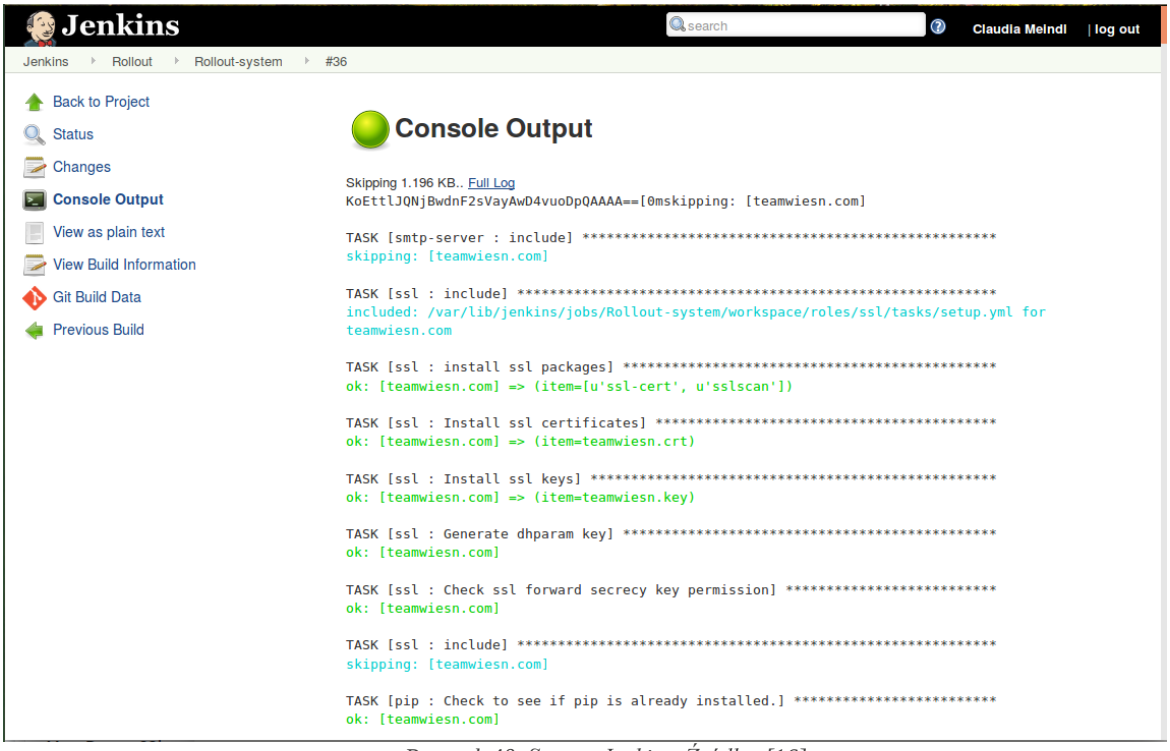
- **DockerFile –** według [38] każdy kontener Docker zaczyna się od prostego pliku tekstowego zawierającego instrukcje dotyczące budowania obrazu kontenera Docker. DockerFile automatyzuje proces tworzenia obrazu Dockera. Zasadniczo jest to lista instrukcji interfejsu wiersza poleceń (CLI), które Docker Engine uruchomi w celu złożenia obrazu. Lista poleceń Dockera jest ogromna, ale ustandaryzowana: operacje Dockera działają tak samo niezależnie od zawartości, infrastruktury lub innych zmiennych środowiskowych.
- **Obrazy Docker** - zawierają wykonywalny kod źródłowy aplikacji, a także wszystkie narzędzia, biblioteki i zależności, których kod aplikacji potrzebuje do działania jako kontener. Po uruchomieniu obrazu platformy Docker staje się on jedną instancją (lub wieloma instancjami) kontenera.
  Możliwe jest zbudowanie obrazu Dockera od zera, ale większość programistów pobiera je z popularnych repozytoriów. Z jednego obrazu podstawowego można utworzyć wiele obrazów Dockera i będą one miały wspólne cechy swojego stosu.
  Obrazy Dockera składają się z warstw, a każda warstwa odpowiada wersji obrazu. Za każdym razem, gdy programista wprowadza zmiany w obrazie, tworzona jest nowa górna warstwa, która zastępuje poprzednią górną warstwę jako bieżąca wersja obrazu. Poprzednie warstwy są zapisywane w celu wycofania lub ponownego wykorzystania w innych projektach.
  Za każdym razem, gdy tworzony jest kontener z obrazu Dockera, tworzona jest kolejna nowa warstwa, nazywana warstwą kontenera. Zmiany dokonane w kontenerze — takie jak dodanie lub usunięcie plików — są zapisywane tylko w warstwie kontenera i istnieją tylko wtedy, gdy kontener jest uruchomiony. Ten iteracyjny proces tworzenia obrazu umożliwia zwiększenie ogólnej wydajności, ponieważ wiele instancji kontenera na żywo może działać z tylko jednego obrazu podstawowego, a kiedy to robią, wykorzystują wspólny stos.
- **Kontenery Docker** - to działające na żywo instancje obrazów Docker. Podczas gdy obrazy Dockera są plikami tylko do odczytu, kontenery to dożywotnia, efemeryczna zawartość wykonywalna. Użytkownicy mogą z nimi wchodzić w interakcje, a administratorzy mogą dostosowywać ich ustawienia i warunki za pomocą poleceń platformy Docker.
- **Docker Hub** - to publiczne repozytorium obrazów platformy Docker, które nazywa się „największą na świecie biblioteką i społecznością obrazów kontenerów”. Przechowuje ponad 100 000 obrazów kontenerów pochodzących od komercyjnych dostawców oprogramowania, projektów *open source* i indywidualnych programistów. Obejmuje obrazy wyprodukowane

przez Docker, Inc., certyfikowane obrazy należące do Docker Trusted Registry oraz wiele tysięcy innych obrazów.

- **Docker Desktop** - to aplikacja dla komputerów Mac lub Windows, która zawiera Docker Engine, klienta Docker CLI, Docker Compose, Kubernetes i inne. Obejmuje również dostęp do Docker Hub.
- **Docker Daemon** - to usługa, która tworzy i zarządza obrazami Dockera za pomocą poleceń klienta. Zasadniczo demon Docker służy jako centrum sterowania implementacją Dockera. Serwer, na którym działa demon Dockera, nazywany jest hostem Dockera.
- **Docker Registry** - to skalowalny system przechowywania i dystrybucji obrazów platformy Docker typu open source. Rejestr umożliwia śledzenie wersji obrazów w repozytoriach za pomocą tagowania w celu identyfikacji. Osiąga się to za pomocą git, narzędzia do kontroli wersji.

5.15. Jenkins

Serwer służący do automatyzacji tworzenia oprogramowania. Umożliwia rozwój w trybie ciągłej integracji i ciągłego dostarczania. Działa on na serwerach Javy m.in. Apache Tomcat czy Jetty. Do pobierania kodu wykorzystuje kontrolę wersji taką jak Git. Rysunek 40 przedstawia przykład użycia serwera Jenkins.



Rysunek 40. Serwer Jenkins. Źródło: [16].

5.15.1. Automatyzacja wg Jenkinsa i jak ona działa

Według [40] Jenkins jest wiodącym serwerem automatyzacji typu *open source* z około 1600 wtyczkami wspierającymi usprawnienie wszelkiego rodzaju zadań programistycznych.

Jak działa Jenkins?

Jenkins jest dystrybuowany jako archiwum WAR i jako pakiety instalacyjne dla głównych systemów operacyjnych, jako pakiet Homebrew, obraz Dockera oraz jako kod źródłowy. Kod źródłowy to głównie Java, z kilkoma plikami Groovy, Ruby i Antlr.

Jenkins WAR można uruchomić samodzielnie lub jako serwlet na serwerze aplikacji Java, takim jak Tomcat. W obu przypadkach tworzy internetowy interfejs użytkownika i akceptuje wywołania swojego interfejsu API REST.

Kiedy jest uruchamiany po raz pierwszy, tworzy on użytkownika administracyjnego z długim losowym hasłem, które możesz wkleić na swojej początkowej stronie internetowej, aby odblokować instalację.

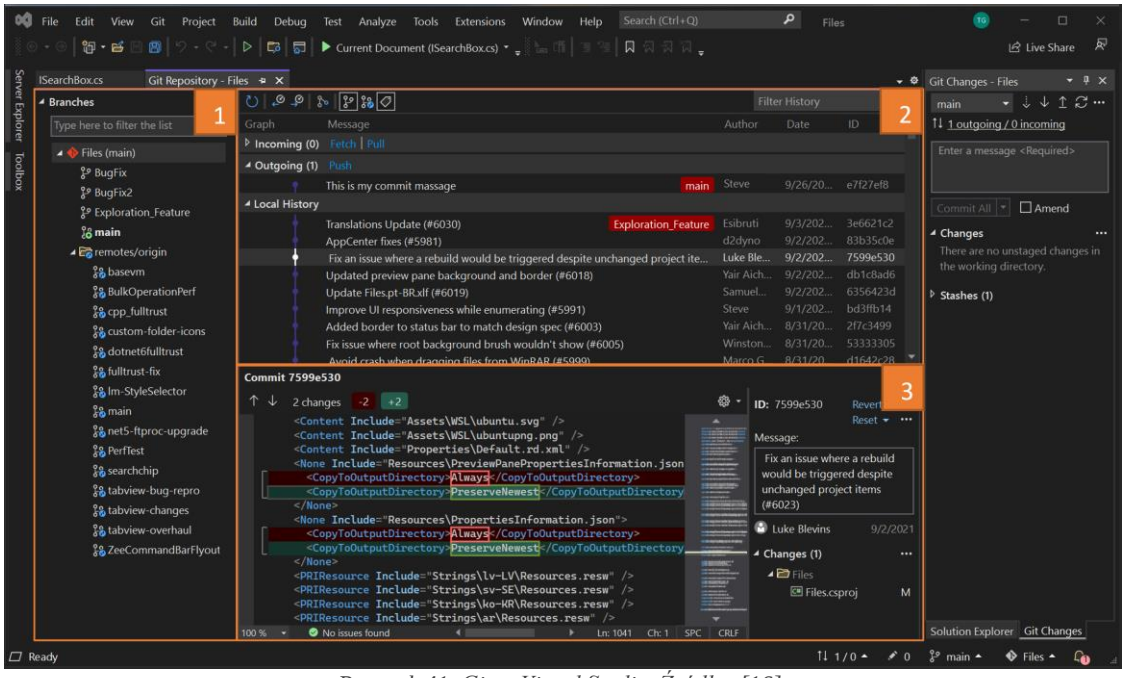
5.16. Git

Według [43] jest to rozproszony system kontroli wersji typu *open source*, zaprojektowany do obsługi wszystkiego, od małych do bardzo dużych projektów, zapewniając odpowiednią szybkość i wydajność.

Najważniejsze cechy

- wsparcie dla rozgałęzionego procesu tworzenia oprogramowania
- możliwość pracy *off-line*
- wsparcie dla istniejących protokołów sieciowych
- efektywna praca z dużymi projektami
- każda rewizja to obraz całego projektu

Rysunek 41 przedstawia przegląd przykładowych repozytoriów oraz porównywanie gałęzi na przykładzie Visual Studio.



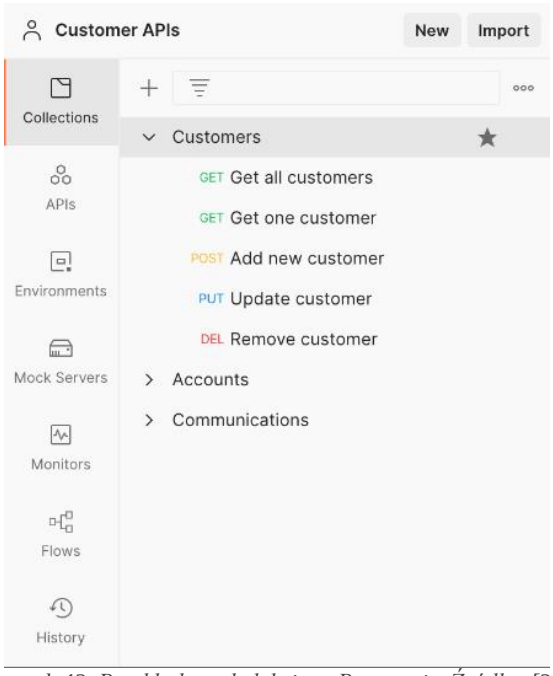
Rysunek 41. Git w Visual Studio. Źródło: [16].

5.17. Postman

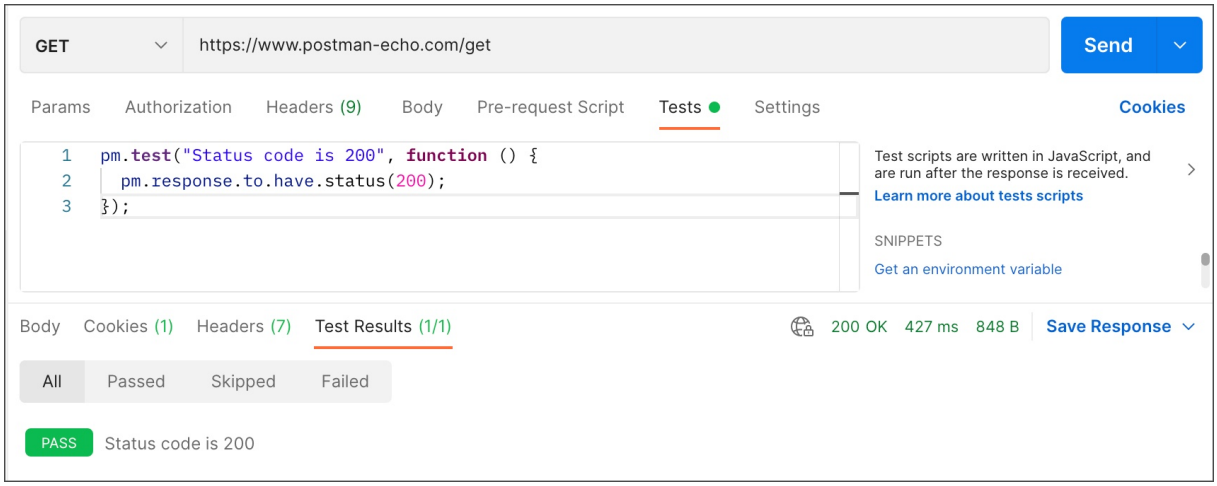
Według [30] jest to narzędzie, które ułatwia pracę, oferując przejrzysty interfejs aplikacji idealny do projektowania, budowania, testowania i iterowania API. Aplikacja jest dostępna dla użytkowników w formie desktopowej oraz w przeglądarce internetowej.

Głównymi funkcjonalnością Postmana jest tworzenie kolekcji, które grupują i porządkują wszystkie powiązane zapytania wraz z niezbędnymi danymi. Ponadto oferuje możliwość wykonywania automatycznych testów, które pozwalają na zmniejszenie liczby nieprzewidywalnych błędów w

aplikacji. Rysunek 42 i 43 przedstawia przykładowe zastosowanie Postmana np. stworzenie kolekcji oraz testów.



Rysunek 42. Przykładowa kolekcja w Postmanie. Źródło: [30].



Rysunek 43. Przykładowy test API w Postmanie. Źródło: [30].

5.17.1. Metody

Postman oferuje wiele metod interakcji z *endpointami*. Oto niektóre z najczęściej używanych:

- `GET`: Zdobądź informacje
- `POST`: Dodaj informacje
- `PUT`: Zastąp informacje
- `PATCH`: Zaktualizuj niektóre informacje
- `DELETE`: Usuń informacje

5.17.2. Kody odpowiedzi

Podczas testowania interfejsów API za pomocą Postmana zazwyczaj uzyskujemy różne kody odpowiedzi. Niektóre z najczęstszych obejmują:

- 100 > Reakcje czasowe, na przykład „Przetwarzanie 102”.
- 200 > Odpowiedzi, w których klient akceptuje żądanie, a serwer pomyślnie je przetwarza, na przykład „200 OK”.
- 300 > Odpowiedzi związane z przekierowaniem adresu URL, na przykład „301 przeniesiony na stałe”.
- 400 > Reakcje klienta na błędy, na przykład „400 Bad Request”.
- 500 > Odpowiedzi na błędy serwera, na przykład „500 Wewnętrzny błąd serwera”.

5.17.3. Kolekcje

Według [39] Postman daje możliwość grupowania różnych zapytań. Ta funkcja jest znana jako „kolekcje” i pomaga organizować testy.
Te kolekcje to foldery, w których przechowywane są żądania i mogą być uporządkowane w dowolny sposób preferowany przez zespół. Możliwy jest również ich eksport-import.

5.17.4. Środowiska

Postman pozwala nam również tworzyć różne środowiska poprzez generowanie/wykorzystywanie zmiennych; na przykład zmienna adresu URL skierowana do różnych środowisk testowych (dev-QA), umożliwiająca nam przeprowadzanie testów w różnych środowiskach przy użyciu istniejących żądań.

5.18. Swagger

Według [31] jest to zestaw narzędzi, które podobnie jak Postman pomagają m.in. programistom projektować, tworzyć, dokumentować i korzystać z usług REST API. Dużą zaletą korzystania z tego narzędzia jest to, że dostarcza on samoaktualizującą się dokumentację. Aby móc korzystać ze Swaggera posiadając aplikację Spring Boot należy zdefiniować zależności przedstawione na Listingu 20.

Listing 20. Przykładowe użycie Swaggera w aplikacji Spring Boot. Źródło: [31].

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

5.18.1. Użycie

Wykorzystanie narzędzi Swaggera można podzielić na różne przypadki użycia: programowanie, interakcja z interfejsami API i dokumentacja.

Programowanie interfejsów API
Podczas tworzenia interfejsów API narzędzia Swagger mogą być używane do automatycznego generowania dokumentu Open API na podstawie samego kodu. To osadza opis API w kodzie źródłowym projektu i jest nieformalnie nazywane tworzeniem API metodą *code-first* lub *bottom-up*.
Alternatywnie, używając Swagger Codegen, programiści mogą oddzielić kod źródłowy od dokumentu Open API i wygenerować kod klienta i serwera bezpośrednio z projektu.

Interakcja z interfejsami API

Korzystając z projektu Swagger Codegen, użytkownicy końcowi generują pakiety SDK klienta bezpośrednio z dokumentu OpenAPI, zmniejszając potrzebę tworzenia kodu klienta przez człowieka. Od sierpnia 2017 r. projekt Swagger Codegen obsługiwał ponad 50 różnych języków i formatów generowania SDK klienta.

Dokumentowanie interfejsów API
Gdy jest to opisane w dokumencie OpenAPI, narzędzia Swagger typu open source mogą być używane do bezpośredniej interakcji z interfejsem API za pośrednictwem interfejsu użytkownika Swagger. Umożliwia to bezpośrednie połączenia z aktywnymi interfejsami API za pośrednictwem interaktywnego interfejsu użytkownika opartego na języku HTML. Żądania można składać bezpośrednio z interfejsu użytkownika.

5.19. JUnit

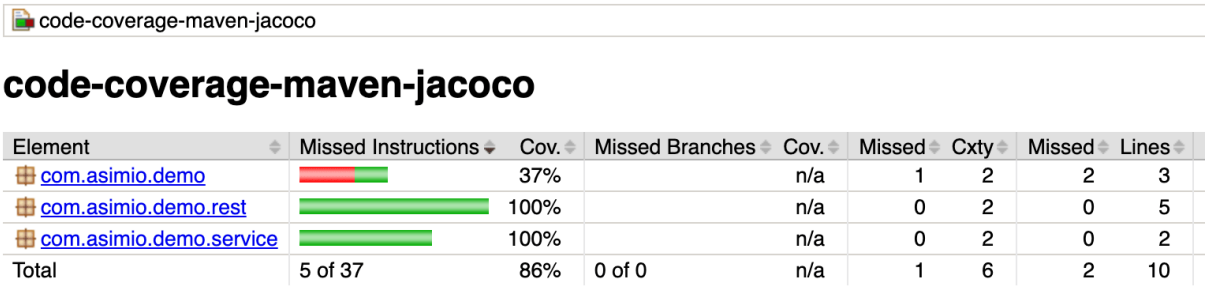
Według [32] JUnit to test jednostkowy, który docelowo sprawdza, czy dana funkcjonalność działa zgodnie z jej przeznaczeniem. Jego cechy to: metoda najmniejszą jednostką testowania, tworzenie przypadków testowych, oddzielenie testów od kodu, wiele mechanizmów uruchamiania, tworzenie raportów, integracja z różnymi środowiskami programistycznymi. Na Listingu 21 przykładowe wywołanie testu.

Listing 21. Przykładowe wywołanie testu JUnit. Źródło: [32].

```
public class HelloWorld extends TestCase
 {
   public void testMultiplication()
   {
     // Testing if 2*2=4:
     assertEquals ("Multiplication", 4, 2*2);
   }
 }
```

5.20. JaCoCo

Według [33] jest to generator raportów pokrycia kodu m.in. dla projektów Java. Pokrycie kodu to metryka oprogramowania używana do mierzenia, ile linii naszego kodu jest wykonywanych podczas testów automatycznych. Uruchomienie testu za pomocą JUnit automatycznie uruchomi agenta JaCoCo. Spowoduje to utworzenie raportu pokrycia w formacie binarnym w katalogu docelowym, target/jacoco.exec. Na Rysunku 44 przedstawiony jest przykładowy raport JaCoCo.

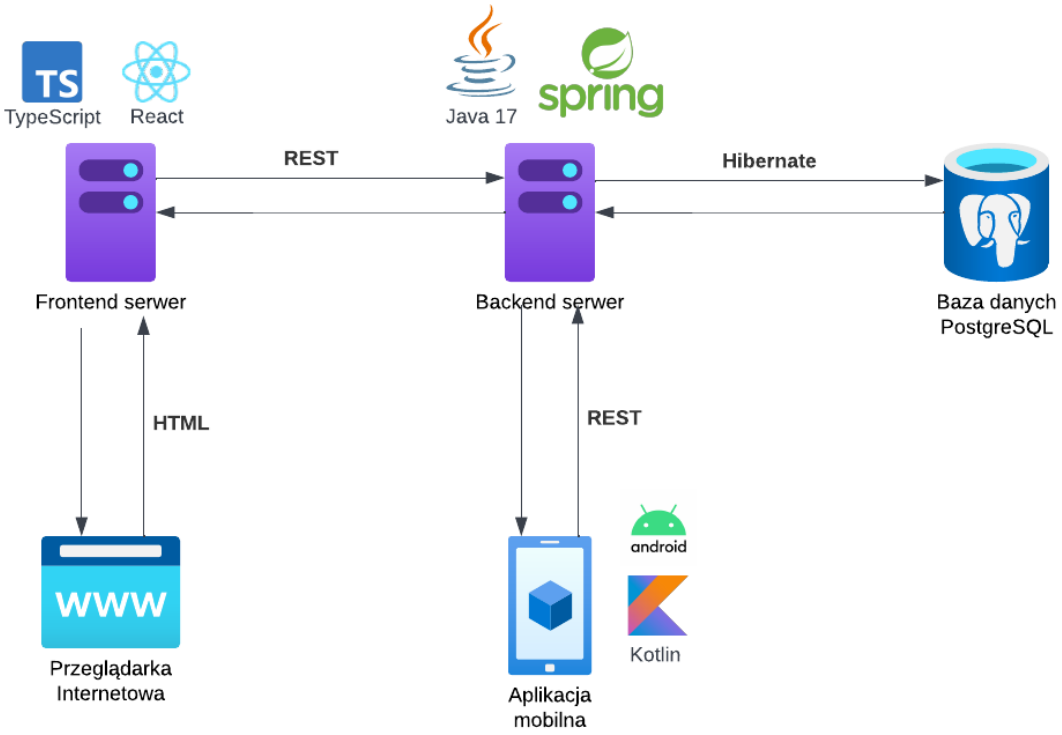


Rysunek 44. Przykładowy raport JaCoCo. Źródło: [34].

6. Implementacja systemu

W tym rozdziale szczegółowo zostanie przedstawiony etap implementacji Kompleksowego Systemu Obsługi Incydentów Ratunkowych – GARY.

Rysunek 45 przedstawia architekturę systemu dla niniejszej pracy. Centralnym jej elementem jest serwer napisany w języku Java z wykorzystaniem Spring. Serwer łączy się przy pomocy Hibernate z bazą danych PostgreSQL. Z serwerem *backend* komunikuje się serwer aplikacji webowej oraz aplikacja mobilna. Komunikacja odbywa się za pomocą zapytań REST. Aplikacja mobilna powstała na telefony z systemem Android, została napisana w języku Kotlin. Aplikacja webowa została napisana przy pomocy TypeScript oraz React. Całość komunikuje się z przeglądarką użytkownika przy pomocy zapytań HTML.



Rysunek 45. Architektura systemu Gary. Źródło: Opracowanie własne

6.1. Wstęp

Zespół implementacji przez cały proces powstawania systemu współpracował z zespołem analizy i testowania. Dzięki temu, wszystkie niejasności i nieścisłości były wyjaśniane na bieżąco, co pozwoliło zachować płynność w postępach w pracy nad aplikacją. Zespół został podzielony na kilka podzespołów odpowiedzialnych za:

- *backend* – implementacja wszystkich funkcjonalności po stronie serwera
- *frontend* – część kliencka odpowiedzialna za interakcję z użytkownikiem za pomocą interfejsu użytkownika w aplikacji

Po stronie *frontendu* można wyróżnić dwa mniejsze zespoły – jeden odpowiedzialny za aplikację webową, drugi za aplikację mobilną dostępną na system Android.

Zazwyczaj prace w poszczególnych podzespołach były zrównoleglone, aby osiągnąć jak najwyższą wydajność pracy.

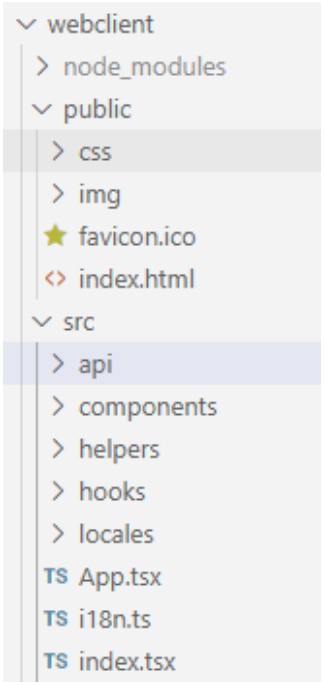
Po implementacji danej funkcjonalności zespół testerów zajmował się skrupulatnym testowaniem zgodności aplikacji z dokumentacją oraz wychwytywaniem potencjalnych

nieprawidłowości. Dzięki temu zostało wyeliminowane ryzyko późniejszych poważnych błędów, które mogłyby skutkować w opóźnieniu prac nad aplikacją.

W poniższych podrozdziałach został przedstawiony opis pracy poszczególnych grup zespołu implementacji.

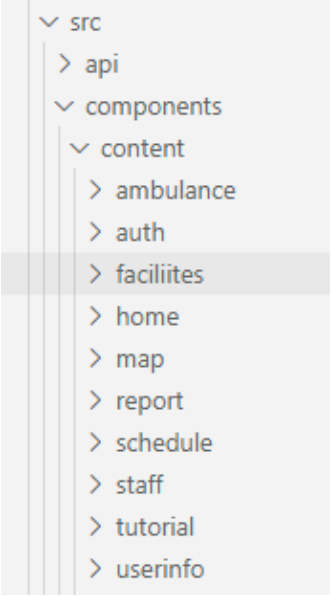
6.2. Implementacja aplikacji webowej

Aplikacja webowa była tworzona przy użyciu oprogramowania Visual Studio Code z wykorzystaniem języka TypeScript, biblioteki React, języka znaczników HTML i arkusza styli CSS. Sporządzony został szkielet projektu, w którym wyróżniamy dwie główne części – public i src. Public zawiera zdjęcia, grafiki, ikony oraz style CSS. Natomiast w części źródłowej znajdują się pliki odpowiedzialne za wygląd stron, komunikację z serwerem oraz niektóre funkcjonalności występujące po stronie klienckiej. Rysunek 46 przedstawia szkielet aplikacji w Visual Studio Code.



Rysunek 46. Projekt aplikacji webowej GARY w Visual Studio Code. Źródło: Opracowanie własne.

Najważniejsza część projektu znajduje się w podfolderze *components*. Zawiera on widoki poszczególnych stron aplikacji. Każdy z nich jest w odpowiadającym mu folderze. np. widok dla obsługi karetek znajduje się w *ambulance*. Rysunek 47 przedstawia zawartość folderu *components*.



Rysunek 47. Zawartość folderu components. Źródło: Opracowanie własne.

W następujących podrozdziałach zostały przedstawione, stosowane dla aplikacji webowej biblioteki.

6.2.1 Biblioteka i18n

Jest to biblioteka, który została wykorzystana do implementacji tłumaczenia stron zawartych w aplikacji. Dzięki niej można przygotować kilka różnych wersji językowych. Pozwala na korzystanie z wielu plików z tłumaczeniami i załadowania ich w dowolnym momencie. Do przechowywania tłumaczeń wykorzystuje się pliki json i są one zapisywane w postaci ”klucz”:”wartość”. Listing 22 przedstawia zastosowanie biblioteki i18n.

Listing 22. Tłumaczenie z wykorzystaniem biblioteki i18n (1). Źródło: Opracowanie własne.

```
”AmbulanceClass”: {
      ”BASIC”: ”Basic”,
      ”SPECIAL”: ”Special”,
      ”TRANSPORT”: ”Transport”,
      ”NEONATOLOGY”: ”Neonatology”
      },
```

W projekcie zostało użyte tłumaczenie z języka angielskiego na język polski. W tym celu został skonfigurowany plik przeznaczony do zarządzania tłumaczeniem. Listing 23 przedstawia przypisanie plików json do odpowiednich translacji.

Listing 23. Tłumaczenie z wykorzystaniem biblioteki i18n (2). Źródło: Opracowanie własne.

```
import resourcesPL from ”./locales/resources.pl.json”;
import resourcesEN from ”./locales/resources.en.json”;
const resources = {
      pl: {
            translation: resourcesPL
      },
      en: {
            translation: resourcesEN
      }
;
```

Wyświetlanie tekstu zapewnione jest dzięki komponentowi *'useTranslation',* co zostało przedstawione na Listingu 24.

Listing 24. Tłumaczenie z wykorzystaniem biblioteki i18n (3). Źródło: Opracowanie własne.

```
import { useTranslation } from ”react-i18next”;
const { t } = useTranslation();
```

6.2.2. Biblioteka Leaflet

Jest to biblioteka służąca do obsługi map np. OpenStreetMap. Pozwala stylizować i tworzyć interaktywne warstwy oraz umożliwia korzystanie z danych w standardzie GeoJson. Zastosowanie tej biblioteki zostało przedstawione na Listingu 25.

W aplikacji webowej ta biblioteka została użyta do pokazania placówek, zgłoszeń czy nawet tras karetek. W łatwy sposób można umieścić tam własne ikony oraz pokazać trasę przemieszczającego się obiektu, w naszym przypadku ambulansów. Użytkownik może wskazać lokalizację zdarzenia poprzez kliknięcie na mapę, a Leaflet zwraca nam dokładne współrzędne.

Listing 25. Obsługa map z wykorzystaniem biblioteki Leaflet (1). Źródło: Opracowanie własne.

```
<MapContainer center={props.center} zoom={props.initialZoom}>
<TileLayer attribution='&copy; <a href
https://www.openstreetmap.org/copyright>OpenStreetMap</a>
contributors'
     url=https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png />
```

Listing 26 przedstawia importowanie niezbędnych komponentów.

Listing 26. Obsługa map z wykorzystaniem biblioteki Leaflet (2). Źródło: Opracowanie własne.

```
import { MarkGeocodeEventHandlerFn, MarkGeocodeEvent } from
”leaflet-control-geocoder/dist/control”;
import Geocoder, { geocoders } from “leaflet-control-geocoder”;
import {MapContainer, TileLayer, Polyline, Marker, Popup,
useMapEvents, useMap } from “react-leaflet”;
```

Następnie zaimportowane komponenty zostały odpowiednio przypisane i zaimplementowane, co przedstawia Listing 27.

Listing 27. Obsługa map z wykorzystaniem biblioteki Leaflet (3). Źródło: Opracowanie własne.

```
return (
<MapContainer center={props.center} zoom={props.initialZoom}>
<TileLayer attribution='&copy; <a href
https://www.openstreetmap.org/copyright>OpenStreetMap</a>
contributors'
     url=https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png />
{props.searchable ? <GeocoderMenu geocoder={geocoder}
onSearch={props.onSearch} /> : “”}
{props.clickable ? <ClickHandler geocoder={geocoder}
onClick={props.onClick} /> : “”}
{props.paths ? props.paths.map((path, index) => <Polyline
key={index} positions={path.points} color={path.color} /> : “”}
{props.marks ? props.marks.map((pos. index) => {
<Marker key={index} position={pos.coords} icon={pos.icon}>
     {pos.desc ? (
          <Popup>
```

```
{pos.to ? <NavButton to={pos.to}>{pos.desc}</NavButton> : pos.desc}
                                    </Popup>
                               ) : ””}
                          </Marker>
                    )) : ””
               </MapContainer>
     );
);
```

6.2.3. Biblioteka Recharts

W projekcie biblioteka Recharts została wykorzystana do tworzenia wykresów. Na Listingu 28 zostały przedstawione użyte komponenty.

Listing 28. Implementacja wykresów z wykorzystaniem biblioteki Recharts (1). Źródło: Opracowanie własne.

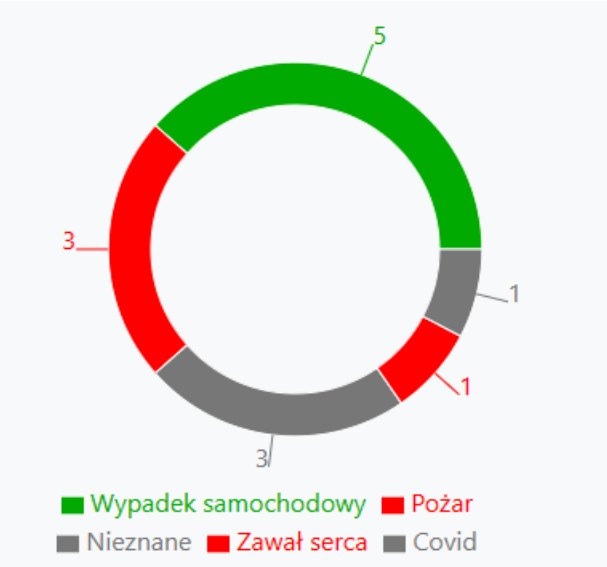
```
import { AreaChart as Inner, CartesianGrid, XAxis, YAxis, Tooltip,
Area, Legend } from "recharts";
```

Zastosowanie biblioteki w projekcie do implementacji wykresów ukazuje Listing 29.

Listing 29. Implementacja wykresów z wykorzystaniem biblioteki Recharts (2). Źródło: Opracowanie własne.

```
return (
     <Inner width={props.width} height={props.height}
     data={props.data}
          syncId={props.syncId} margin={props.margin}
     onClick={props.onClick}>   {props.grid ? <CartesianGrid
     strokeDasharray="3 3"
     stroke={stroke} /> :
          ""}
          <XAxis dataKey="key" stroke={stroke} />
          <YAxis stroke={stroke} />
          {props.tooltip ? <Tooltip
          wrapperClassName={`bg${customTheme(darkMode)}`}
          cursor={{
                    stroke: stroke
          }} /> : ""}
          {props.settings.map((set, index) => <Area
          dataKey={`values.${set.key}`}
          type={set.type} stroke={set.stroke} fill={darkMode ?
     set.fillDark :
          set.fill}
          fillOpacity={set.opacity} key={index} name={set.key} />)}
          {props.legend ? <Legend /> : ""}
     </Inner>
  );
```

Rysunek 48 przedstawia przykładowy diagram z aplikacji webowej, który został stworzony z wykorzystaniem biblioteki Recharts.



Rysunek 48. Diagram otwartych zgłoszeń z użyciem biblioteki Recharts. Źródło: Opracowanie własne.

6.2.4. Biblioteka React-icons

Wykorzystanie tej biblioteki pozwoliło na łatwe dodanie ikon do projektu. Ponadto React-icons umożliwia stylizowanie ich, żeby zachować spójność z aplikacją. Dużym plusem wynikającym z jej użycia jest to, że wymagane jest jedynie zaimportowanie tych komponentów, które są używane w projekcie. Listing 30 i 31 przedstawia implementację tej biblioteki w projekcie.

Listing 30. Implementacja ikon z wykorzystaniem biblioteki React-icons (1). Źródło: Opracowanie własne.

```
import { AiOutlineStar, AiFillStar } from "react-icons/ai";
```

Listing 31. Implementacja ikon z wykorzystaniem biblioteki React-icons (2). Źródło: Opracowanie własne.

```
return <Inner emptySymbol={<AiOutlineStar size={35}/>}
                fullSymbol={<AiFillStar size={35}/>}
                onClick={props.onClick}
                onChange={props.onChange}
                onHover={props.onHover}
                initialRating={props.initialValue &&
props.initialValue > 0 ? calcRating(props.initialValue) : 0}
                readonly={props.disabled}/>;
};
```

6.2.5. Biblioteka React-rating

W projekcie React-rating znalazła zastosowanie w implementacji systemu oceniania poradników. Wykorzystuje ona niestandardowe wbudowane style oraz narzędzia takie jak Bootstrap. Listing 32 i 33 obrazuje wykorzystanie tej biblioteki w aplikacji.

Listing 32. Implementacja poradników z wykorzystaniem biblioteki React-rating (1). Źródło: Opracowanie własne.

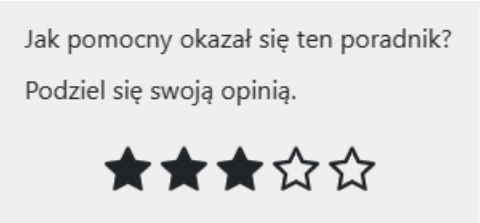
```
import Inner from "react-rating";
```

Listing 33. Implementacja poradników z wykorzystaniem biblioteki React-rating (2). Źródło: Opracowanie własne.

```
return <Inner emptySymbol={<AiOutlineStar size={35}/>}
                fullSymbol={<AiFillStar size={35}/>}
```

```
                onClick={props.onClick}
                onChange={props.onChange}
                onHover={props.onHover}
                initialRating={props.initialValue          &&
props.initialValue > 0 ? calcRating(props.initialValue) : 0}
                readonly={props.disabled}/>;
};
```

Rysunek 49 przedstawia zaimplementowany React-rating z ocenionym przez użytkownika poradnikiem.



Rysunek 49. Ocena poradnika z użyciem React-rating. Źródło: Opracowanie własne.

6.2.6. Biblioteka Axios

Biblioteka ta jest używana jako klient HTTP, dzięki któremu można stworzyć zapytania w przeglądarce. Zwraca status pozytywny lub negatywny dla danego zapytania. Umożliwia wysyłanie zapytań typu Get i Post.

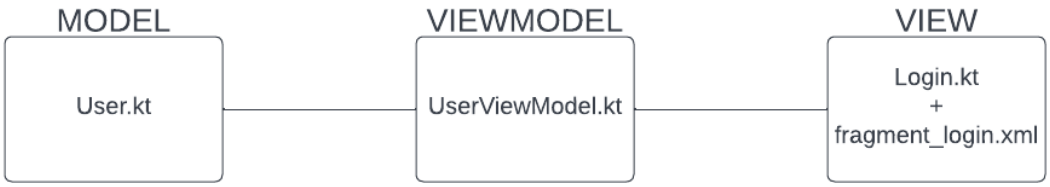
6.3. Implementacja aplikacji mobilnej dla systemu Android

Aplikacja mobilna została zaprojektowana w oparciu o wzorce takie jak MVVM oraz Adapter. Pomimo możliwości użycia Java do implementacji aplikacji, zdecydowaliśmy się na Kotlin. Język ten jest nowy dla wszystkich osób uczestniczących w projekcie. Chcieliśmy, aby nasza aplikacja wpasowywała się w obecne trendy na rynku mobilnym. Obecnie większość aplikacji przeznaczonych na Androida jest napisana w Kotlinie, który jest przystosowany do tworzenia aplikacji mobilnych, w odróżnieniu od Javy, która jest jest uniwersalnym językiem. Ich zastosowanie zostanie przedstawione w poniższych podrozdziałach.

6.3.1. Wzorzec MVVM

Wzorzec projektowy MVVM, czyli *Model - View – ViewModel*. Całość opiera się na wydzieleniu trzech logicznych warstw w systemie, co ma na celu podział odpowiedzialności i zmniejszenie zależności pomiędzy odpowiednimi warstwami. Zastosowanie wzorca w naszym projekcie zostało przedstawione na Rysunku 50. Wyróżniamy:

- Model, czyli najniższą warstwę. Zawiera ona klasy, których zadaniem jest odzwierciedlenie domeny oraz dodatkowo warstwę dostępu do danych.
- View, która jest odpowiedzialna za wizualną reprezentację oraz interakcję z użytkownikiem. W przypadku aplikacji mobilnej na Androida znajdzie się tutaj fragment/activity razem z definicją pliku w XML. Warstwą widoku nazwiemy również pliki korzystające z Android Compose. W aplikacji została napisana w językach Kotlin i XML z użyciem klas Fragment.
- ViewModel ma za zadanie udostępnianie danych widokowi oraz wymianę informacji z modelem. Jej funkcją jest pobieranie i aktualizacja danych. ViewModel zawiera sam model widoku oraz udostępnia tylko to czego potrzebuje.



Rysunek 50. Zastosowanie wzorca MVVM w projekcie Gary. Źródło: Opracowanie własne.

6.3.2. Wzorzec Adapter

Adapter jest to strukturalny wzorzec projektowy, którego celem jest umożliwienie współpracy dwóm klasom o niekompatybilnych interfejsach. Adapter ma za zadanie przekształcenie interfejsu jednej z klas na interfejs drugiej klasy oraz opakowanie istniejącego interfejsu w nowy. Adapter jest swego rodzaju przejściówką pomiędzy dwoma bytami. Istnieją dwa rodzaje tego wzorca: Adapter Klasowy oraz Adapter Obiektowy. W sytuacji, gdy nie można rozszerzyć klasy adaptowanej używany jest Adapter Obiektowy, natomiast gdy klasa adaptowana jest abstrakcyjna stosuje się Adapter Klasowy. W większości sytuacji stosuje się Adapter Obiektowy, ponieważ jest bezpieczniejszy.

6.3.3. Biblioteka Retrofit

Komunikacja z back-endem aplikacji odbywa się za pomocą biblioteki Retrofit. Jest to najpopularniejsza biblioteka do wykonywania zapytań http, używana do otrzymywania informacji z API. W niniejszym projekcie w klasie obsługującej konfigurację powołujemy instancje klasy Retrofit, gdzie zostały dostarczone informacje takie jak np.: adres API.
Listing 34 przedstawia instację z użyciem biblioteki Retrofit.

Listing 34. Zastosowanie biblioteki Retrofit (1). Źródło: Opracowanie własne.

```
object RetrofitInstance {
    private val retrofit by lazy {
        Retrofit.Builder().baseUrl(PJATK).addConverterFactory(
        GsonConverterFactory.create(gson))
        .build()
    }

    val api: BackendAPI by lazy {
        retrofit.create(BackendAPI::class.java)
    }
}
```

Przykład metody pobierającej poradniki z back-endu, z wykorzystaniem naszej instancji klasy Retrofit obrazuje Listing 35.

Listing 35. Zastosowanie biblioteki Retrofit (2). Źródło: Opracowanie własne.

```
object Repository {
     suspend fun getTutorials(): Response<List<Tutorial>> {
          return RetrofitInstance.api.getTutorials()
}
```

6.4. Implementacja części serwerowej – backend

Serwer został zaimplementowany w oparciu o wzorce takie jak MVC oraz Factory. Ich zastosowanie oraz sposób praktyki zastosowane w implementacji zostaną przedstawione w poniższych podrozdziałach.

6.4.1. Wzorzec MVC

Skrót MVC oznacza model – view – controller i jest najpopularniejszym stosowanym wzorcem w tworzeniu oprogramowania. Wyszczególniamy w nim trzy elementy:

- Model który przedstawia logikę biznesową. Wszystkie klasy umieściliśmy w folderze o tej samej nazwie oraz opatrzyliśmy adnotacją @Entity, patrz Listing 36.

Listing 36. Przykład zastosowania modelu w projekcie Gary. Źródło: Opracowanie własne.

```
@Getter
@Setter
@Entity
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Facility {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer facilityId;

	private String name;

	@Embedded
	private Location location;

	@Enumerated(EnumType.STRING)
	private FacilityType facilityType;
}
```

- Kontroler używany jest do obsługi żądań użytkowników. Na zdjęciu poniżej widać kontroler z przykładową końcówką dla modelu Facility przedstawionego na Listingu 37. Za pomocą adnotacji *@RequestMapping* możemy określić dokładną ścieżkę do danej końcówki. Oprócz tego możemy pobierać z żądania informacje od użytkownika za pomocą dwóch adnotacji: *@PathVariable* oraz *@RequestBody*. Kiedy otrzymujemy dane z ciała żądania są one sprawdzane pod względem poprawności. Czy wymagane stringi nie są puste, czy liczby mieszczą się w zakresie. Odpowiada za to postawiona przed *@RequestBody* adnotacja *@Valid*.

Listing 37. Przykład zastosowania kontrolera w projekcie Gary. Źródło: Opracowanie własne.

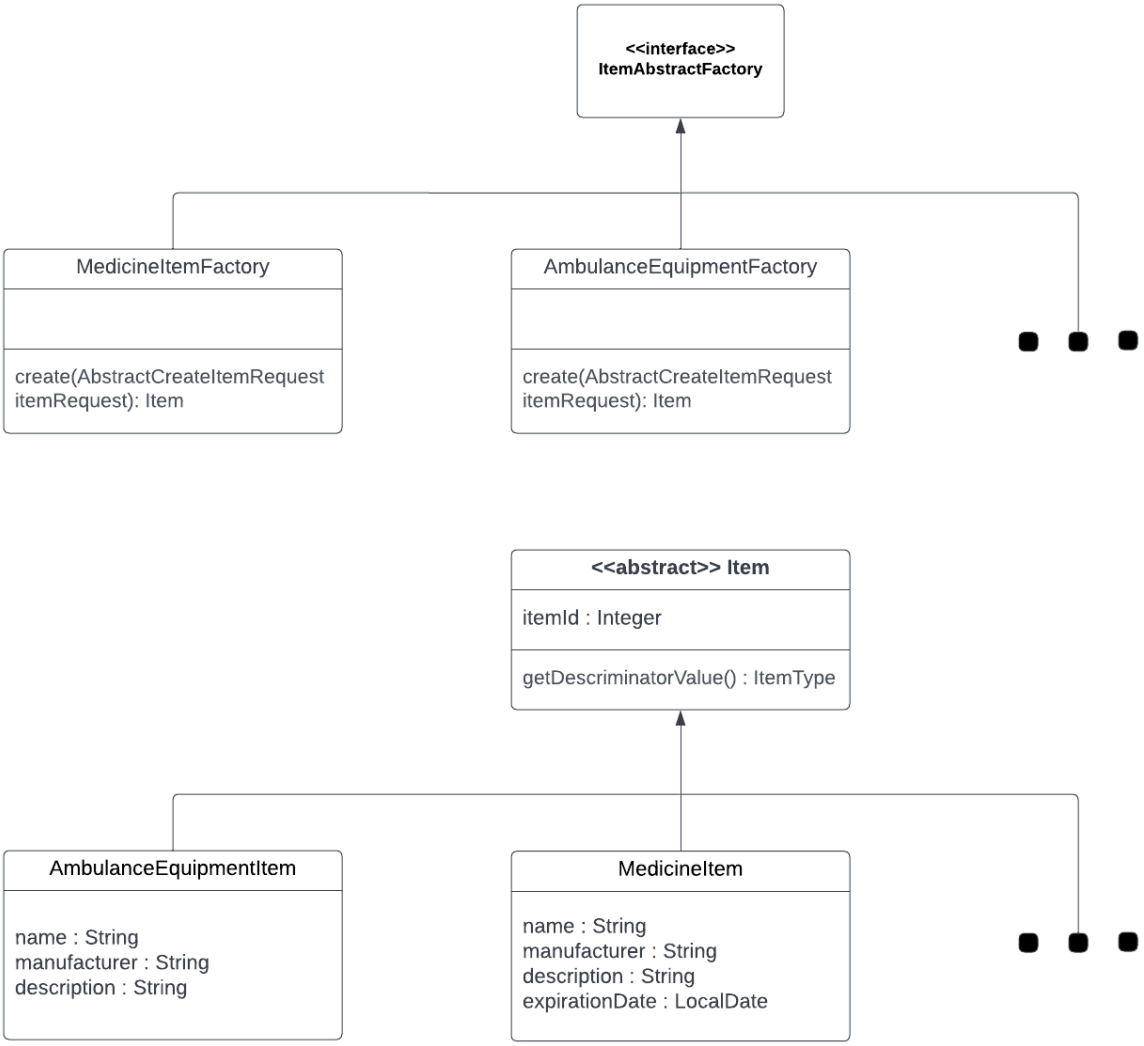
```
@RequiredArgsConstructor
@RestController
@RequestMapping("/facility")
public class FacilityController {
	private final FacilityService facilityService;

	@GetMapping
	public ResponseEntity<?> getAll(){
		return ResponseEntity.ok(facilityService.getAll());
	}
```

- Widok czyli interfejs użytkownika przechowujemy w folderze contents dla aplikacji webowej

6.4.2. Wzorzec Factory

Wzorzec ten pozwala na tworzenie wielu obiektów jednego typu, które dziedziczą po jednym interfejsie.

Zastosowaliśmy ten wzorzec dla modeli związanych z wyposażeniem i jego elementami. Szczegóły można zaobserwować na diagramie poniżej. Zdecydowaliśmy się na ten wzorzec ze względu na łatwe dodawanie nowych przedmiotów do wyposażenia, co niewykluczone że wystąpi wraz z rozwojem aplikacji. Rysunek 51 przedstawia schemat zastosowania w omawianej pracy.



Rysunek 51. Schemat wzorca Factory. Źródło: Opracowanie własne

6.4.3. DTO

W niniejszej pracy zostało użyte DTO (Data Transfer Object) jest to jeden z istotniejszych elementów architektury oprogramowania. Pozwala na przenoszenie danych między wieloma warstwami. DTO użyliśmy do niektórych żądań i odpowiedzi w naszym REST API. Dzięki temu treści żądań są krótsze, a odpowiedzi nie zawierają nieskończonych pętli spowodowanych dwukierunkowymi asocjacjami. Na Listingu 38 znajduje się przykład odpowiedzi, a Listing 39 przedstawia odpowiadający mu model.

Listing 38. Zastosowanie DTO w projekcie Gary (1). Źródło: Opracowanie własne

```
@Getter
@Setter
@Builder
public class AllergyResponse {
      private Integer allergyId;
      private AllergyType allergyType;
      private String allergyName;
      private String other;
}
```

Listing 39. Zastosowanie DTO w projekcie Gary (2). Źródło: Opracowanie własne

```
@Builder
@NoArgsConstructor
@Table(name = "Allergy")
@Entity
@AllArgsConstructor
@Getter
@Setter
public class Allergy {
      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private Integer allergyId;

      @Enumerated(EnumType.STRING)
      private AllergyType allergyType;

      @Column(nullable = false, unique = true)
      private String allergyName;

      private String other;

      @ManyToMany
      @JoinTable(name = "allergy_medical_infos",
                  joinColumns = @JoinColumn(name = "allergy"),
                  inverseJoinColumns     =     @JoinColumn(name     =
"medical_info"))
      private Set<MedicalInfo> medicalInfos = new LinkedHashSet<>();
}
```

6.4.4. Komunikacja z bazą danych

Na potrzeby naszej pracy przechowujemy spore zbiory danych powiązanych ze sobą. Do komunikacji z bazą użyliśmy Hibernate wraz z frameworkiem Spring. Aby uzyskać dostęp do danych stworzyliśmy interfejsy DAO (Data Access Object) opatrzone adnotacją @Repository, co można zauważyć na Listingu 40.

Listing 40. Komunikacja z bazą danych. Źródło: Opracowanie własne

```
@Repository
public interface FacilityRepository extends JpaRepository<Facility,
Integer> {
      Optional<Facility> findByFacilityId(Integer id);
}
```

Na Listingu 41 można zobaczyć w klasie serwisowej użycie interfejsu DAO w celu znalezienia rekordu oraz jego usunięcia.

Listing 41. Interfejs DAO. Źródło: Opracowanie własne

```
public void delete(Integer id){
     Optional<Facility>             facilityOptional             =
facilityRepository.findByFacilityId(id);
     if(facilityOptional.isEmpty()){
          throw      new      HttpException(HttpStatus.NOT_FOUND,
String.format("Cannot find facility with %s", id));
     }
     facilityRepository.delete(facilityOptional.get());
}
```

6.4.5. Dostęp do funkcjonalności wymagających logowania

Z uwagi na używane technologie w niniejszej pracy zostało użyte Spring Security. Pomaga nam w autentykacji oraz uwierzytelnianiu użytkowników. Dodatkowo zastosowaliśmy rolę, które określają zakres uprawnień zalogowanych użytkowników. Co można zobaczyć na Listingu 42.

Listing 42. Zastosowanie Spring Security w projekcie Gary. Źródło: Opracowanie własne

```
.antMatchers("/dispatch/**").hasRole("DISPATCHER")
```

6.4.6. Obsługa błędów z REST API

W niniejszej pracy zostały zaimplementowane mechanizmy zwracające status błędu wraz z wiadomością dla błędnych żądań.
Na listingu 43 można zobaczyć klasę, którą stosowaliśmy do informowania o błędnych żądaniach.

Listing 43. Implementacja obsługi błędów API. Źródło: Opracowanie własne

```
public class HttpException extends HttpStatusCodeException {
    public HttpException(HttpStatus statusCode) {
        super(statusCode);
    }

    public HttpException(HttpStatus statusCode, String statusText)
{
        super(statusCode, statusText);
    }
}
```

6.4.7. Odwrócenie sterowania

Z racji użycia frameworku Spring została mu przekazana kontrola nad obiektami oraz jego użyciem. Zmniejsza to ilość napisanego kodu oraz zaoszczędza czas poświęcony na implementację.

7. Interfejs użytkownika

Interfejs użytkownika jest częścią oprogramowania, odpowiadającą za interakcję między użytkownikiem, a systemem. To właśnie od niego zależy czy i w jakim stopniu aplikacja będzie czytelna i funkcjonalna. W czasach, gdzie konkurencja na rynku jest bardzo duża, a dostępność poprzez Internet do innych systemów ułatwiona, użytkownicy nie chcą tracić czasu na mało intuicyjne i nieczytelne w użytkowaniu strony. Dlatego ważne jest, żeby aplikacja była:

- w odpowiedniej kolorystyce,
- prosta i posiadała intuicyjne menu minimalizujące wysiłek poznawczy,
- niezawodna,
- skalowalna,
- estetyczna.

System Gary ze względu na swoją charakterystykę stawia w dużym stopniu nacisk na prostotę i minimalistyczny wygląd, aby w kryzysowej sytuacji nikt nie miał problemu ze znalezieniem odpowiedniej funkcjonalności.

Rozdział o interfejsach użytkownika podzielony jest ze względu na typ aplikacji i autora systemu:

1. Aplikacja mobilna
    - Interfejs dla wszystkich aktorów aplikacji mobilnej,
    - Interfejs dla gościa,
    - Interfejs dla użytkownika,
    - Interfejs dla ratownika medycznego,
2. Aplikacja webowa
    - Interfejs dla wszystkich aktorów aplikacji webowej,
    - Interfejs dla dyspozytora,
    - Interfejs dla kierownika karetki,
    - Interfejs dla użytkownika,
    - Interfejs dla gościa.

7.1. Aplikacja mobilna

Poniższe rozdziały poświęcone są interfejsowi w aplikacji mobilnej, z uwzględnieniem aktorów systemu.

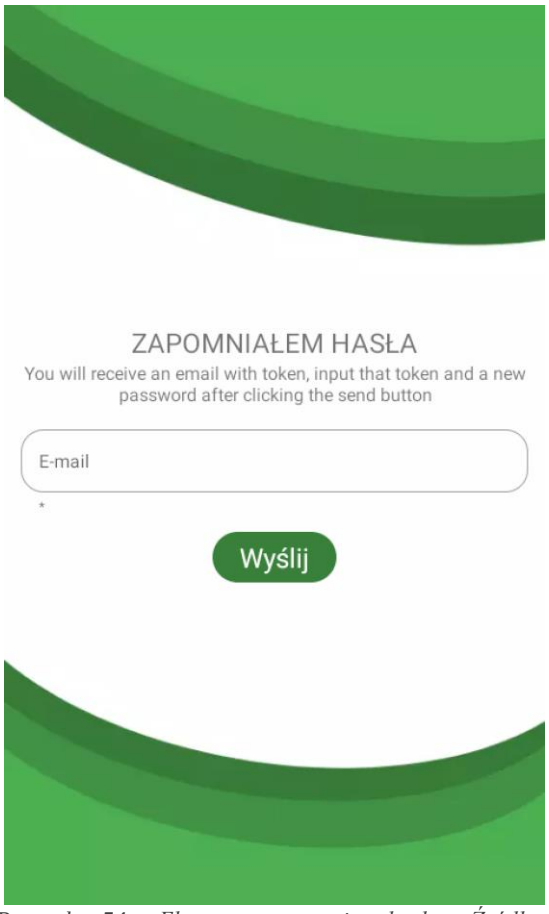
7.1.1. Interfejs dla wszystkich aktorów

Pierwsze widok pojawiające się w aplikacji mobilnej związane są z logowaniem się.

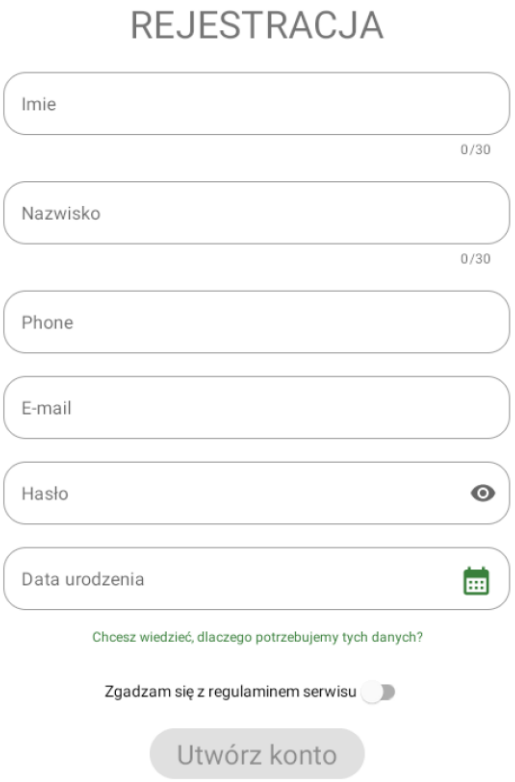
- Ekran początkowy (patrz Rysunek 52)
    - Z tego ekranu użytkownik może zdecydować, czy chce się zalogować, zarejestrować czy może wejść do aplikacji z uprawnieniami gościa.
- Ekran logowania (patrz Rysunek 53)
    - Podając e-mail oraz hasło użytkownik może zalogować się na swoje konto w aplikacji.
- Ekran “Zapomniałem hasła” (patrz Rysunek 54)
    - Podając e-mail związany z kontem użytkownik może zresetować swoje hasło.
- Ekran rejestracji (patrz Rysunek 55)
    - Wypełniając formularz rejestracji i podając wymagane informacje osoba nieposiadająca konta może je założyć.

Rysunek 52. Ekran przed zalogowaniem się do aplikacji mobilnej. Źródło: Opracowanie własne.

Rysunek 53. Ekran logowania się do aplikacji. Źródło: Opracowanie własne.



Rysunek 54. Ekran resetowania hasła. Źródło: Opracowanie własne



Rysunek 55. Ekran formularza rejestracji. Źródło: Opracowanie własne

Jak widać na powyższych rysunkach, motyw jest bardzo prosty i schematyczny. Dzięki temu, liczymy, że żaden użytkownik nie będzie miał problemu z poruszaniem się po aplikacji oraz z wyszukiwaniem tych funkcjonalności, których chce użyć.

7.1.2. Interfejs dla użytkownika

Pierwszym widokiem z jakim możemy się spotkać po zalogowaniu do aplikacji jako użytkownik jest ekran wyświetlający listę poradników. Przedstawiony jest on na rysunku 6.

Aby móc dalej poruszać się po aplikacji, należy kliknąć w znajdującą się w lewym górnym ikonę trzech poziomych kresek – wówczas z lewej strony ukaże się menu w postaci *sidebaru*.

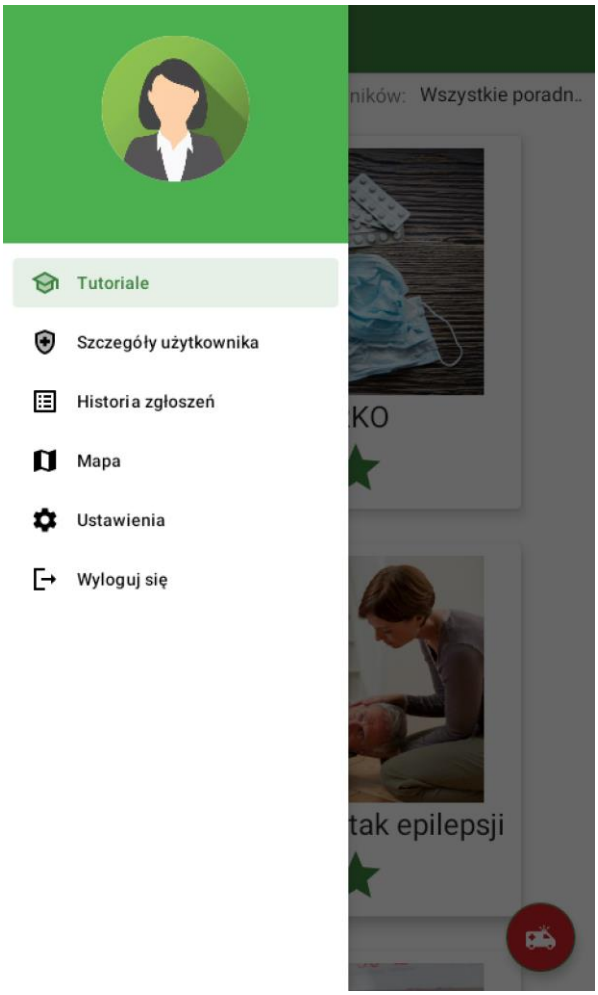
Nawigacja na stronie została podzielona na 6 obszarów - 5 głównych zakładek oraz 1 przycisk do wylogowania. Widok menu przedstawiono na rysunku 56 - jak można zauważyć, w menu jest podświetlana na zielono zakładka, w której obecnie się znajdujemy, co ma na celu pomoc w orientacji użytkownikowi w jakim miejscu się znajduje.

W skład menu wchodzą następujące zakładki:

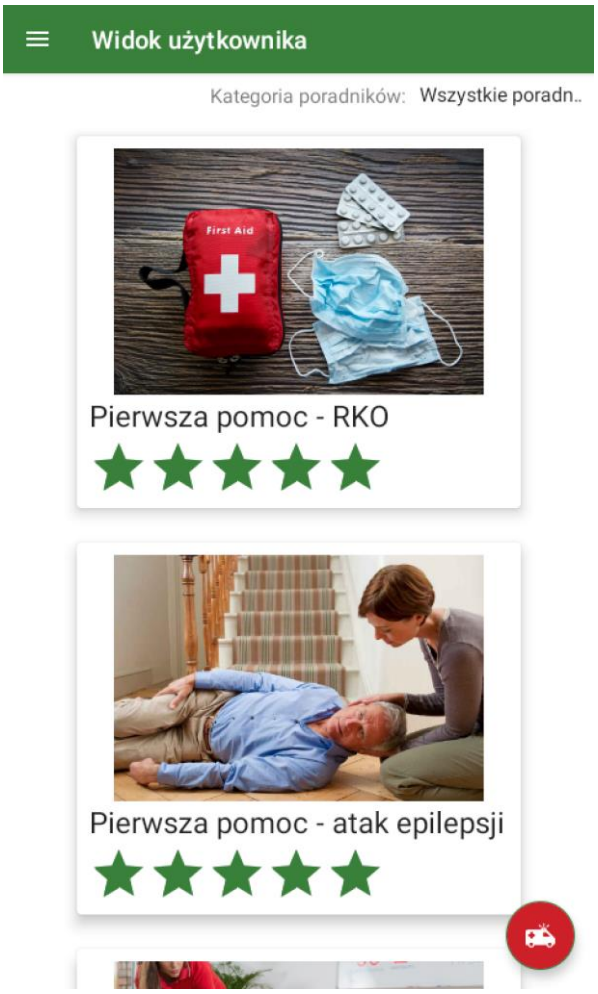
- Poradniki,
- Szczegóły użytkownika,
- Zgłoszenia,
- Mapa,
- Ustawienia,
- Wyloguj się.

Najważniejszą funkcjonalnością aplikacji mobilnej z perspektywy użytkownika jest możliwość utworzenia zgłoszenia. Należy przy tym mieć na uwadze, że funkcja ta będzie najczęściej wykorzystywana w sytuacji silnie stresującej, dlatego istotne jest, aby użytkownik mógł to zrobić możliwie jak najsprawniej. Właśnie z tej przyczyny istnieją dwie ścieżki tworzenia zgłoszenia.

Pierwsza z nich zostanie zaprezentowana na rysunku nr 56. Na ekranie, który widoczny jest od razu po zalogowaniu, w prawym dolnym rogu możemy zauważyć czerwony przycisk z ikoną karetki. Po jego naciśnięciu, przekierowani zostaniemy do formularza tworzenia zgłoszenia.



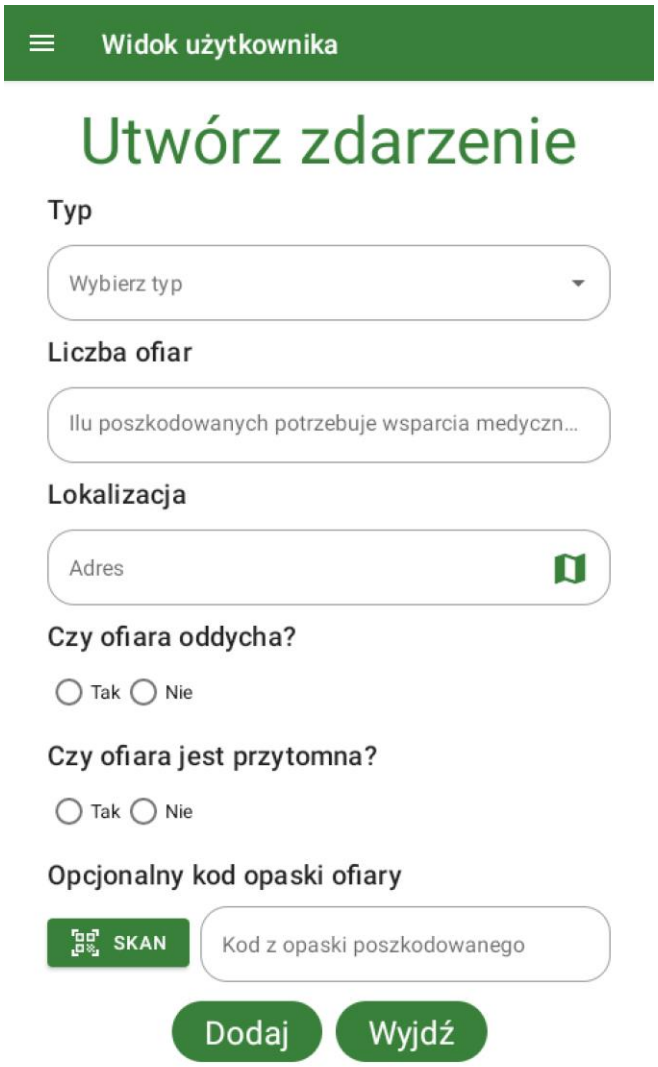
Rysunek 56. Ekran rozwiniętego menu. Źródło: Opracowanie własne.



Rysunek 57. Ekran listy poradników. Źródło: Opracowanie własne.

Druga ścieżka jest bardziej uniwersalna i dostępna z większości miejsc w aplikacji. W menu znajduje się zakładka odpowiedzialna za tworzenie zgłoszeń. Po jej wybraniu, podobnie jak w przypadku naciśnięcia na ikonę karetki, zostanie wyświetlony nam formularz zgłoszeniowy.

Przyglądając się formularzowi zgłoszeniowemu (Rysunek nr 58), zauważyć można jego duże podobieństwo do formularza, który wypełniał użytkownik podczas rejestracji (rysunek nr 55). Ich uniwersalizacja oraz zastosowane elementy mają na celu ułatwienie obsługi aplikacji również osobom, które nie są zaznajomione z technologią. Pomóc również w tym mają dodatkowe pytania lub słowa znajdujące się w poszczególnych polach. Chcąc zminimalizować ilość wpisywanego tekstu przez użytkowników w pierwszym polu zastosowana została lista wyboru, za pomocą której możliwe będzie wybranie typu zgłoszenia, a w polach 4 i 5 do udzielenia odpowiedzi należy użyć pól opcji. Ikona mapy znajdująca się w trzecim polu związanym z lokalizacją ma po kliknięciu w nią otworzyć mapę, gdzie za pomocą pinezki, użytkownik będzie mógł podać swoją lokalizację Ostatnie pole odpowiedzialne za podanie numeru opaski osoby poszkodowanej może zostać wypełnione za pomocą przycisku „Skanuj”, który otwiera aparat, za pomocą którego możliwe będzie odczytanie opaski.

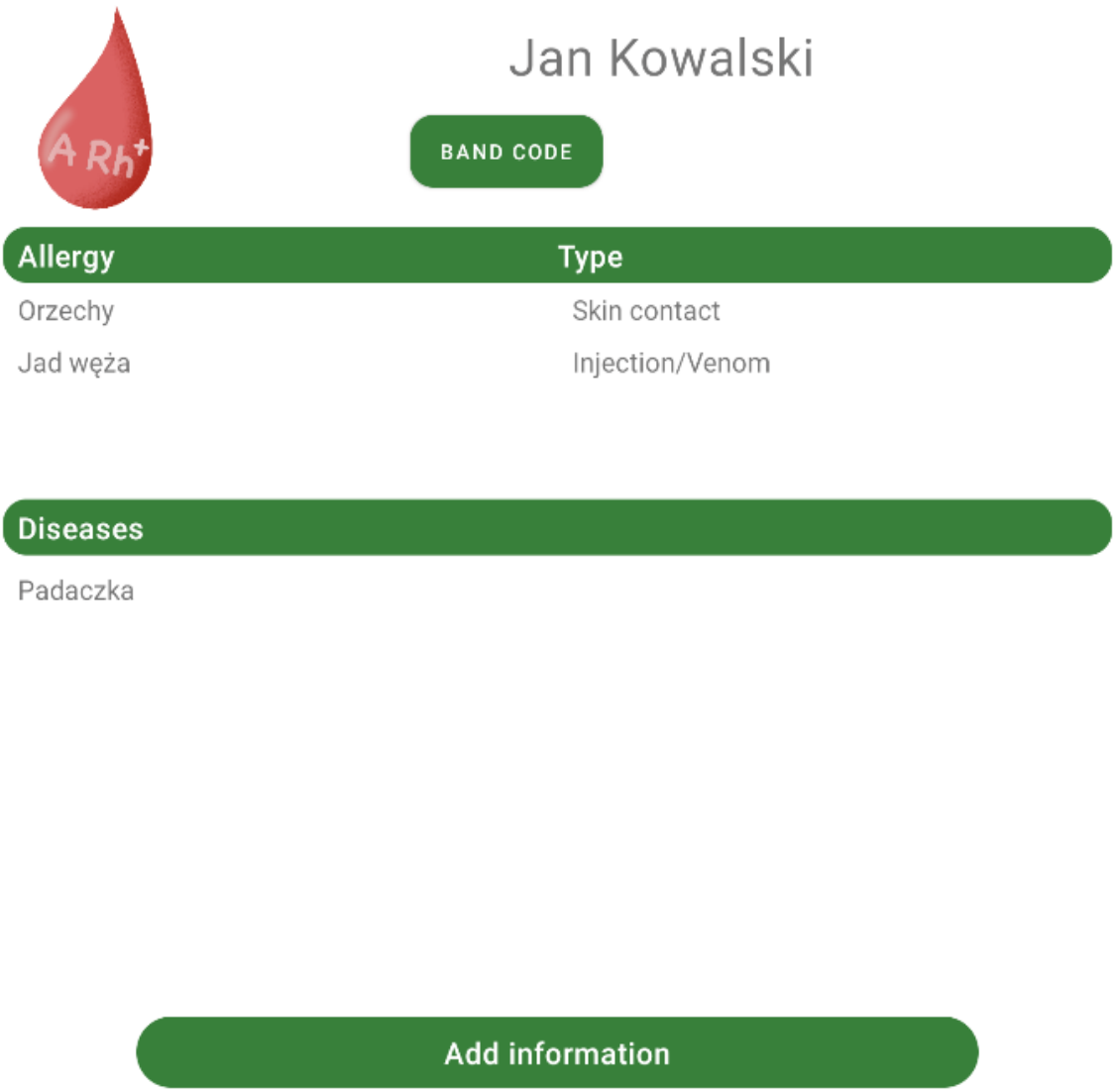


Rysunek 58. Widok formularza zgłoszeniowego. Źródło: Opracowanie własne.

Na rysunku 59 zauważyć można kolejny ważny widok w aplikacji, który uzyskujemy po kliknięciu w zakładkę „Szczegółowe informacje”. Wyświetlane są na nim najważniejsze informacje, takie jak:

- Alergie,
- Choroby,
- Typ krwi.

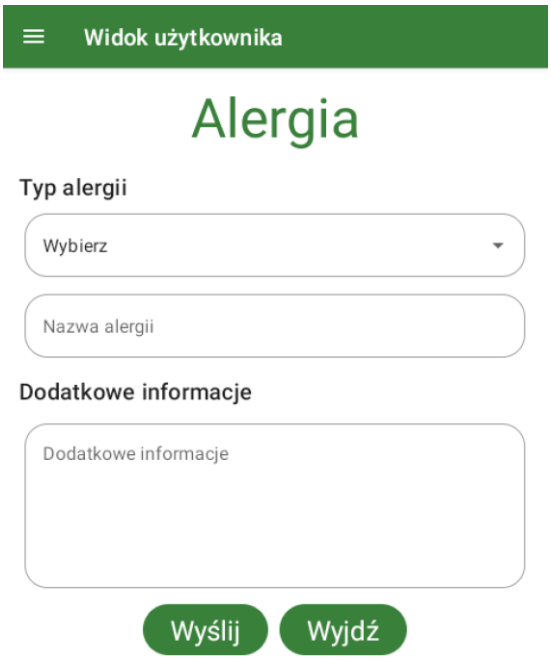
Kropla krwi w lewym górnym rogu informuje nas o typie naszej krwi.



Rysunek 59. Widok szczegółowych informacji. Źródło: Opracowanie własne

Aby dokonać edycji poszczególnych informacji, należy kliknąć przycisk „Dodaj informacje”, a następnie z listy wybrać konkretną już opcję, którą chcemy edytować.

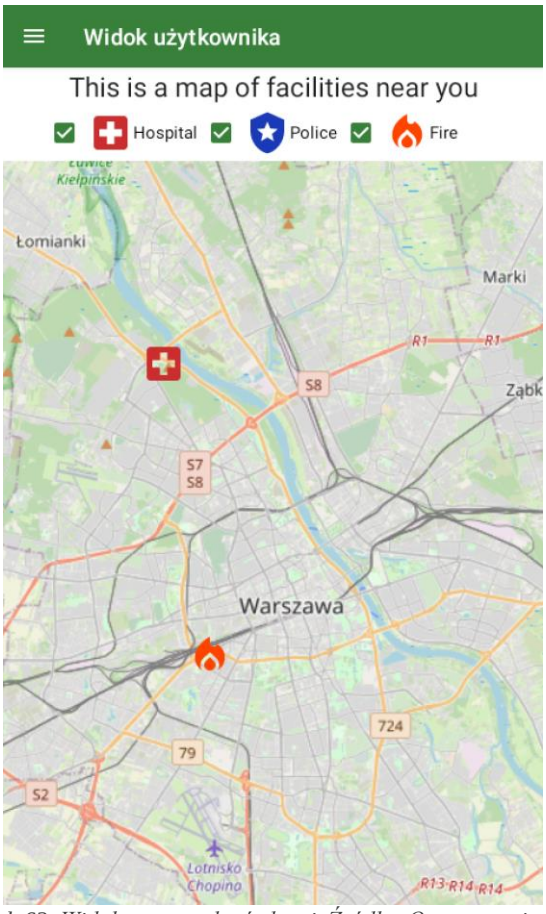
Na rysunkach 60 i 61 przedstawione zostały widoki edycji alergii oraz typu krwi. Również i one zyskały elementy upraszczające takie jak listy wyboru czy pola opcji.



Rysunek 60. Widok dodawania informacji o alergiach do szczegółowych informacji. Źródło: Opracowanie własne

Rysunek 61. Widok dodawania informacji o typie krwi. Źródło: Opracowanie własne

Ważnym elementem w aplikacji są widoki map. Użytkownik za ich pomocą powinien bez problemu móc odnaleźć, gdzie w jego okolicy znajduje się komisariat policji, szpital lub niebezpieczne zdarzenie, którego lepiej unikać. Pola do zaznaczenia znajdujące się na samej górze na rysunku obok poszczególnych ikon, pozwalają na posortowanie wyświetlających się informacji. Rysunek 62 przedstawia mapę z dostępnymi placówkami.



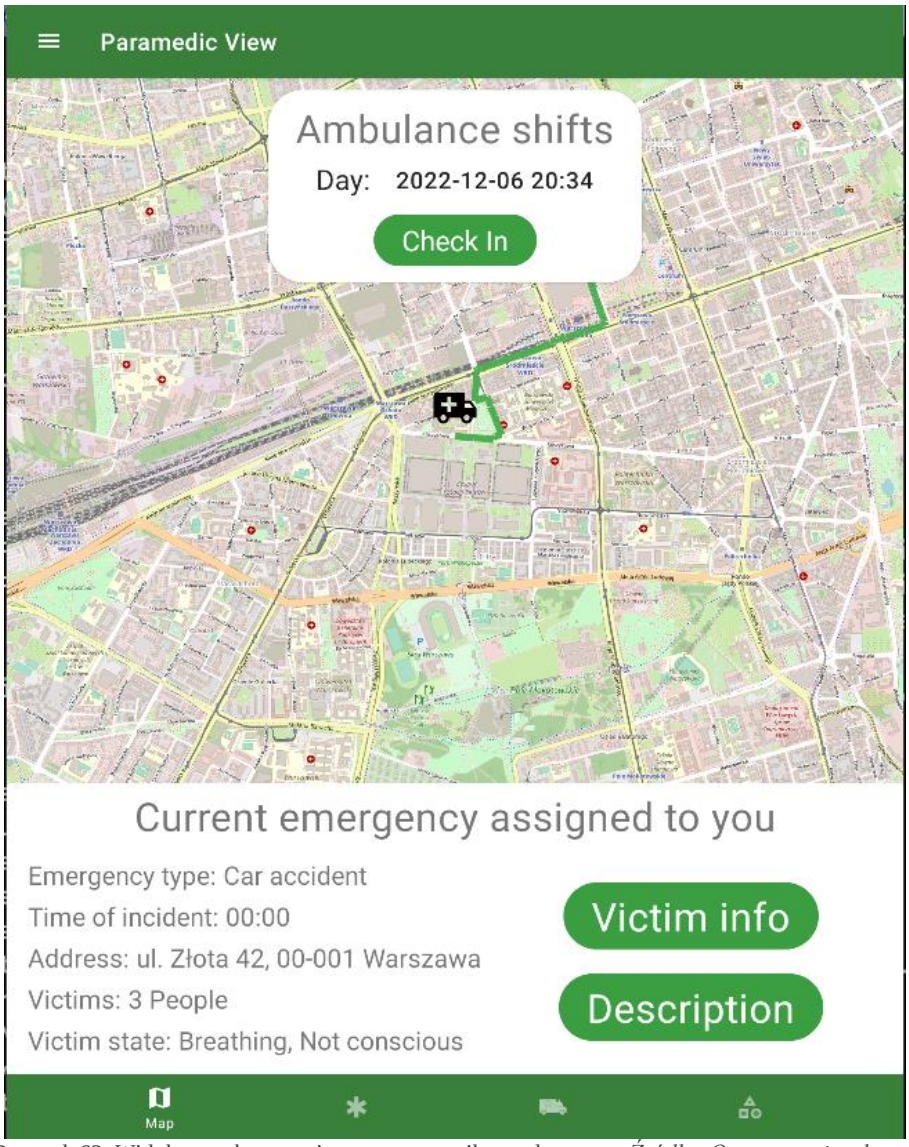
Rysunek 62. Widok mapy z placówkami. Źródło: Opracowanie własne

7.1.3. Interfejs dla gościa

Gość jest aktorem systemu o bardzo ograniczonych funkcjonalnościach. Jedyną funkcjonalnością, jest przeglądanie poradników. Wygląd ten jest taki sam jak dla widoku użytkownika, z różnicą polegającą na tym, że gość nie ma przycisku do tworzenia zgłoszenia, znajdującego się w prawym dolnym rogu.

7.1.4. Interfejs dla ratownika medycznego

Ekran główny widoczny po zalogowaniu się przez ratownika medycznego został pokazany na rysunku 63. Przedstawia on mapę z zaznaczony w centralnym miejscu za pomocą ikony karetki aktualnego położenia karetki. Dodatkowo, po zalogowaniu się do aplikacji, na górze ekranu widnieje komunikat z obecną datą i polem „Zacznij dyżur”, po kliknięciu, w którego dla ratownika rozpocznie się dyżur. Z drugiej strony, na dole, widoczny jest komunikat z informacjami o aktualnym zgłoszeniu. Widoczna na mapie jest również najszybsza możliwa droga do miejsca zdarzenia.



Rysunek 63. Widok po zalogowaniu przez ratownika medycznego. Źródło: Opracowanie własne.

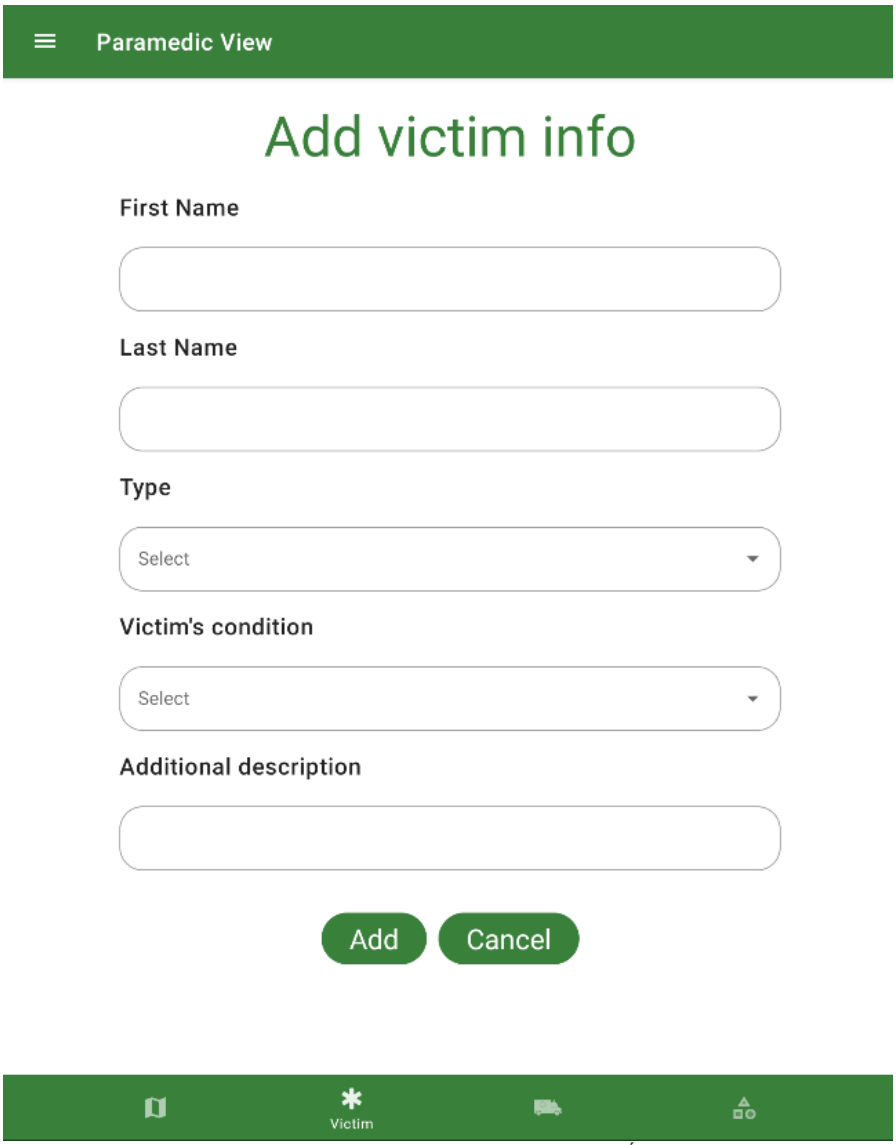
Nawigacja w przypadku ratownika medycznego jest inna niż u zwykłego użytkownika. Spowodowane jest to innym domyślnym rodzajem urządzenia, na którym aplikacja będzie uruchamiana. Dla użytkownika jest to smartfon, natomiast dla ratownika medycznego będzie to tablet. Z tego też powodu nawigacja została podzielona na 4 zakładki znajdujące się na dole ekranu, co można zaobserwować na rysunku 63. Ikony tworzące główne menu przedstawiają kolejno:

- Mapę,
- Poszkodowanych,
- Prośbę o wsparcie,
- Wyposażenie.

W lewym górnym rogu zauważyć jednak można, ikonę na której znajdują się trzy poziome kreski, która w widoku użytkownika rozwijała *sidebar*. Dla ratownika medycznego pełni ona rolę dodatkowego menu, w którym znajdują się mniej ważne zakładki, takie jak:

- Zgłoś przerwę,
- Ustawienia,
- Wyloguj.

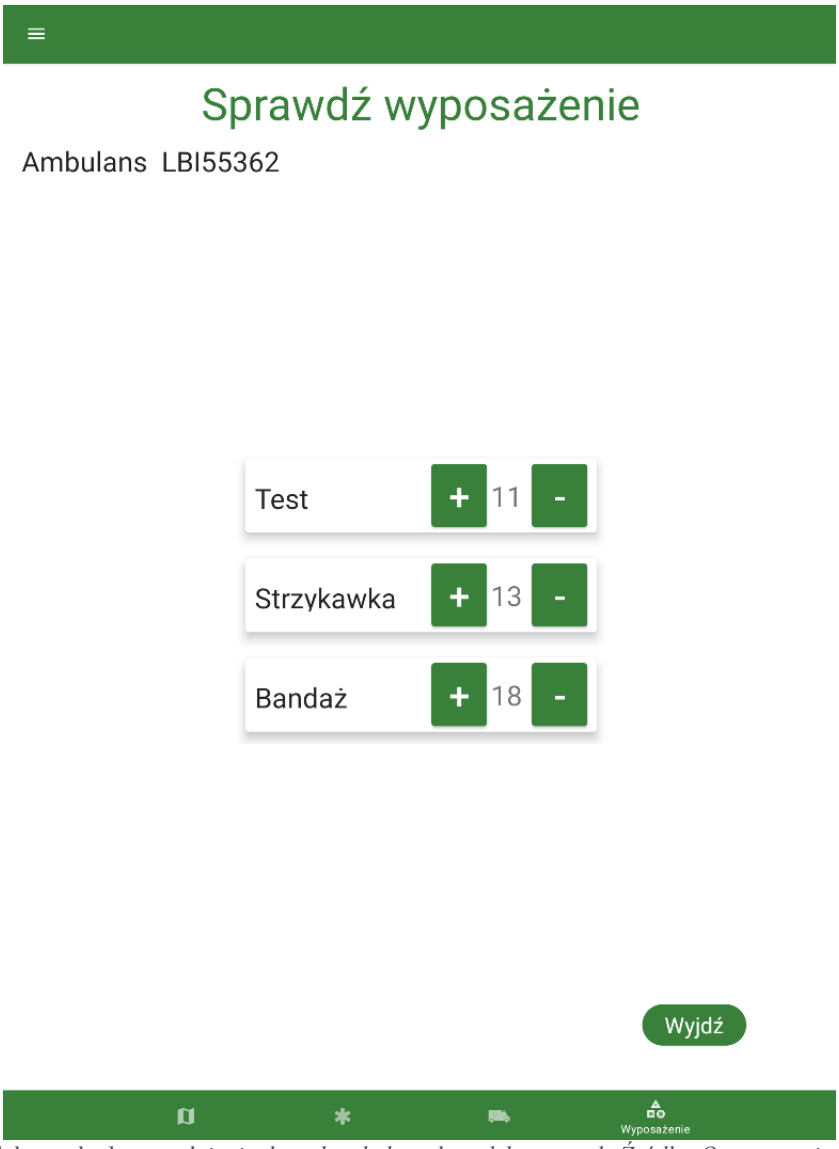
Zadaniem zakładki o poszkodowanych jest umożliwienie dodania informacji o poszkodowanym. Przekazanie tych danych odbywa się poprzez prosty formularz pokazany na rysunku 64.



Rysunek 64. Formularz dodawania informacji o poszkodowanym. Źródło: Opracowanie własne.

Trzecia zakładka jest skorelowana ze zgłoszeniem, przez co jest dostępna tylko i wyłącznie wtedy, gdy ratownik bierze udział w akcji ratunkowej. Za jej pomocą, możliwe jest wysłanie zapytania o dodatkowe wsparcie do dyspozytora. Tam też za pomocą pól wyboru można wybrać rodzaj potrzebnego wsparcia, a pole tekstowe pozwala uzasadnić daną prośbę.

Sprawą nadrzędną jest to, żeby karetka była dobrze wyposażona i żeby nic w niej nie brakowało. Aby ułatwić kontrolę i zarządzanie medykamentami, ratownicy mają do swojej dyspozycji zakładkę, w której wyświetlana jest lista wyposażenia odpowiedniego dla danego typu karetki. Dzięki temu w każdej sytuacji jest możliwość sprawdzenia, jak dużo medykamentów znajduje się w karetce, ile trzeba uzupełnić. Ratownik po każdym zdarzeniu, powinien uzupełnić dane w aplikacji. Przykładowy widok znajduje się w rysunku 65.



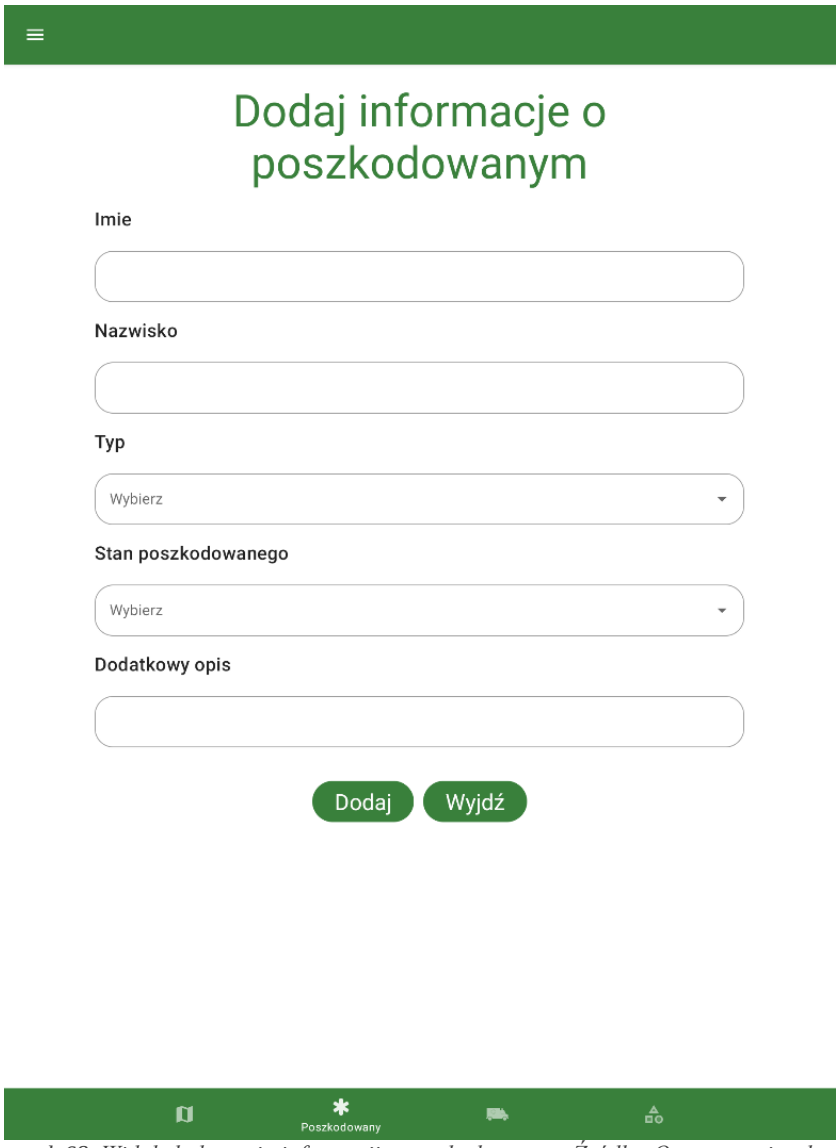
Rysunek 65. Widok panelu do uzupełniania danych o dodanych medykamentach. Źródło: Opracowanie własne

Dla uniknięcia niepotrzebnych nieporozumień z dyspozytorem, ratownik ma możliwość zmiany swojego statusu na „Na przerwie”. Po zgłoszeniu przerwy, zmienia się kolorystyka w całej aplikacji. Zauważyć to można na rysunkach 66 oraz 67, gdzie na pierwszym z nich przerwa nie jest aktywowana. Kolorystyka górnej belki zmienia się w całej aplikacji do czasu wyłączenia przerwy i wznowienia tym samym dyżuru.

Rysunek 66. Ekran rozpoczęcia przerwy. Źródło: Opracowanie własne.

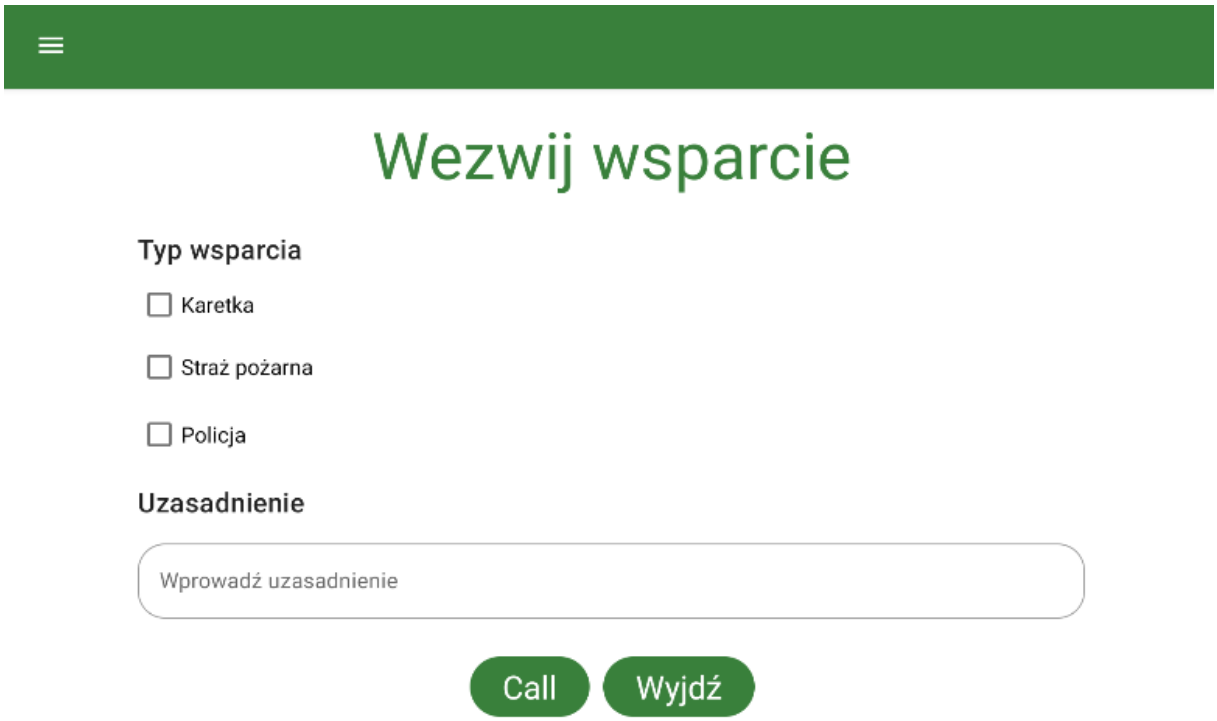
Rysunek 67. Ekran zakończenia przerwy. Źródło: Opracowanie własne

Ratownik medyczny ma możliwość dodania poszkodowanego do bazy, jeśli uda mu się zidentyfikować daną osobę. Opcja ta dostępna jest w drugiej zakładce na dolnym menu. Po jej otwarciu, wyświetla się formularz, za pomocą którego można dodać informacje o poszkodowanym. Jego wypełnianie odbywa się w już tradycyjny dla naszego sytemu sposób. Typy pól zaobserwować można na rysunku 68.



Rysunek 68. Widok dodawania informacji o poszkodowanym. Źródło: Opracowanie własne.

W trzeciej zakładka o nazwie „Wsparcie” ratownik może wysłać prośbę do dyspozytora o przysłanie dodatkowych służb – policja, straż pożarna, czy dodatkowe służby medyczne. Aby wybrać pomoc należy zaznaczyć pole, które odpowiada konkretnemu rodzajowi wsparcia. Wymagane jest również krótkie uzasadnienie, które można wprowadzić w polu poniżej. Cała zakładka pokazana jest na rysunku 69.



Rysunek 69. Widok zakładki wsparcie. Źródło: Opracowanie własne.

7.2. Aplikacja webowa

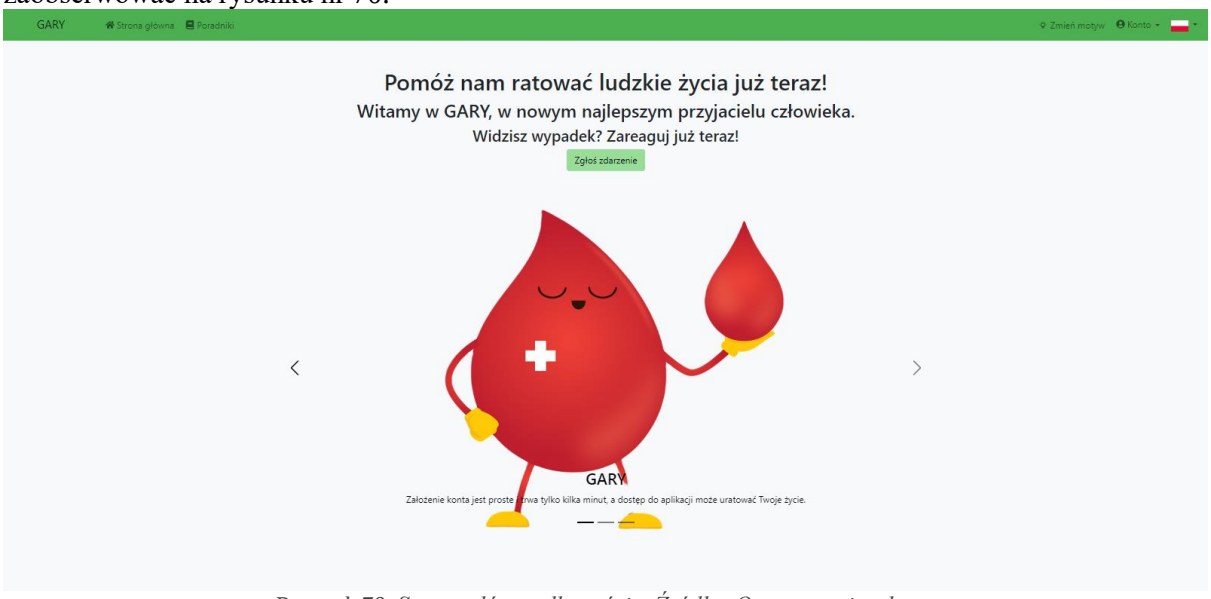
Aplikację webową możemy podzielić ze względu na aktorów, którzy będą ją stosować. Wyróżniamy:

- Użytkownika,
- Gościa,
- Dyspozytora,
- Kierownika karetki.

Poniższe rozdziały szczegółowo przedstawiają interfejs dla aplikacji webowej z uwzględnieniem wcześniej wymienionych aktorów.

7.2.1. Interfejs dla gościa

Okno, które wyświetla się dla gościa, czyli zanim nastąpi zalogowanie do systemu, możemy zaobserwować na rysunku nr 70.



Rysunek 70. Strona główna dla gościa. Źródło: Opracowanie własne.

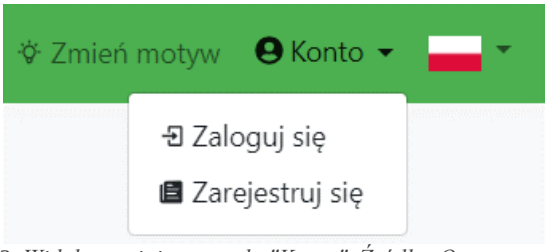
Pierwszą rzeczą, która rzuca się w oczy jest 3 obrazowy *slider* widoczny po środku ekranu. Obrazy na nim zmieniają się samoczynnie po krótkiej chwili, jednakże, osoba przeglądająca stronę może również samemu po nim nawigować za pomocą strzałek znajdujących się na lewo i na prawo od rysunku. Wymienione na *sliderze* informacje opisują główne funkcjonalności użytkownika co ma zachęcić osoby nieposiadające konta w aplikacji do zarejestrowania się.

Rysunek 71. Pasek nawigacji w widoku gościa. Źródło: Opracowanie własne.

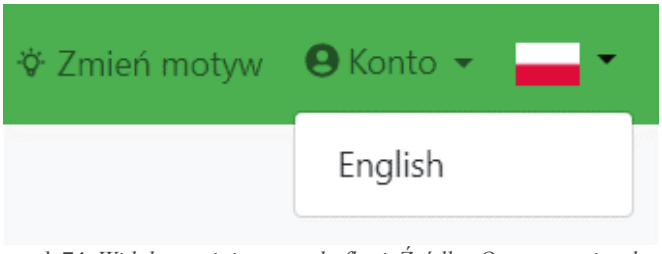
Pasek nawigacji dla osoby niezalogowanej przedstawiony na rysunku 71 zawiera tylko kilka podstawowych zakładek, odpowiadających dostępnym dla gościa funkcjonalnością. Warto w tym miejscu zwrócić uwagę na trzy ostatnie pola; „Zmień motyw”, „Konto” oraz flagę. Pierwszy z nich zmienia motyw strony z jasnego na ciemny i na odwrót. Widok strony głównej w po zmianie motywu można zaobserwować na rysunku 72.

Rysunek 72. Strona główna w ciemnym motywie. Źródło: Opracowanie własne.

Po kliknięciu w pole „Konto" i pole flagi, rozwijają się dodatkowe opcje, które widoczne są poniżej, odpowiednio na rysunkach 73 i 74.



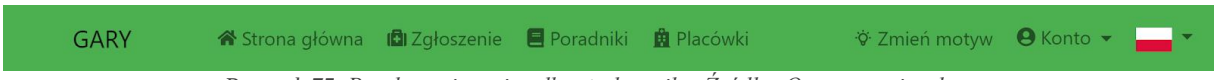
Rysunek 73. Widok rozwiniętego pola "Konto". Źródło: Opracowanie własne.



Rysunek 74. Widok rozwiniętego pola flagi. Źródło: Opracowanie własne.

7.2.2. Interfejs dla użytkownika

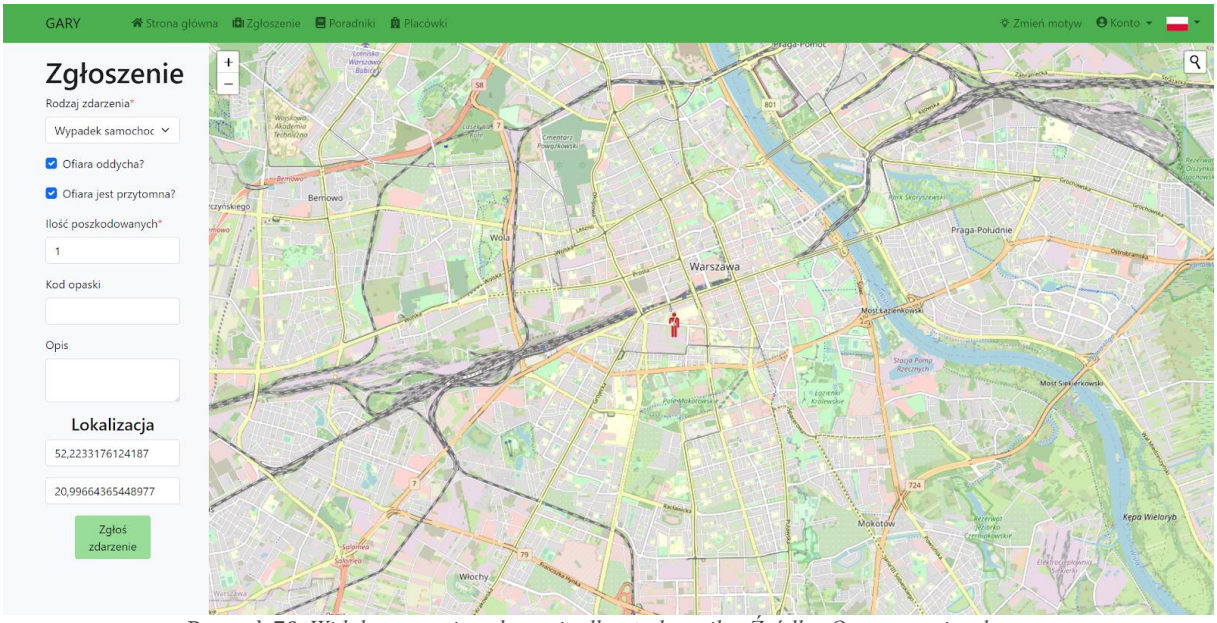
Pasek nawigacji w wyglądzie użytkownika można zaobserwować na rysunku nr 75.



Rysunek 75. Pasek nawigacyjny dla użytkownika. Źródło: Opracowanie własne.

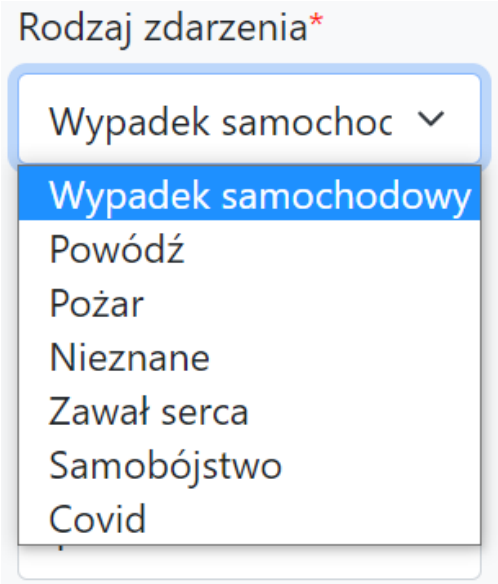
Jak można zauważyć, pola w menu pozostały niezmienione względem aplikacji mobilnej, ze względu na to, że dla użytkownika zarówno aplikacja mobilna jak i aplikacja webowa posiadają ten same funkcjonalności.

Najważniejszą funkcjonalnością systemu jest tworzenie przez użytkownika zgłoszeń. Do tego celu, utworzona została zakładka „Zgłoszenie", po kliknięciu której, wyświetla się ekran, wskazany na rysunku nr 76. Większość widoku zajmuje mapa okolicy, w której się obecnie znajdujemy, natomiast po lewej stronie na ekranie wyświetla się ankieta, w najważniejszymi dla ratownika i dyspozytora informacjami.



Rysunek 76. Widok tworzenia zgłoszenia dla użytkownika. Źródło: Opracowanie własne

Pierwszym polem w formularzu jest rodzaj zdarzenia. Po kliknięciu w pole, rozwinie się lista, z której możemy wybrać czego będzie się tyczyło zgłoszenie. Listę opcji możemy zaobserwować na rysunku nr 77.



Rysunek 77. Dostępne rodzaje zdarzenia przy tworzeniu zgłoszenia. Źródło: Opracowanie własne.

Kolejne dwa pola wyboru odpowiedzialne są za dostarczenie informacji o stanie funkcji życiowych poszkodowanego. Następne trzy pola tekstowe odpowiadają za przekazanie informacji o wypadku. Pierwsze z nich odpowiada za uzupełnienie informacji, ma status informacji obowiązkowej, ponieważ powiadamia o liczbie osób poszkodowanych.

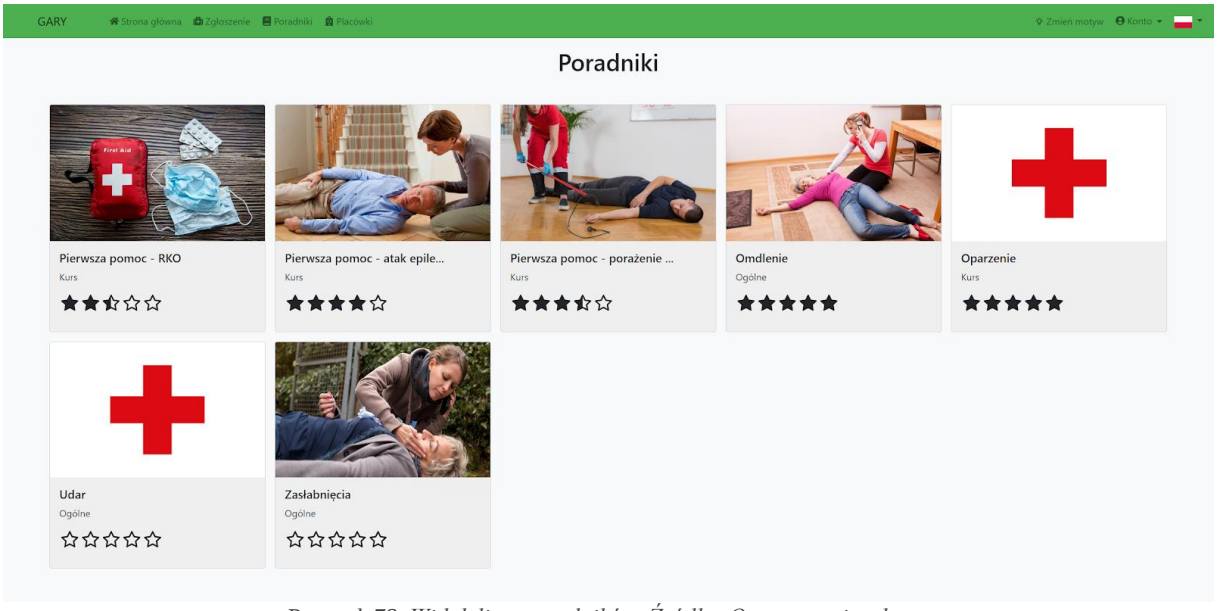
Dwa pola umieszczone pod nim mają dostarczyć informacji nieobligatoryjnej - odpowiadają za ewentualne przekazanie numeru opaski poszkodowanego oraz krótki opis zdarzenia. W dolnej części opisywanego fragmentu, widać ostatnie dwa pola - nie podlegają one opcji modyfikacji, ponieważ odpowiadają za udostępnienie lokalizacji. Wartości w nich zmieniają się automatycznie w momencie zaznaczenia na mapie miejsca znaczenia. Dla usprawnienia procesu wyszukiwania konkretnych miejsc, zastosowano dwa nawigatory - lupę oraz plus i minus.

Znajdująca się w prawym górnym rogu lupa, umożliwia zaznaczenie na mapie wyszukiwanego miejsca. W celu wyszukania miejsca tym sposobem, należy kliknąć lupę, a następnie wypełnić pola wyszukiwarki - na mapie zaznaczone zostanie szukane przez nas miejsce. Ta opcja ma na celu ułatwienie lokalizowania rejonu bądź konkretnego miejsca.

Druga opcja ułatwiająca nawigację to plus i minus znajdujący się w lewym górnym rogu. Za ich pomocą, możemy powiększać oraz zwiększać szczegółowość mapy. Plus służy zwiększeniu dokładności mapy, minus natomiast pomniejszeniu.

Po kliknięciu w wybranego miejsca, pojawia się ikona, której wygląd jest uzależniony od konkretnego typu zdarzenia – w tym wypadku, jako że wybrany został wypadek samochodowy, ikona jest czerwona i przedstawia człowieka.

Kolejna dostępna zakładka jest odpowiedzialna za bardziej edukacyjną część aplikacji, a mianowicie poradniki. Po wejściu w zakładkę widoczny jest ekran przedstawiony na rysunku 78.

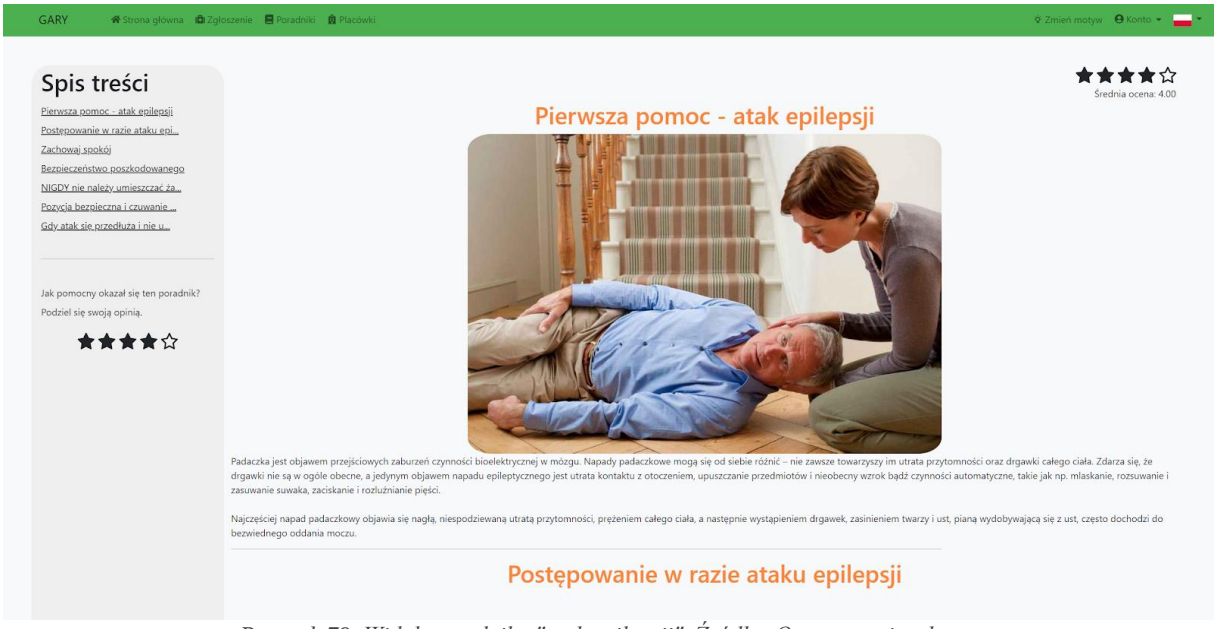


Rysunek 78. Widok listy poradników. Źródło: Opracowanie własne

Pod każdym poradnikiem, widoczny jest tytuł, pod którym znajduje się średnia ocena wszystkich użytkowników.

Na rysunku 79, można zaobserwować widok przykładowego tutorialu – w tym wypadku mówiącego jak radzić sobie w przypadku ataku epilepsji.

Z lewej stronie widoczny jest spis treści podrozdziałów poradnika, po kliknięciu w który widok przesunie się na daną część poradnika. Pod spisem, zauważyć można 5 gwiazd Jest to miejsce, w którym możemy dokonać oceny obejrzanego przez nas poradnika w skali od 1 do 5, gdzie 5 jest oceną najwyższą. Po dokonaniu oceny i odświeżeniu strony, średnia tutorialu, która znajduje się w prawym górnym rogu, zostanie zaktualizowana.

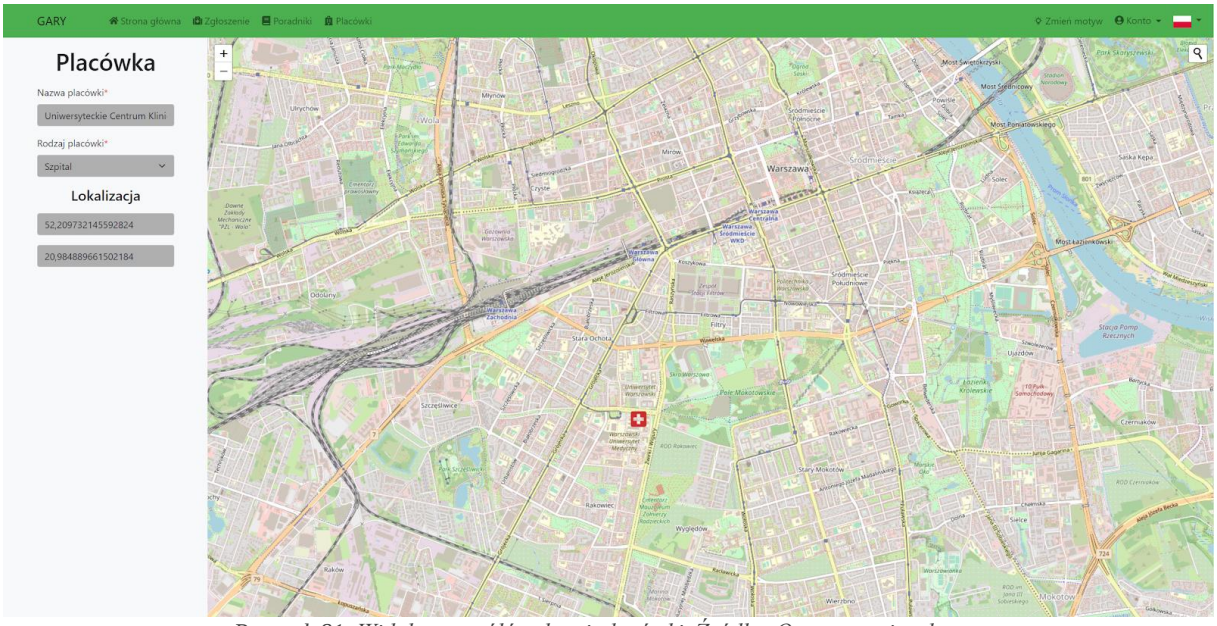


Rysunek 79. Widok poradnika "atak epilepsji". Źródło: Opracowanie własne

Zakładka „Placówki”, odpowiedzialna jest za część informacyjną. Za jej pomocą, można zobaczyć listę placówek ukazaną za pomocą tabeli, w której zawarte są informacje oraz lokalizacje: komisariatów policji, szpitali, jednostek straż pożarnych, punktów AED; znajdującą się w bazie. Widok tabeli zaprezentowany jest na rysunku 80. W celu ułatwienia wyszukiwania interesujących nas placówek, do dyspozycji użytkownika są narzędzia sortujące i filtrujące, znajdujące się powyżej każdej z kategorii.

Rysunek 80. Widok listy placówek. Źródło: Opracowanie własne.

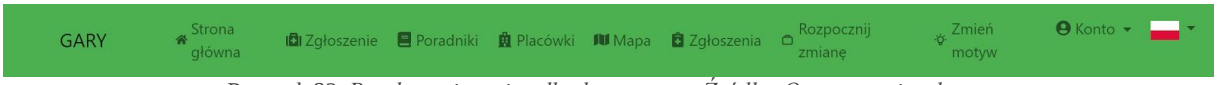
Użytkownik ma dodatkowo możliwość zobaczenia, gdzie na mapie znajduje się dana placówka. Aby tego dokonać, należy kliknąć w numer identyfikacyjny danego miejsca. Nastąpi następnie przekierowanie, na mapę, na której za pomocą odpowiedniej ikony przedstawiona zostanie placówka. Przykładowy widok zaprezentowany jest na rysunku 81. Na mapie szpital został pokazany za pomocą ikony krzyża.



Rysunek 81. Widok szczegółów danej placówki. Źródło: Opracowanie własne.

7.2.3. Interfejs dla dyspozytora

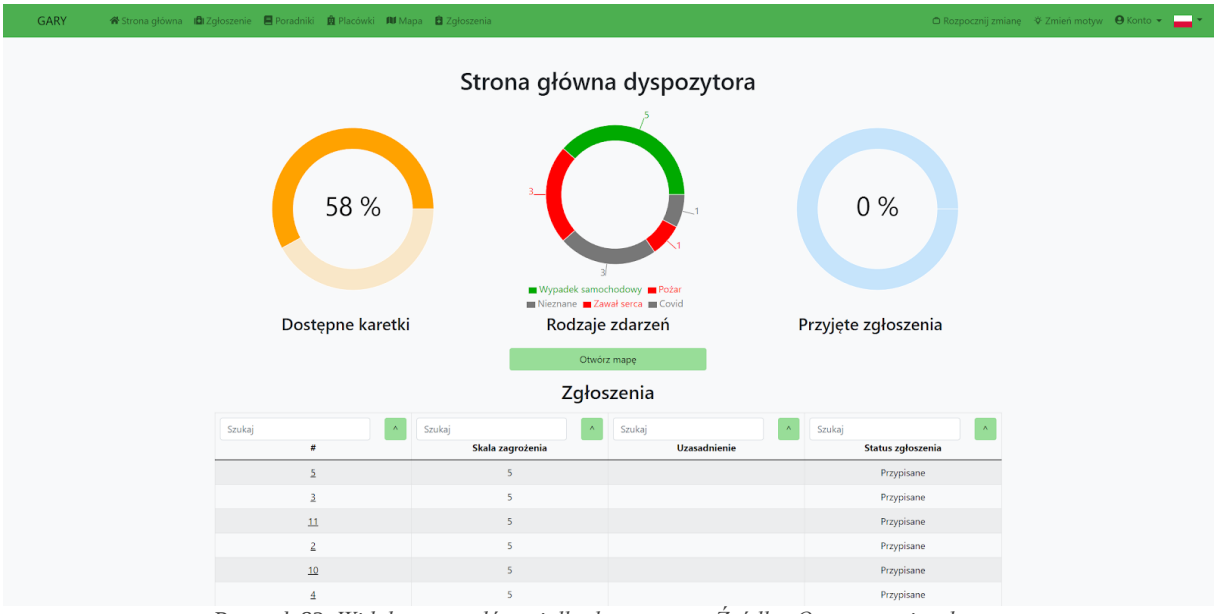
Pasek nawigacji w widoku dyspozytora został przedstawiony na rysunku 82.



Rysunek 82. Pasek nawigacyjny dla dyspozytora. Źródło: Opracowanie własne.

Dyspozytor po zalogowaniu się, ma wgląd w statystyki obecne oraz z całego dnia. Informacje te zostały przekazane za pomocą kolorowych wykresów kołowych. Głównym celem tego zabiegu jest ułatwienie interpretacji, tak, aby już na pierwszy rzut oka było widać, czy do kolejnych zgłoszeń czekają jakieś karetki, czy może wszystkie są obecnie wykorzystywane.
Ekran, o którym mowa, został zaprezentowany na rysunku 83.

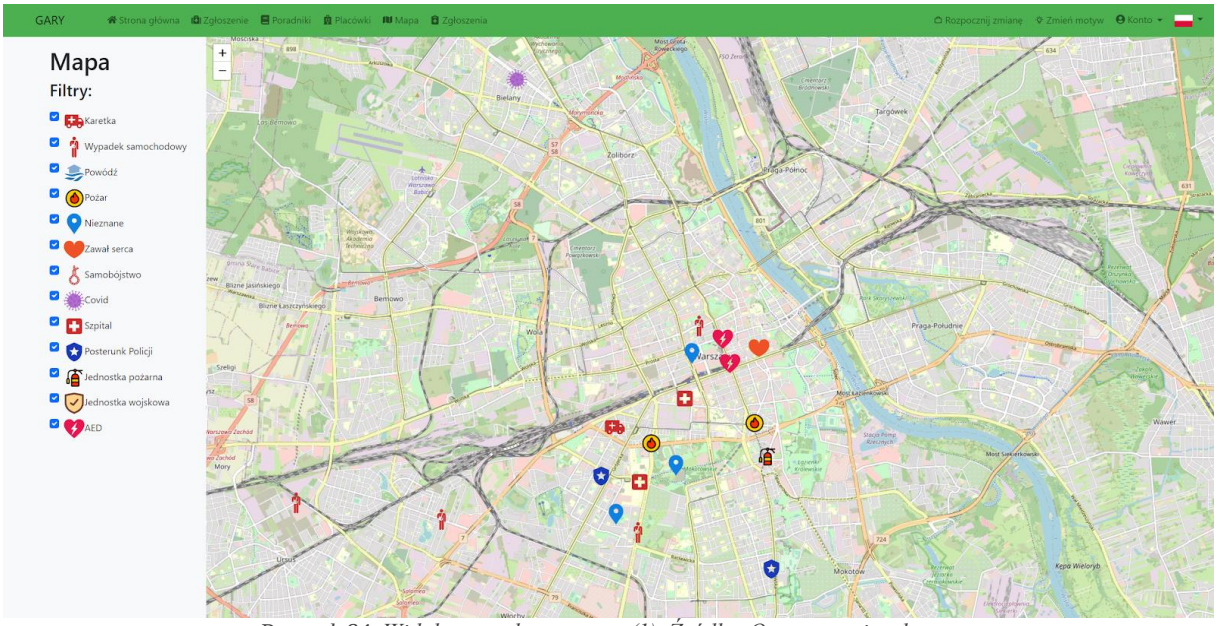
Ważnym aspektem w przypadku pracy dyspozytora, jest przycisk „Rozpocznij zmianę". Bez jego kliknięcia, nie jest w systemie odnotowywane to, że dyspozytor zaczął swój dyżur.



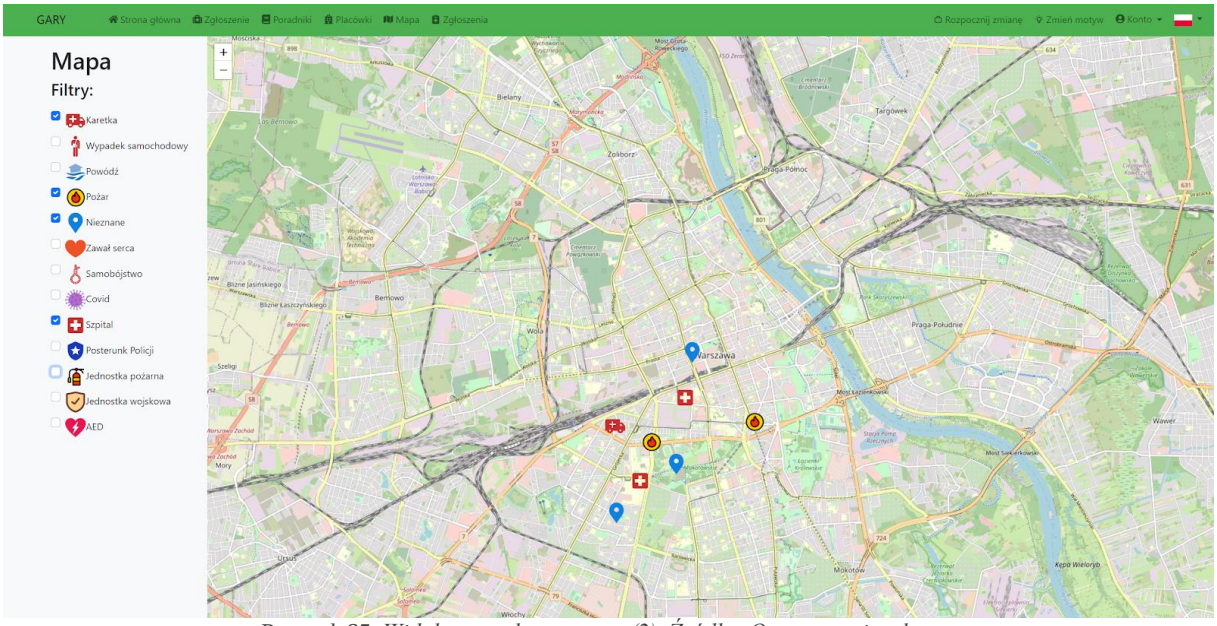
Rysunek 83. Widok strony głównej dla dyspozytora. Źródło: Opracowanie własne.

Poniżej wykresów kołowych, zauważyć można tabelę o tytule zgłoszenia. Jej celem jest podsumowanie zgłoszeń, za które dyspozytor jest w danej chwili odpowiedzialny. Do lepszego zarządzania zgłoszeniami powstały narzędzia do filtrowania oraz do wyszukiwania. Sortować można po czterech kategoriach: kolejność zgłoszenia – w tym wypadku jest to kolumna z numerami zgłoszeń, stopień zagrożenia, uzasadnienia i po statusie zgłoszenia. Aby wyświetlić informację lub zacząć obsługiwać konkretne zgłoszenie, należy kliknąć w odpowiadający zgłoszeniowi numer.

Dyspozytor powinien wiedzieć, co dzieje się na zarządzanym przez niego terenie. Abu mu to umożliwić, powstała zakładka „Mapa”. Jej celem, jest ukazanie lokalizacji wszystkich zgłoszeń, placówek oraz karetek na mapie. Po kliknięciu w daną ikonę możliwy jest podgląd dodatkowych informacji na temat obiektu. Aby ułatwić posługiwanie się mapą, stworzony został specjalny filtr z legendą widoczny na rysunku 84. Za jego pomocą można zmniejszyć bądź zwiększyć ilość pokazywanych lokalizacji. Przykład użycia filtrów został zaprezentowany na rysunkach 84 i 85.

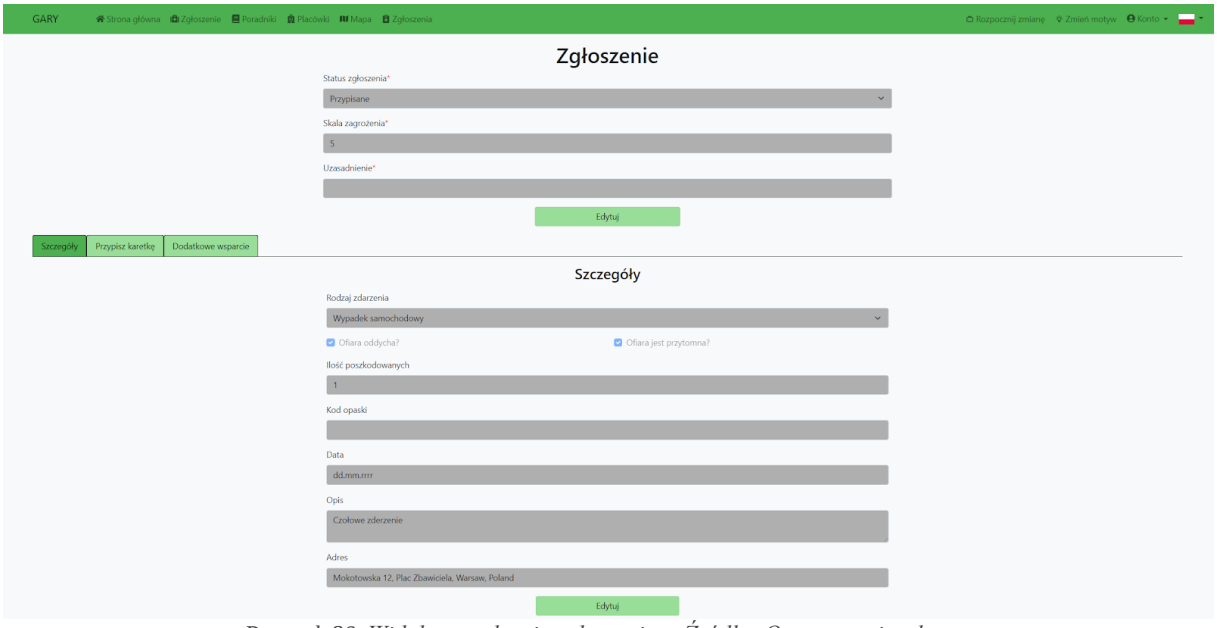


Rysunek 84. Widok mapy dyspozytora (1). Źródło: Opracowanie własne.



Rysunek 85. Widok mapy dyspozytora (2). Źródło: Opracowanie własne.

Najważniejszą funkcjonalnością dyspozytora jest obsługa zgłoszeń i przypisywanie do nich karetek. Odpowiedzialna jest za to zakładka „Zgłoszenia”. Znajdująca się tam lista posiada te same funkcjonalności, co tabela widoczna na stronie głównej. Aby zarządzać zgłoszeniem, należy kliknąć w numer odpowiadający zgłoszeniu. W następstwie tego, wyświetlony zostanie ekran zarządzania zgłoszeniem przedstawiony na rysunku 86.

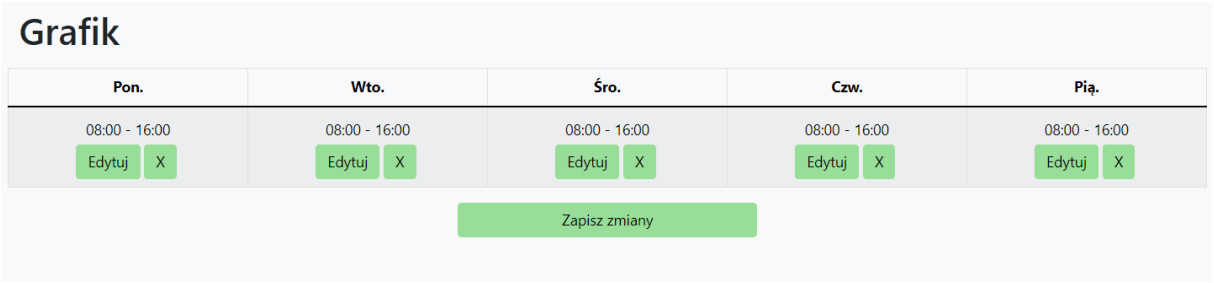


Rysunek 86. Widok zarządzania zgłoszeniem. Źródło: Opracowanie własne.

W tym miejscu, można edytować wszystkie przedstawione na rysunku 86 informacje. Aby to zrobić należy w odpowiednim miejscu nacisnąć przycisk „Edytuj”. Gdy wszystkie wymagane pola zostaną wypełnione, można przystąpić do przypisania do zgłoszenia karetki. Na początku należy wybrać zakładkę „Przypisz karetkę”. W efekcie czego wyświetlony zostanie ekran z dostępnymi karetkami, widoczny na rysunku 87. Następnie z listy dostępnych karetek należy wybrać te, które będą odpowiednie dla zdarzenia. W celu uzyskania większej ilości informacji o karetce, można kliknąć w numer rejestracyjny karetki. Zostaną wyświetlone wtedy dodatkowe informacje.

Rysunek 87. Widok listy dostępnych karetek. Źródło: Opracowanie własne.

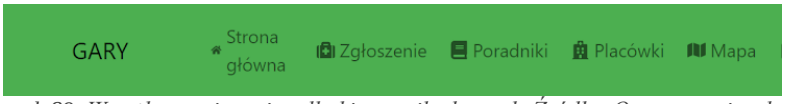
Ostatnią funkcją systemu dyspozytora jest Grafik. Aby sprawdzić grafik pracowników, należy wejść w zakładkę „Konto”, a następnie w interesującą nas opcję. Zawartość widoku grafiku widoczna jest na zdjęciu 88.



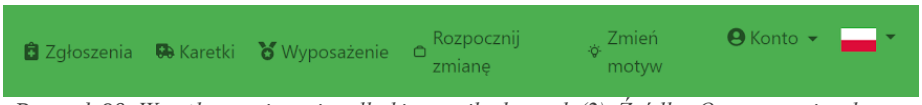
Rysunek 88. Widok grafiku dla dyspozytora. Źródło: Opracowanie własne.

7.2.4. Interfejs dla kierownika karetek

Na rysunkach 89 oraz 90, przedstawiony został widok paska nawigacyjnego dla kierownika karetki. Pasek podzielono na dwa rysunki w celu lepszej czytelności.



Rysunek 89. Wstążka nawigacyjna dla kierownika karetek. Źródło: Opracowanie własne.

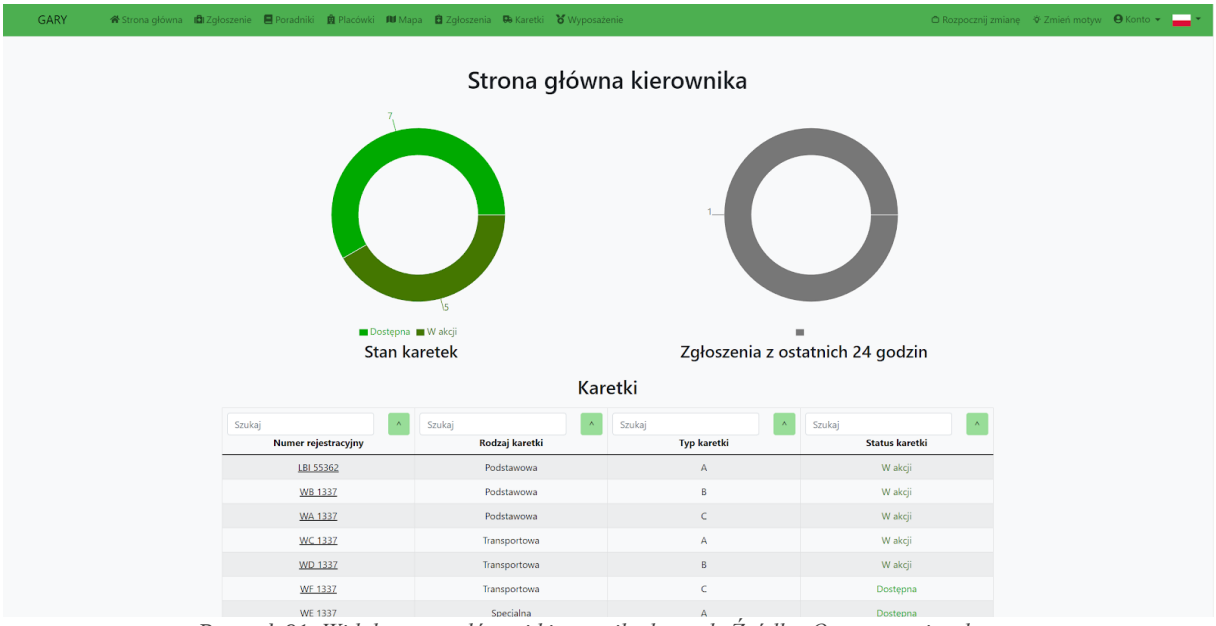


Rysunek 90. Wstążka nawigacyjna dla kierownika karetek (2). Źródło: Opracowanie własne.

W poprzednich podrozdziałach rozdziału o interfejsie webowym, opisane zostały zakładki „Zgłoszenie", „Poradniki", „Placówki" „Mapa" oraz „Zgłoszenia".

Tak jak w przypadku dyspozytora, bardzo ważne jest to, aby na początku dyżuru kliknąć w przycisk „Rozpocznij zmianę". Bez tego, praca nie zostanie zarejestrowana w systemie".

Ekran strony głównej, widoczny po zalogowaniu się na konto z uprawnieniami kierownika karetki, przedstawiony jest na rysunku 91.

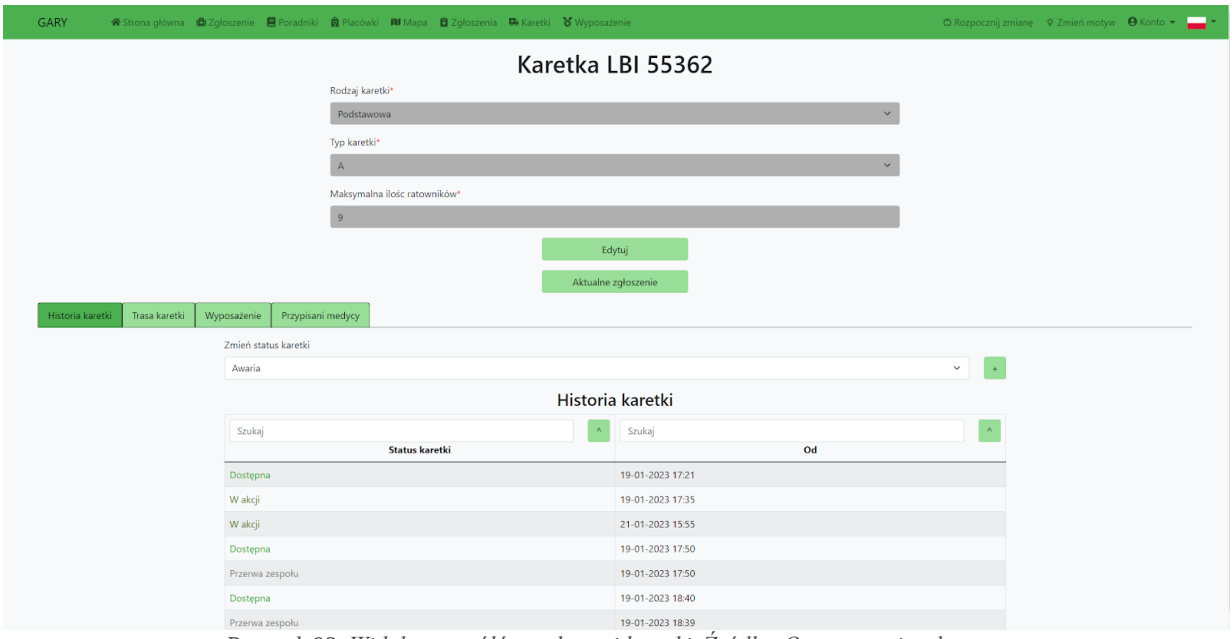


Rysunek 91. Widok strony głównej kierownika karetek. Źródło: Opracowanie własne.

Na samym środku widoczne są dwa wykresy kołowe, przedstawiające informacje bieżące oraz z całego dnia dotyczące karetek i zgłoszeń.

Poniżej, możemy zauważyć, listę karetek, którymi zarządza kierownik. Aby ułatwić nadzorowanie, zastosowane zostały filtry i wyszukiwarki, za pomocą których, łatwiej znaleźć szukany pojazd.

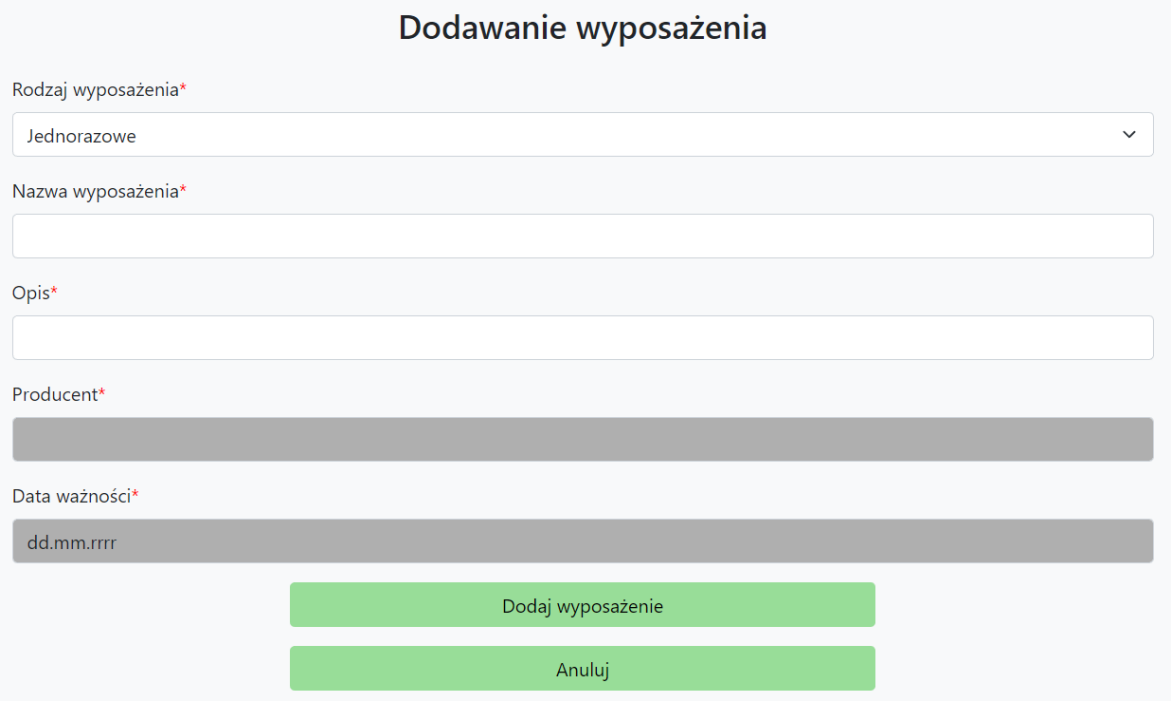
Aby sprawdzić szczegóły, historię lub edytować informacje odnośnie karetek, należy kliknąć w wybrany numer rejestracyjny. Następnie, otworzone zostanie okno, które zaprezentowane zostało na rysunku 92.



Rysunek 92. Widok szczegółów wybranej karetki. Źródło: Opracowanie własne.

Jak można zauważyć, na górze ekranu widoczne są parametry techniczne karetki, które możemy w każdej chwili zmodyfikować. Po wybraniu przycisku „Aktualne zgłoszenie" zostaniemy przekierowani do wglądu w informacje o zdarzeniu, w którym bierze udział dana karetka. Poniżej, widoczna jest tabela z historią karetki, która tak jak inne tabele, posiada możliwości sortowania i filtrowania, w celu szybszego znalezienia interesujących nas informacji.

Ważną funkcjonalnością systemu jest możliwość dodawania nowych rodzajów medykamentów do systemu. Aby móc tego dokonać, należy wejść w zakładkę „Wyposażenie", znajdującą się na głównym pasku nawigacji. Dzięki niej wyświetlona zostanie tabela, w której znajdują się wszystkie obecne w systemie medykamenty. Aby dodać nowe wyposażenie, należy kliknąć przycisk z plusem znajdujący się w prawym górnym rogu nad tabelą z lekami. W efekcie czego, wyświetlany zostanie formularz, po wypełnieniu którego, wyposażenie trafi do systemu. Formularz widoczny jest na rysunku 93.



Rysunek 93. Widok dodawanie wyposażenia. Źródło: Opracowanie własne

8. Testowanie aplikacji

Testowanie stanowi ważny element przy tworzeniu infrastruktury informatycznej. Pozwala na tworzenie kodu wysokiej jakości, zminimalizowanie ilości możliwych błędów w przyszłym użytkowaniu oprogramowania oraz zapewnia postępy w tworzeniu aplikacji. Testy rozpoczyna się już przy implementacji pierwszych funkcjonalności systemu. Dzięki temu na dalszych etapach tworzenia aplikacji jesteśmy w stanie uniknąć poważnych błędów, a co za tym idzie stworzenia wadliwego oprogramowania.

Kod opisanej aplikacji jest przechowywany na Githubie, czyli hostingowym serwisie internetowym przeznaczony do projektów programistycznych opartym na systemie kontroli wersji Git. Jest on podzielony na 3 repozytoria: *backend*, *frontend* i aplikacja mobilna.

Testowanie aplikacji rozpoczynamy od pobrania najbardziej aktualnej wersji aplikacji poprzez system kontroli wersji Git. Następnie jest ona uruchamiania lokalnie na środowisku programistycznym – IntelliJ IDEA.

Część kliencka jaką jest m.in. aplikacja webowa również jest budowana i uruchamiana lokalnie na odpowiednim środowisku programistycznym – Visual Studio Code. Aplikacja jest dostępna do testów na lokalnym hoście.

Aby przetestować aplikację mobilną, zespół odpowiedzialny za jej implementację generuje plik APK i udostępnia go testerom. Aplikacja jest uruchamiana na Android Studio - środowisku programistycznym przeznaczonym dla Androida. Wbudowany emulator oraz fizyczne urządzenia z włączoną opcją debbugowania pozwalają na przeprowadzenie testów manualnych.

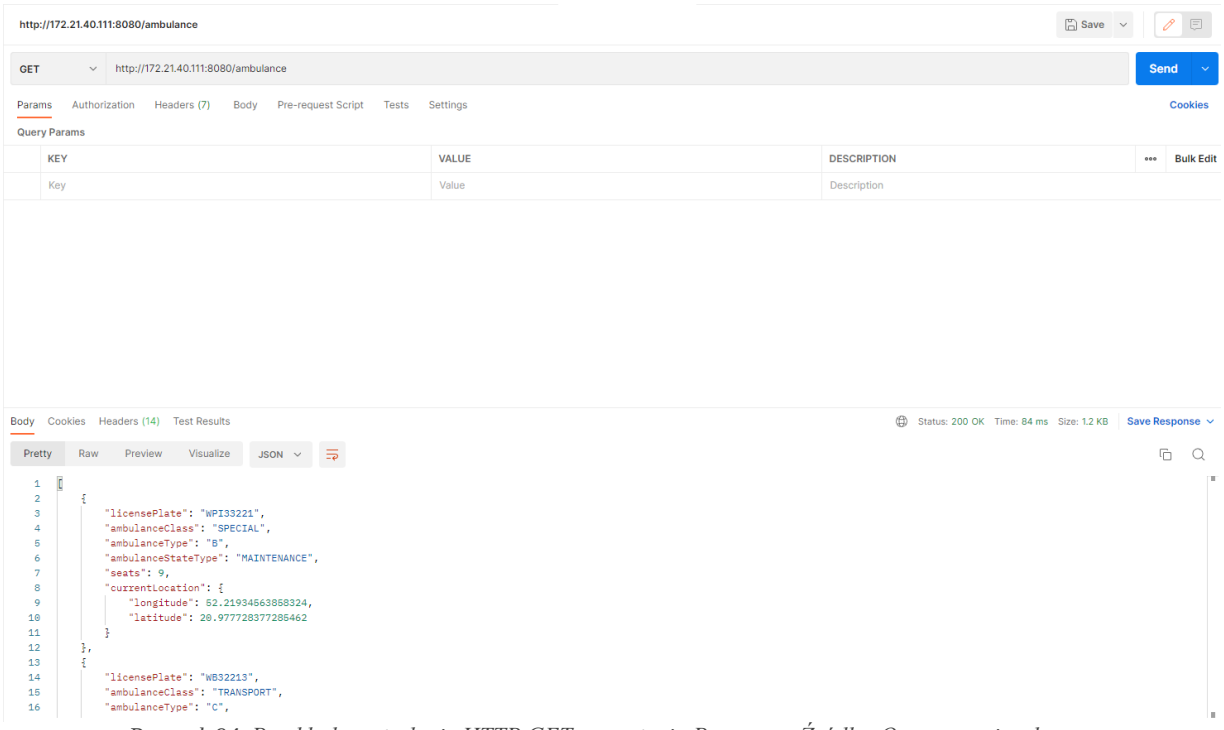
Na potrzebę testów grupa analizy stworzyła scenariusze z przypadkami użycia dla naszej aplikacji. Składają się one ze wszystkich najważniejszych funkcjonalności jakie są przypisane dla poszczególnych typów użytkowników. Ponadto zostało uwzględnione pożądane działanie aplikacji na skutek prawidłowych działań użytkownika, jak również niepożądane sytuacje spowodowane błędem użytkownika np. we wprowadzaniu danych.

Zaistniałe błędy, które zostały wyłonione podczas testów są zgłaszane na Githubie w sekcji *Issues*. Każdy użytkownik ma możliwość w takim zgłoszeniu (*Issue*) odwołać się do konkretnego miejsca w kodzie, opisać zaistniały problem, załączyć zdjęcia czy filmy oraz ustawić priorytet zgłoszenia. Śledzenie błędów ma na celu uprawnienie projektu i dokumentowanie wgrywania poprawek. Kolejne rozdziały opisują szczegółowo przeprowadzone w ramach projektu testy oraz ich rodzaje.

8.1. Testy manualne i integracyjne

W celu przeprowadzenia testów API został użyty program Postman. Narzędzie jest rozwijane od ponad 10 lat, cieszy się największą popularnością na rynku oprogramowania API. Aplikacja jest rozwijana przez wielu twórców, którzy odpowiadają na sugestie użytkowników, dodając nowe funkcjonalności. Postman to bardzo potężne narzędzie, które poprawia wygodę testowania oprogramowania. Służy głównie do wysyłania żądań http, pozwala na tworzenie kolekcji i folderów z zapytaniami, które usprawniają pracę ułatwiając współpracę z innymi.

Na rysunku 94 został przedstawiony przykładowy test żądań http z wykorzystaniem tej aplikacji. W tym celu został użyty getter getAmbulance, który w odpowiedzi zwraca nam informacje o karetkach.

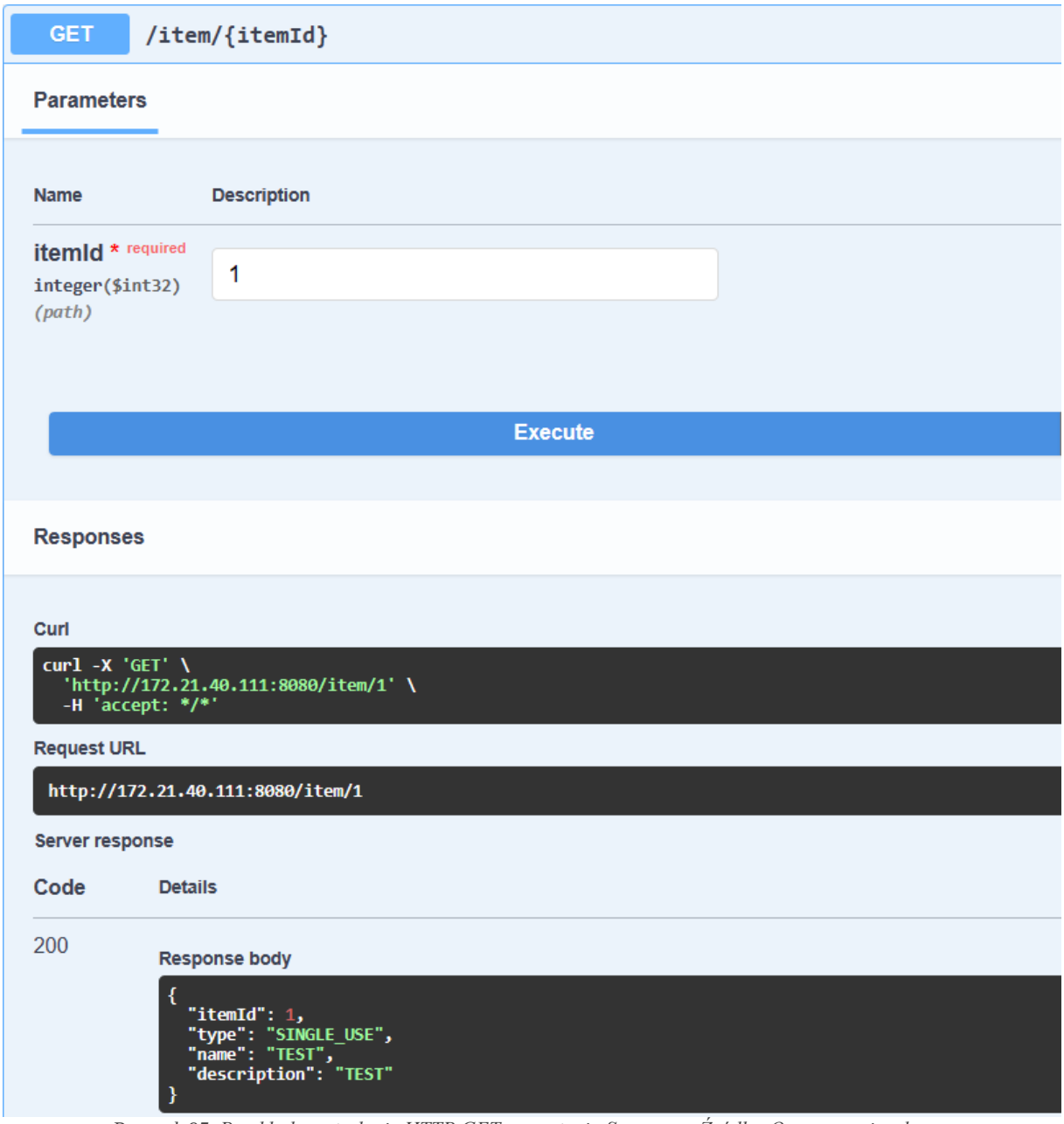


Rysunek 94. Przykładowe żądanie HTTP GET przy użyciu Postmana. Źródło: Opracowanie własne.

Ponadto do wykonania testów API wykorzystywane było również narzędzie Swagger (ze względu na prostotę użytkowania). Swagger jest użyteczną biblioteką pozwalającą na wizualizowanie i korzystanie z API aplikacji, co jest bardzo przydatne przy tworzeniu dokumentacji. Dokumentacja jest aktualizowana automatycznie, co jest jedną z większych korzyści tego oprogramowania. Podstawowa konfiguracja Swaggera nie jest zbyt trudna, trwa zwykle nie więcej niż kilka minut.

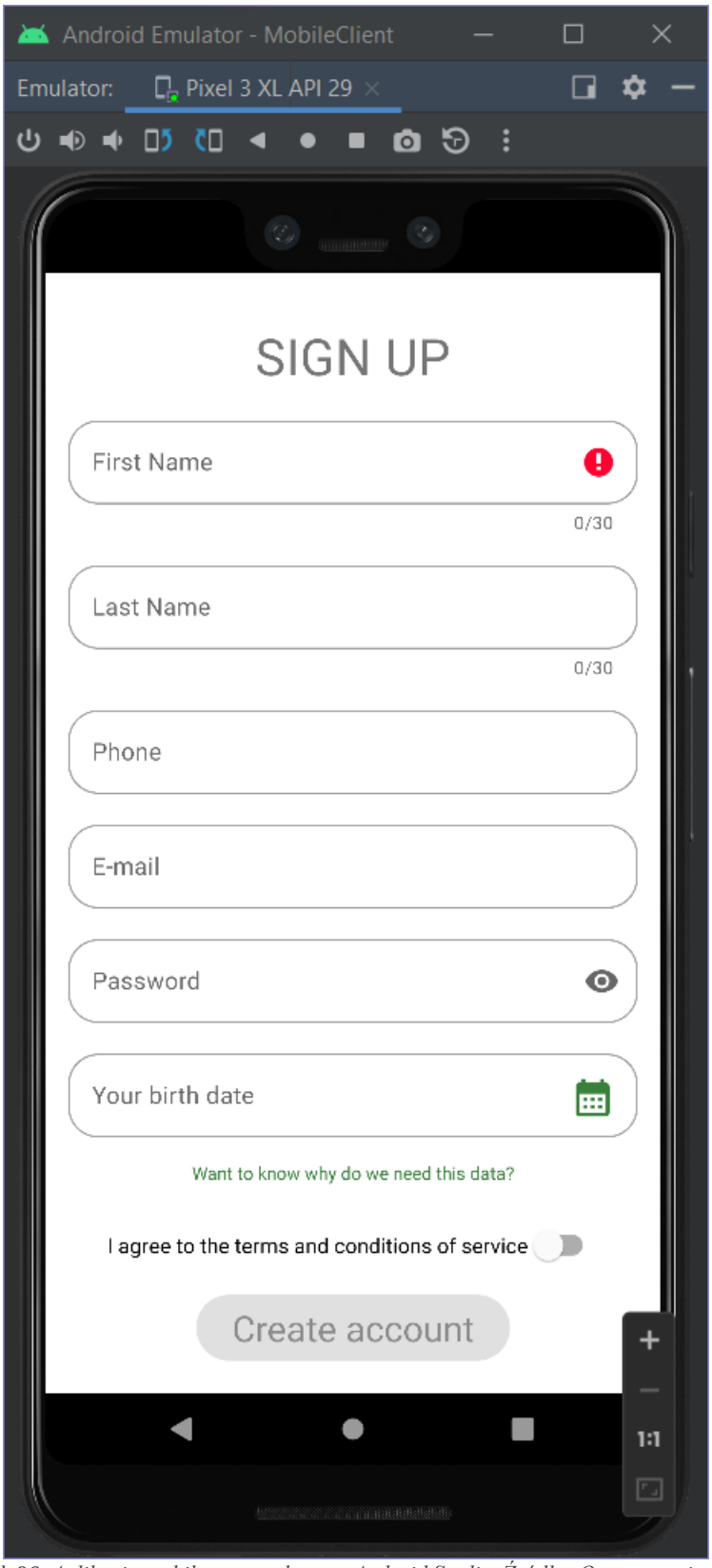
Główną funkcjonalnością Swaggera wykorzystaną w niniejszej pracy było sprawdzenie prawidłowości w działaniu kontrolerów, weryfikowanie potencjalnych błędów jak np. prześledzenie działania żądań, których niepożądanym wynikiem było wstawienie nieprawidłowych danych.

Na rysunku 95 został przedstawiony przykładowy proces testowania przy użyciu Swaggera. Polega on na wysłaniu żądania GET na adres wybranego do testów *endpointa*.
W odpowiedzi otrzymujemy listę wszystkich karetek, której zawartość możemy porównać ze stanem bazy danych w celu zweryfikowania poprawności działania.



Rysunek 95. Przykładowe żądanie HTTP GET przy użyciu Swaggera. Źródło: Opracowanie własne.

Testowanie aplikacji mobilnej zostało przeprowadzone przy pomocy środowiska programistycznego Android Studio. Oprogramowanie to umożliwia uruchomienie emulatora urządzenia mobilnego z systemem Android. Android Studio umożliwia wybór urządzenia m.in. czy chcemy przeprowadzać testy na tablecie, czy na telefonie. Dzięki temu testerzy mają możliwość przetestowania obu wersji niniejszej aplikacji mobilnej. Rozpoczęcie procesu zaczyna się od uruchomienia najnowszej wersji APK, przygotowanej przez zespół implementacyjny. Następnie testerzy manualnie sprawdzają, czy wszystkie funkcjonalności działają zgodnie z oczekiwaniami i czy po drodze nie występują jakieś błędy. Rysunek 96 przedstawia uruchomioną aplikację na emulatorze w Android Studio.



Rysunek 96. Aplikacja mobilna w emulatorze Android Studio. Źródło: Opracowanie własne.

8.2. Testy jednostkowe

Testy jednostkowe znajdują swoje zastosowanie w sprawdzeniu poszczególnych funkcjonalności programu. Przeprowadza się je w celu znalezienia potencjalnych błędów

w pojedynczych fragmentach kodu, aby zweryfikować czy istnieje jakiś błędny komponent, który zaburza funkcjonowanie aplikacji i powoduje brak zgodności z wymaganiami. Te testy wykonywane są w trakcie rozwijania aplikacji, aby jak najszybciej wykryć błędnie napisane fragmenty kodu.
Wśród ich zalet możemy wymienić:

- Poprawienie jakości napisanego kodu
- Wykrycie błędów w kodzie przed dopuszczeniem go na produkcję
- Ułatwienie w zrozumieniu funkcjonalności programu

W języku Java zautomatyzowane testy jednostkowe tworzone są przy pomocy biblioteki JUnit. Udostępnia ona funkcjonalności takie jak odpowiednie środowisko testerskie, automatyczne uruchomianie testów, asercje do porównywania oczekiwanych wyników z rzeczywistymi, a także adnotacje, którymi oznacza się metody do testowania kodu. Logika testów w wyżej wymienionej bibliotece polega na tworzeniu metod testujących odnoszących się do rzeczywistych metod. Ich celem jest sprawdzenie, czy rzeczywiste metody działają prawidłowo oraz zgodnie z oczekiwaniami i wymaganiami. Testowa metoda porównuje wynik oczekiwany z wynikiem rzeczywistym zwracanym przez metodę.

We wspomnianej aplikacji głównym testowanym aspektem przez testy jednostkowe były serwisy. Na Listingu 44 i 45 znajdują się przykładowe testy wspomnianych wyżej serwisów.

Listing 44. Przykładowy test jednostkowy serwisu (1). Źródło: Opracowanie własne.

```
    @Test
    void getTrustedPersonByEmailShouldFind() {
        TrustedPersonRequest     trustedPersonRequest     =     new
TrustedPersonRequest();
        trustedPersonRequest.setUserEmail("tom@pjatk.pl");
        trustedPersonRequest.setFirstName("Tomasz");
        trustedPersonRequest.setLastName("Kowalski");
        trustedPersonRequest.setPhone("123456789");

        User user = new User();
        when(userRepository
                .findByEmail(trustedPersonRequest
                        .getUserEmail()))
                .thenReturn(Optional.of(user));
        trustedPersonService
                .addTrustedPerson(trustedPersonRequest);

        TrustedPersonResponse result = trustedPersonService
                .getByEmail("tom@pjatk.pl");


        assertNotNull(result);
        assertEquals(trustedPersonRequest.getEmail(),
result.getEmail());

    }
```

Listing 45. Przykładowy test jednostkowy serwisu (2). Źródło: Opracowanie własne.

```
    @Test
    void updateAmbulance() {
        AddAmbulanceRequest ambulanceRequest = new
AddAmbulanceRequest();
        ambulanceRequest.setAmbulanceType(AmbulanceType.A);
```

```
        ambulanceRequest.setLicensePlate("WPI208");
        ambulanceRequest.setAmbulanceClass(AmbulanceClass.BASIC);
        ambulanceRequest.setSeats(3);
        ambulanceRequest.setLongitude(2.0);
        ambulanceRequest.setLatitude(3.0);

        when(ambulanceRepository
                .findByLicensePlate("WPI208"))
                .thenReturn(Optional.ofNullable(new Ambulance()));

        ambulanceService.updateAmbulance(ambulanceRequest);

        verify(ambulanceRepository, times(1))
                .save(any(Ambulance.class));
    }

    @Test
    void updateAmbulanceShouldFail() {
        String plates = "WPI208";

        AddAmbulanceRequest ambulanceRequest = new
AddAmbulanceRequest();
        ambulanceRequest.setAmbulanceType(AmbulanceType.A);
        ambulanceRequest.setLicensePlate(plates);
        ambulanceRequest.setAmbulanceClass(AmbulanceClass.BASIC);
        ambulanceRequest.setSeats(3);
        ambulanceRequest.setLongitude(2.0);
        ambulanceRequest.setLatitude(3.0);

        Exception exc = assertThrows(HttpException.class, () -> {
            ambulanceService.updateAmbulance(ambulanceRequest);
        });

        String expected = String.format("404 Ambulance %s not
found", plates);
        String actual = exc.getMessage();

        assertEquals(expected, actual);
    }
```

W niniejszej pracy oprócz możliwości przetestowania w każdym momencie znaczącej części aplikacji w momencie tworzenia *Pull Request*, aby dodać wprowadzone zmiany na serwer, wymagane było aby wszystkie dostępne w danym momencie testy przechodziły pozytywnie.

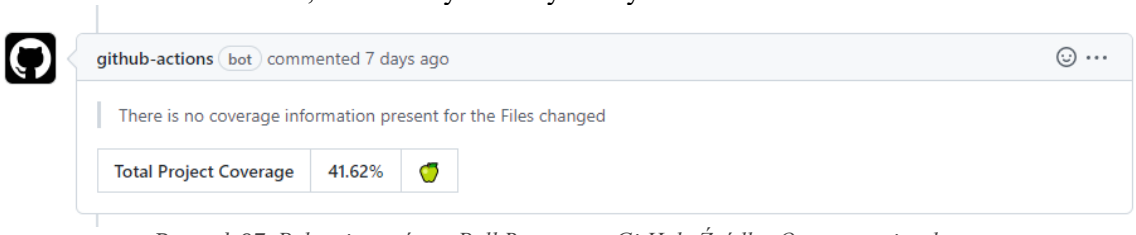
JaCoCo *plugin* to wtyczka umożliwiająca wygenerowanie raportu z pokrycia kodu źródłowego testami. Liczenie pokrycia może odbywać się na dwa sposoby: *line coverage* lub *branch coverage* (z tego korzysta JaCoCo). Pierwsza z metod to suma wszystkich linii uruchomionych poprzez testy dzielona przez wszystkie linie. Jest to najprostszy sposób liczenia pokrycia, ale również najmniej dokładny. Przy *branch coverage* uwzględnia się , by każda z decyzji miała przynajmniej raz wartość *true* oraz przynajmniej raz wartość *false*. Zakładając, że mamy test na usuwanie choroby, gdzie istnieje choroba z podanym id to na podstawie Listingu 46 w przypadku *line coverage* otrzymujemy 75%, a dla *branch coverage* 50%.

Listing 46. Liczenie pokrycia na podstawie metody. Źródło: Opracowanie własne.

```
public void removeDisease(Integer id){
```

```
      Optional<Disease>              optionalDisease              =
diseaseRepository.findByDiseaseId(id);
      if (optionalDisease.isEmpty()) {
            throw new HttpException(HttpStatus.NOT_FOUND,
                        String.format("Cannot find disease with %s",
id));
      }
      Disease disease = optionalDisease.get();
      diseaseRepository.delete(disease);
}
```

Każdy *Pull Request* wyświetla pokrycie procentowe projektu, jak i jego poszczególnych elementów w momencie stworzenia PRa, co możemy zobaczyć na rysunku 97.



Rysunek 97. Pokrycie testów w Pull Request na GitHub. Źródło: Opracowanie własne.

9. Podsumowanie

Prace nad przedstawionym projektem były czasochłonne i wymagające. Trwały one trzy semestry nauki z uwzględnieniem przerw wakacyjnych oraz świątecznych. Zespół zdobył duże doświadczenie i rozeznanie w tym, jak wygląda tworzenie oprogramowania od podstaw. Zdobyte umiejętności pomogły wielu osobom z grupy projektowej zdobyć posadę w zawodzie oraz otworzyć się na nowe kierunki rozwoju. Można stwierdzić, że postawione założenia zostały spełnione oraz prace nad projektem zakończyły się sukcesem. W następnych podrozdziałach zostały opisane napotkane trudności oraz pomysły na przyszły rozwój aplikacji.

9.1. Wprowadzenie

Zespół składał się z dziesięciu studentów informatyki ze specjalizacji Inżynieria Oprogramowania I Baz Danych. W momencie przystępowania do projektu większość nie posiadała komercyjnego doświadczenia w szeroko pojętym IT. Większość kwalifikacji opierało się na wiedzy wyniesionej w toku studiów, a w trakcie trwania prac zespół znacznie rozwinął umiejętności twarde, jak i miękkie oraz poznał nowe obszary i zagadnienia takie jak Jenkins, unit testy, czy React. Pracę nad projektem zostały rozpoczęte od analizy zagadnienia oraz wybrania funkcjonalności, które były w planach wprowadzenia do systemu. Jednocześnie powstał stan sztuki, który można przeczytać w rozdziale trzecim niniejszej pracy. Następnie trwały równolegle prace implementacyjne, testowe i dokumentacyjne. Podczas wdrażania systemu kilkakrotnie nanoszone były poprawki do diagramów przygotowanych podczas fazy analizy.

9.2. Napotkane trudności

Dla całego zespołu Gary to był pierwszy projekt jaki został rozwinięty od początku do końca. W związku z tym wystąpiło wiele trudności - od organizacji pracy po niewielką znajomość technologii. Wszystkie zostały pokonane a rozwiązane problemy stanowią doświadczenie w przyszłej pracy w branży IT. Poniżej kilka przykładowych problemów napotkanych w toku pracy:

1. Zmienność grupy – pracę zaczęto mając do dyspozycji piętnaście osób. Jednak wraz z postępami ta liczba zmniejszyła się z różnych powodów – zakończenie studiów, zmiana specjalizacji itd. Było to wyzwanie dla wszystkich pozostałych uczestników projektu – zwiększała się ilość pracy do wykonania przez kontynuujące projekt osoby oraz trzeba było modyfikować przydzielone zadania i harmonogram.
2. Problemy z serwerem – kilka miesięcy był problem z działaniem serwera. Skorzystano z serwera udostępnionego przez uczelnię. Niestety w trakcie drugiego semestru pracy w wyniku pomyłki do adresu niniejszego serwera został przypisany inny.
   Na szczęście udało się wyjaśnić sytuację z władzami uczelni oraz ponownie serwer wrócił na wcześniejszy adres IP. Bez problemu udało się przywrócić jego wcześniejszy stan.
3. Niewielkie doświadczenie w fazie analizy – skutkiem tego jest wielokrotne wprowadzanie poprawek i modyfikacji trwających prawie do końca projektu. Skutkowało to opóźnieniami wdrożenia systemu.
4. Nieznajomość technologii – wraz z rozpoczęciem prac zespół nie miał doświadczenia z tworzeniem aplikacji mobilnych, czy językiem Kotlin. Spowodowało to miesięczne opóźnienie wdrożenia aplikacji mobilnej na rzecz intensywnej nauki języka i programowania mobilnego od podstaw. Dużym wsparciem okazał się darmowy kurs od Google, dzięki któremu w łatwy i przyjemny sposób mogliśmy nadrobić braki wiedzy oraz umiejętności.

9.3. Rozwój aplikacji

Z powodu ograniczonych środków oraz czasu nie zostały wprowadzone wszystkie funkcjonalności, które zaproponowaliśmy wraz z pierwszą wersją wymagań funkcjonalnych.

Skupiliśmy się na kluczowych funkcjonalnościach, które są wymagane dla podstawowej wersji systemu. Poniżej znajduje się lista z propozycją kolejnych elementów:

1. Niebezpieczne zdarzenia w okolicy – użytkownik może zgłosić niebezpieczne zdarzenie, albo niebezpieczną okolicę bez powiadamiania służb np. powódź. Użytkownicy będący w danej strefie zagrożenia otrzymują powiadomienie na telefon. Możliwe jest również przeglądanie niebezpiecznych zdarzeń z poziomu mapy. Użytkownicy na terenie zagrożenia mogą potwierdzić lub zanegować jego istnienie.
2. Czujnik wstrząsów w telefonie – po znacznym upadku telefonu, jeśli użytkownik do minuty nie odwoła zgłoszenia, zostaną wezwane do niego służby ratunkowe. Podobne systemy znajdują się obecnie w najnowszych autach.
3. Zwiększenie wykorzystania zaufanej osoby – wysłanie wiadomości SMS, w tym do osób nieposiadających zainstalowanej aplikacji.
4. Rozwinięcie obsługiwania systemu o inne służby ratunkowe takie jak straż pożarna, czy policja wraz z niezbędnymi do ich obsługi modułami.
5. Wprowadzenie interaktywnych poradników oraz zaproponowanie wyświetlenia odpowiedniego poradnika po wysłaniu zgłoszenia.

9.4. Podsumowanie

W trakcie projektu zostały stworzone oraz wdrożone wszystkie zaplanowane moduły dla Systemu Obsługi Incydentów Ratunkowych – Gary.

Zgodnie z założeniami zrealizowane zostały podstawowe funkcjonalności zaplanowane na początku powstawania systemu:

- Możliwość wezwania przez użytkownika służb ratunkowych.
- Możliwość reagowania przez pracowników na stworzone zgłoszenie
- Łatwy dostęp do materiałów związanych z pierwszą pomocą przedmedyczną.

Ostatnia z funkcjonalności jest dostępna dla wszystkich osób zarówno tych zalogowanych, jak i nie. Jednym z głównych celów aplikacji jest zwiększenie ilości osób udzielającej pierwszej pomocy w razie wypadku oraz zwiększenie poprawności jej udzielania.

Pozostałe wymienione funkcjonalności dostępne są po uprzednim zalogowaniu użytkownika z uwzględnieniem przypisanej do niego roli. Użytkownik nie będący pracownikiem może stworzyć zgłoszenie i obserwować jego status. Dyspozytor może przyjąć lub odrzucić zgłoszenie oraz przeprowadzić je przez cały proces zaczynając od przypisania karetki, poprzez wezwanie dodatkowych służb. Ratownik otrzymuje wszystkie informacje o zgłoszeniu wraz z dokładną lokalizacją. Po jego zakończeniu może w łatwy sposób sporządzić raport ze zgłoszenia.

Ratowanie życia ludzkiego nie zawsze jest możliwe, a na pewno nie jest łatwe. Niniejsza aplikacja ma za zadanie zwiększyć szybkość udzielanej pomocy, co zwiększa szanse na przeżycie osób w stanie krytycznym.

10. Bibliografia

[1]. Sklep Google Play – Aplikacja Ratunek Polkomtel Sp. Z.o.o. – Dostęp: 16.12.2022
https://play.google.com/store/apps/details?id=com.pagasolutions.emergencycall_pl&hl=pl&gl=US&pli=1
[2]. Sklep Google Play – Aplikacja Rodzinne S.O.S. Lacon Sp. z o. o. – Dostęp: 16.12.2022
https://play.google.com/store/apps/details?id=pl.locon.gjd.safety&hl=pl&gl=US
[3]. Sklep Google Play – Aplikacja Alarm112 MSWiA – Dostęp: 16.12.2022
https://play.google.com/store/apps/details?id=com.help.deafhelp
[4]. Strona internetowa – Aplikacja AsthmaMD – Dostęp: 16.12.2022
https://www.asthmamd.org
[5]. Sklep Google Play – Aplikacja Informacje Medyczne POZ Asseco Poland S.A. – Dostęp: 16.12.2022
https://play.google.com/store/apps/details?id=pl.asseco.informacjemedyczne&hl=pl&gl=US
[6]. Strona internetowa – Aplikacja Niezbędnik Ratownika – Dostęp: 16.12.2022
https://ratownicy24.pl/aplikacja-niezbednik-ratownika-wkrocz-w-nowy-wymiar-ratownictwa/
[7]. Sklep Google Play – Aplikacja Ratownictwo medyczne algorytmy Bit Med. Marcin Andrzejczak. – Dostęp: 16.12.2022
https://play.google.com/store/apps/details?id=pl.waw.bitmed.protokols
[8]. Strona internetowa – Aplikacja Ratownik KPP: aktualne testy – Dostęp: 16.12.2022
https://www.centrumratownictwa.com/site/index?url=aktualnosci%2Faplikacja-ratownik-kpp-aktualne-testy
[9]. Strona internetowa – SWD PRM – Dostęp: 16.12.2022
https://kcmrm.pl
[10]. Wikipedia – Kotlin – Dostęp: 03.01.2023
https://pl.wikipedia.org/wiki/Kotlin_(język_programowania)
[11]. Strona internetowa – Kotlinlang.org – Dostęp: 03.01.2023
https://kotlinlang.org/docs/multiplatform-mobile-create-first-app.html
[12]. Wikipedia – Android Studio – Dostęp: 03.01.2023
https://pl.wikipedia.org/wiki/Android_Studio
[13]. Strona internetowa – Elementor.com – Dostęp: 03.01.2023
https://elementor.com/blog/what-is-material-design/
[14]. Strona internetowa – Github – Dostęp: 03.01.2023
https://square.github.io/retrofit/
[15]. Github – Osmdroid – Dostęp: 03.01.2023
https://github.com/osmdroid/osmdroid
[16]. Strona internetowa – webporady.pl – Dostęp: 03.01.2023
https://webporady.pl/prosta-strona-internetowa-w-html/
[17]. Wikipedia – CSS – Dostęp: 03.01.2023
https://pl.wikipedia.org/wiki/Kaskadowe_arkusze_stylów
[18]. Strona internetowa - lokalise.com – Dostęp: 03.01.2023
https://lokalise.com/blog/json-l10n/
[19]. Strona internetowa – switch2.osm.org – Dostęp: 03.01.2023
https://switch2osm.org/using-tiles/getting-started-with-leaflet/
[20]. Wikipedia – Java – Dostęp: 03.01.2023
https://en.wikipedia.org/wiki/Java_(software_platform)
[21]. Strona internetowa – spring.io – Dostęp: 03.01.2023
https://spring.io/guides/gs/spring-boot/
[22]. Strona internetowa – digitalocean.com – Dostęp: 03.01.2023
https://www.digitalocean.com/community/tutorials/hibernate-tutorial-for-beginners
[23]. Wikipedia – Hibernate – Dostęp: 03.01.2023
https://pl.wikipedia.org/wiki/Hibernate

[24]. Strona internetowa – baeldung.com – Dostęp: 03.01.2023
https://www.baeldung.com/intro-to-project-lombok
[25]. Strona internetowa – OVHCloud – Dostęp: 03.01.2023
https://www.ovhcloud.com/pl/public-cloud/postgresql/
[26]. Strona internetowa – enterprisedb.com – Dostęp: 03.01.2023
https://www.enterprisedb.com/postgres-tutorials/postgresql-query-introduction-explanation-and-50-examples
[27]. Strona internetowa – JFrog – Dostęp: 03.01.2023
https://jfrog.com/knowledge-base/the-basics-a-beginners-guide-to-docker/
[28]. Wikipedia – Jenkins – Dostęp: 03.01.2023
https://pl.wikipedia.org/wiki/Jenkins_%28oprogramowanie%29
[29]. Strona internetowa – Microsoft – Dostęp: 03.01.2023
https://learn.microsoft.com/pl-pl/visualstudio/version-control/git-browse-repository?view=vs-2022
[30]. Strona internetowa – droptica.pl – Dostęp: 03.01.2023
https://www.droptica.pl/blog/co-jest-postman-do-czego-sluzy-i-jakie-sa-jego-funkcjonalnosci/
[31]. Strona internetowa – bykowski.pl – Dostęp: 03.01.2023
https://bykowski.pl/swagger-ui-przejrzysta-wizualizacja-zasobow-api/
[32]. Wikipedia – Junit – Dostęp: 03.01.2023
https://pl.wikipedia.org/wiki/JUnit
[33]. Strona internetowa - baeldung.com – Dostęp: 03.01.2023
https://www.baeldung.com/jacoco
[34].Strona internetowa - dzone.com – Dostęp: 03.01.2023
https://dzone.com/articles/reporting-code-coverage-using-maven-and-jacoco-plu
[35]. Strona internetowa – guru99.com – Dostęp: 19.01.2023
https://www.guru99.com/java-platform.html#7
[36]. Strona internetowa – techtarget.com – Dostęp: 19.01.2023
https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework
[37]. Strona internetowa – auth0.com – Dostęp: 19.01.2023
https://auth0.com/blog/a-complete-guide-to-lombok/#The-Lombok-Plugin
[38]. Strona internetowa – ibm.com – Dostęp: 19.01.2023
https://www.ibm.com/topics/docker
[39]. Strona internetowa – encora.com – Dostęp: 19.01.2023
https://www.encora.com/insights/what-is-postman-api-test
[40]. Strona internetowa infoworld.com – Dostęp: 19.01.2023
https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html
[41]. Strona internetowa – androidcode.pl – Dostęp: 24.01.2023
http://androidcode.pl/blog/biblioteki/retrofit/
[42]. Wikipedia – HTML5 – Dostęp: 24.01.2023
https://pl.wikipedia.org/wiki/HTML5
[43]. Wikipedia – Git – Dostęp: 24.01.2023
https://pl.wikipedia.org/wiki/Git_(oprogramowanie)
[44]. Strona internetowa – javastart.pl – Dostęp: 24.01.2023
https://javastart.pl/baza-wiedzy/frameworki/project-lombok

11. Spis tabel

12. Spis listingów

13. Spis rysunków