

WYŻSZA SZKOŁA BANKOWA W GDAŃSKU
WYDZIAŁ INFORMATYKI I NOWYCH TECHNOLOGII

Jakub Krzykwa

Nr albumu 53412

**Projekt aplikacji mobilnej
do zarządzania parkiem maszynowym
w gospodarstwie rolnym**

Praca inżynierska

Na kierunku Informatyki

Praca napisana pod kierunkiem

Dr inż. Artura Ziółkowskiego

Gdańsk 2023

Spis treści

Wstęp	5
1 Stan wiedzy o projektowaniu aplikacji mobilnych	7
1.1 Trendy w zakresie projektowania aplikacji mobilnych	7
1.1.1 Analiza trendów technologicznych	7
1.1.2 Analiza trendów użytkowych	11
1.2 Przegląd aktualnych rozwiązań rynkowych	13
1.3 Technologie wspomagające budowanie aplikacji mobilnej	17
1.4 Podsumowanie części teoretycznej	18
2 Problematyka tworzenia aplikacji zarządzającej parkiem maszynowym	19
2.1 Określenie problemu zarządzania parkiem maszynowym	19
2.2 Wymagania funkcjonalne i нефункционалне	20
2.3 Model aplikacji	22
2.3.1 Diagramy przypadków użycia – UML	22
2.3.2 Modele procesów realizowanych przez aplikację	31
2.4 Podsumowanie części problemowej	36
3 Projekt front-endu (interfejsu) aplikacji do zarządzania parkiem maszynowym w gospodarstwie rolnym	37
3.1 Widok menu głównego	37
3.2 Widok maszyn i ich edycji	38
3.3 Widok wymian	40

3.4	Widok wymian olejów	40
3.5	Widok wymian części	43
3.6	Widok tankowania	45
3.7	Widok zbiorników w gospodarstwie.....	46
3.8	Widok zamówień kierowców.....	48
3.9	Widok kierowców w gospodarstwie	50
3.10	Ocena przygotowanych widoków interfejsu	51
4	Projektowanie back-endu aplikacji do zarządzania parkiem maszynowym	52
4.1	Implementacja bazy danych	52
4.2	Implementacja aplikacji	58
4.2.1	Widoki, komponenty	58
4.2.2	Połączenie bazy danych z aplikacją	59
4.2.3	Przechodzenie pomiędzy głównymi widokami aplikacji.....	61
4.2.4	Działanie i implementacja listy RecyclerView	63
4.2.5	Działanie listy ListView	68
4.2.6	Dodawanie, usuwanie i aktualizacja danych w aplikacji	69
4.2.7	Pobieranie daty z kalendarza	74
4.3	Testy	75
	Zakończenie	77
	Spis rysunków	79

Bibliografia.....	82
-------------------	----

Wstęp

W dzisiejszych czasach coraz częściej można zauważyć jak ogromny wpływ i ułatwienia wprowadza implementacja technologii w przeróżnych przedsiębiorstwach. Celem zastosowania technologii w firmach może być zwiększenie skuteczności i szybkości wykonywania działań, ułatwienie zrozumienia systemu, prowadzenie dogłębnych analiz czy bardziej profitowe zarządzanie przedsiębiorstwem.

Autor pracy zauważył możliwość automatyzacji procesu zarządzania parkiem maszynowym w gospodarstwie rolnym i przedstawił propozycję stworzenia programu ułatwiającego pracę osobie na stanowisku mechanizatora w tym gospodarstwie. Poprzez własne doświadczenia oraz rozmowę z doświadczoną w tej dziedzinie osobą, której system byłby diametralnym ułatwieniem pracy, wyciągnięto odpowiednie wnioski dotyczące urządzenia jakie byłoby odpowiednie do wykonywania owej pracy oraz funkcji jakie powinien posiadać system. Fakt iż obiorca powinien posiadać stały dostęp do informacji, był powodem do podjęcia decyzji o stworzeniu aplikacji mobilnej.

Aplikacja umożliwia użytkownikowi sprawdzanie danych na temat konkretnych maszyn, ich wymian i zatankowań, monitorowanie ilości paliwa w zbiornikach znajdujących się na terenie gospodarstwa, a także przeglądanie oraz dodawanie dostaw paliw. Ponadto system będzie pomagał użytkownikowi w śledzeniu aktualnie potrzebnych pracownikom przedmiotów do kupienia, poprzez sekcje zamówień wyposażoną w możliwości dodania, usuwania i aktualizacji danych. Aplikacja zapewni także wgląd w kierowców pojazdów na gospodarstwie razem z podstawowymi informacjami o nich.

Praca została podzielona na cztery rozdziały. Pierwszy z nich przedstawia ogólny stan wiedzy na temat projektowania aplikacji mobilnej. Rozdział zawiera analizę rynku w dziedzinie trendów technologicznych oraz użytkowych, pomagając tym samym w wyborze najlepszego systemu oraz jego wersji. Ponadto opisuje i analizuje dostępne rozwiązania na rynku. Kolejny rozdział opisuje problem zarządzania parkiem maszynowym. Przedstawia on funkcjonalności jakie powinna posiadać aplikacja oraz ich zobrazowanie przez diagram przypadków użycia oraz modele procesów w nim zachodzących. Przedostatni rozdział prezentuje i opisuje projekt interfejsu aplikacji. W ostatnim rozdziale zawarto najważniejsze informacje o procesie tworzenia back-endu

systemu. Zostały w nim przedstawione przykładowe fragmenty kodu, naprowadzające na to jak działają poszczególne funkcjonalności zawarte w finalnej wersji projektu.

1 Stan wiedzy o projektowaniu aplikacji mobilnych

W tym rozdziale zostaną przedstawione trendy w zakresie tworzenia aplikacji mobilnych. Zostaną przeanalizowane trendy zarówno technologiczne jak i użytkowe, poprzez grafy, wykresy oraz informacje z serwisów poświęconych urządzeniom mobilnym.

W pracy zostaną również przedstawione i przeanalizowane aktualne rozwiązania rynkowe na podstawie przykładu, jak również omówione użyte technologie, które pomogły osiągnąć początkową wersję projektu do zarządzania parkiem maszynowym.

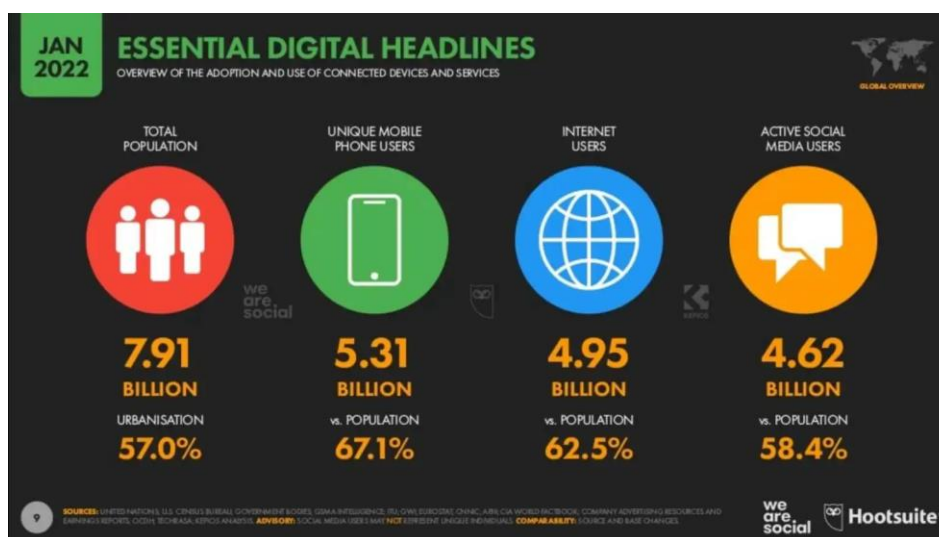
1.1 Trendy w zakresie projektowania aplikacji mobilnych

W tej części zostaną przedstawione i przeanalizowane trendy technologiczne oraz użytkowe w dzisiejszym społeczeństwie, w celu lepszego zrozumienia najpopularniejszych rozwiązań oraz inspiracji do stworzenia koncepcji projektu.

1.1.1 Analiza trendów technologicznych

Nie jest tajemnicą, że większość ludzi na świecie jest w posiadaniu smartfona i używa go praktycznie codziennie. Aktualnie nie można wyobrazić sobie życia bez tego urządzenia, gdyż zapewnia użytkownikom stały kontakt z innymi, dostęp do informacji, funkcje bankowości, płatności w sklepie, rozrywkę i wiele innych. Smartfony posiadają również wiele mniej kompleksowych funkcjonalności, a posiadanie ich wszystkich w jednym urządzeniu doprowadziło do wyparcia wielu dobrze nam znanych urządzeń takich jak zegarki, budziki, odtwarzacze MP3 i MP4 czy kalkulatory. Smartfon pomimo tak wielu funkcjonalności nie jest niezbędny w życiu człowieka, jednak dysponowanie nim diametralnie ułatwia życie posiadacza. Dlatego też używa go ogromna część populacji na ziemi.

Rysunek 1.1 Użytkownicy smartfonów względem populacji

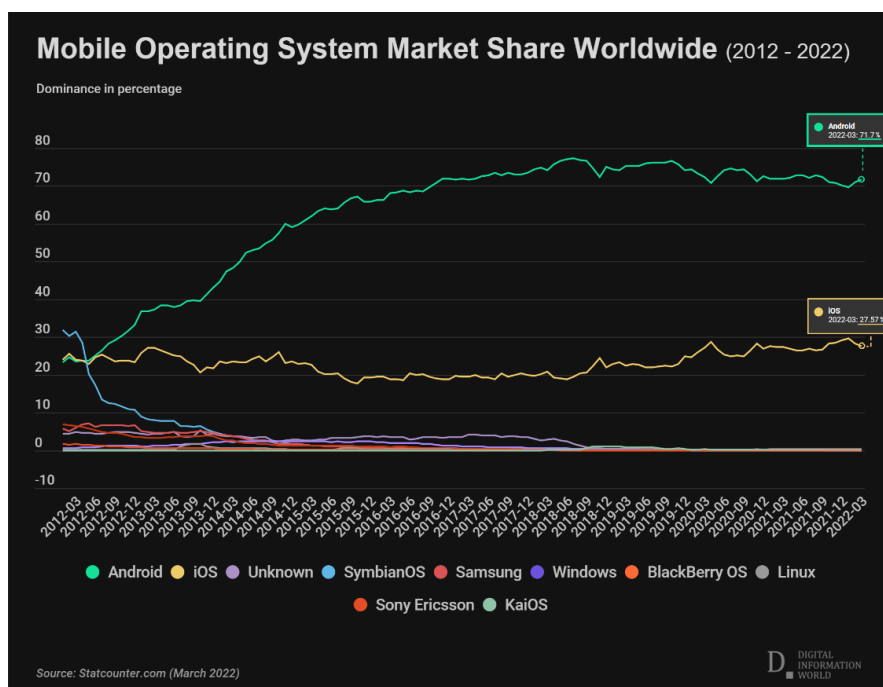


Źródło: <https://mobirank.pl/2022/01/27/digital-i-mobile-w-2022-roku/> [data odczytu 28.01.2023]

Jak przedstawia rysunek z serwisu poświęconego urządzeniom mobilnym, smartfony były używane przez ponad 67% całej populacji w roku 2022. Patrząc również na informacje z poprzednich lat, ilość unikalnych użytkowników urządzeń mobilnych stale rośnie.

Wraz z tak dużą ilością smartfonów na świecie wzrasta konkurencja na rynku. Producenci przez lata próbowali sprostać oczekiwaniom użytkowników tworząc oraz modyfikując coraz to nowsze wersje urządzeń mobilnych. Czasy gdy telefon służył człowiekowi wyłącznie do kontaktu z innymi oraz posiadał najprostsze funkcjonalności dawno minął i został wyparty przez erę smartfonów, które zostały wyposażone w systemy operacyjne, tym samym ewoluując w tak zwane, komputery przenośne. Technologia rozwinęła się bardzo szybko, a popyt na smartfony rósł co wyłoniło wiele urządzeń z różnymi systemami operacyjnymi, na których temat tych topowych z nich, mało kto nie słyszał.

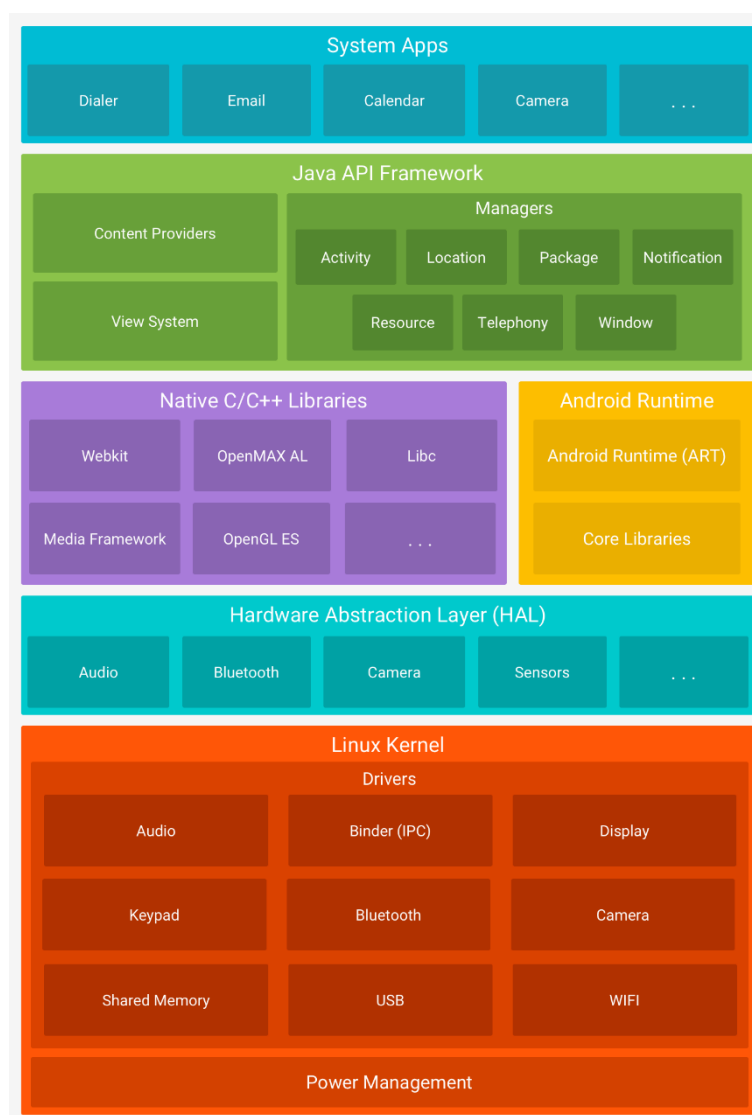
Rysunek 1.2 Udziały w rynku mobilnych systemów operacyjnych na świecie od 2012 do 2022 roku



Źródło: <https://www.digitalinformationworld.com/2022/04/google-android-vs-apple-ios-data.html> [data odczytu 28.01.2023]

Jak można zaobserwować na podanym wykresie od wielu lat, rynek jest zdominowany przez system operacyjny Android, który jest używany przez ponad 71% urządzeń mobilnych. System nie bez powodu jest liderem w swojej dziedzinie, cechuje się on bowiem łatwą i intuicyjną obsługą, przez co może być używany przez dosłownie każdego, nawet osobę dopiero odkrywającą świat smartfonów. Równie łatwym jest stworzenie aplikacji mobilnej na system Android, gdyż potrzebne do tego narzędzia oraz dokumentacja są za darmo udostępnione w sieci.

Rysunek 1.3 Architektura systemu Android



Źródło: <https://developer.android.com/guide/platform> [data odczytu 28.01.2023]

Jak przedstawia ilustracja, architektura systemu Android jest podzielona na 6 głównych warstw. Analiza architektury pomoże to w zrozumieniu działania systemu.

Podstawową funkcją warstwy Linux Kernel jest zajmowanie się wielowątkowością oraz zarządzaniem pamięcią. Odpowiada również za bezpieczeństwo systemu oraz jądra systemu linux. Posiada sterowniki do sprzętu typu kamera, USB, Wi-Fi czy Bluetooth.

Warstwa Hardware Abstraction Layer (HAL) zapewnia standardowe interfejsy, przeznaczone do komunikacji z elementami sprzętowymi smartfona, wytworzonego przez producenta, takimi jak kamera, Bluetooth czy audio.

Warstwa Android Runtime to środowisko wykonawcze. Tej warstwy używają wszystkie aplikacje w systemie, umożliwiając im działanie na własnym procesie i z własną instancją Android Runtime. Posiada ona również zestaw podstawowych bibliotek wykonawczych, zapewniających funkcje języka programowania Java.

Warstwa Native C/C++ Libraries udostępnia podstawowe komponenty i usługi systemu Android jak ART i HAL, które są zbudowane z natywnego kodu używającego natywne biblioteki napisane w języku C i C++. W warstwie zostały zawarte biblioteki służące np. generowaniu grafiki 3D czy obsługi przeglądarki internetowej.

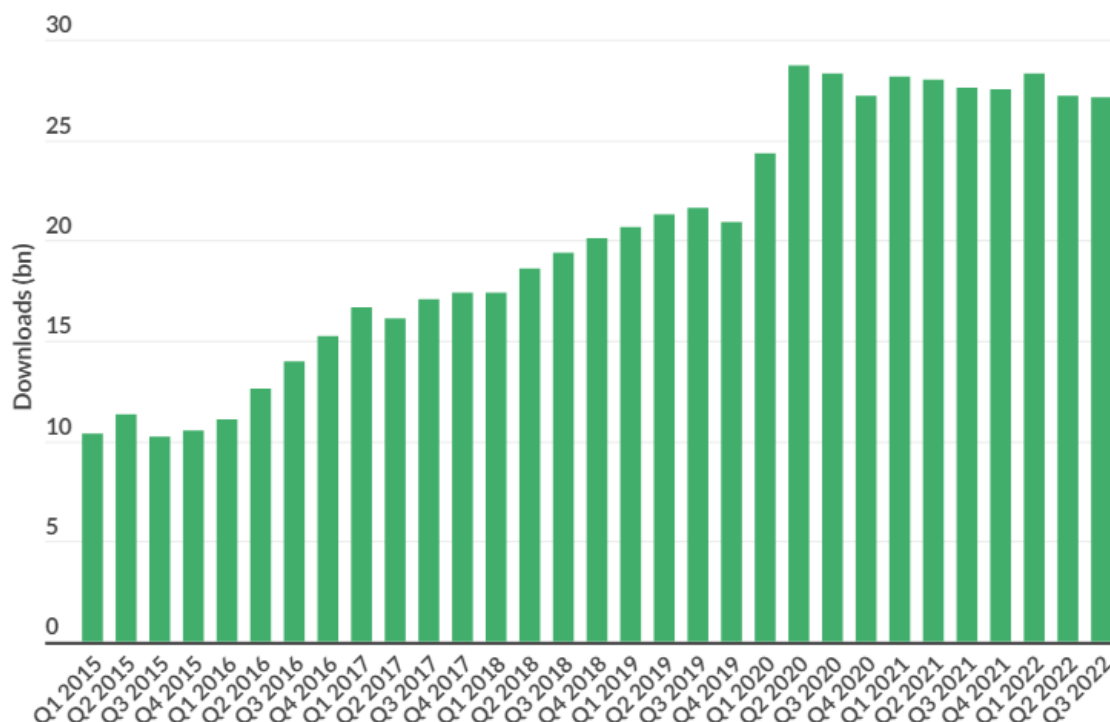
Java Api Framework zawiera zestaw klas służących w tworzeniu aplikacji dla systemu Android.

Warstwa System Apps dostarcza zestaw podstawowych funkcjonalności systemowych takich jak obsługa kamery, kalendarza, wiadomości SMS czy kontaktów.

1.1.2 Analiza trendów użytkowych

Wraz z rosnącym zainteresowaniem smartfonów używających systemu Android, powstawały coraz to nowsze jego systemy operacyjne oraz funkcjonalności. Z początku powstał Android Market, sklep internetowy, służący do instalacji przeróżnych aplikacji na urządzenie. Nie długo potem zostały również wprowadzone usługi umożliwiające pobieranie popularnych filmów, książek oraz muzyki. Stosunkowo szybko Android Market oraz Google Music przekształcono w jedną usługę, znaną dzisiaj jako Google Play. Z biegiem lat sklep internetowy stawał się coraz lepszy i prostszy w użyciu z łatwym dostępem do aplikacji, filmów, muzyki i wiele więcej.

Rysunek 1.4 Kwartalne pobrania aplikacji oraz gier z Google Play w latach 2015-2022

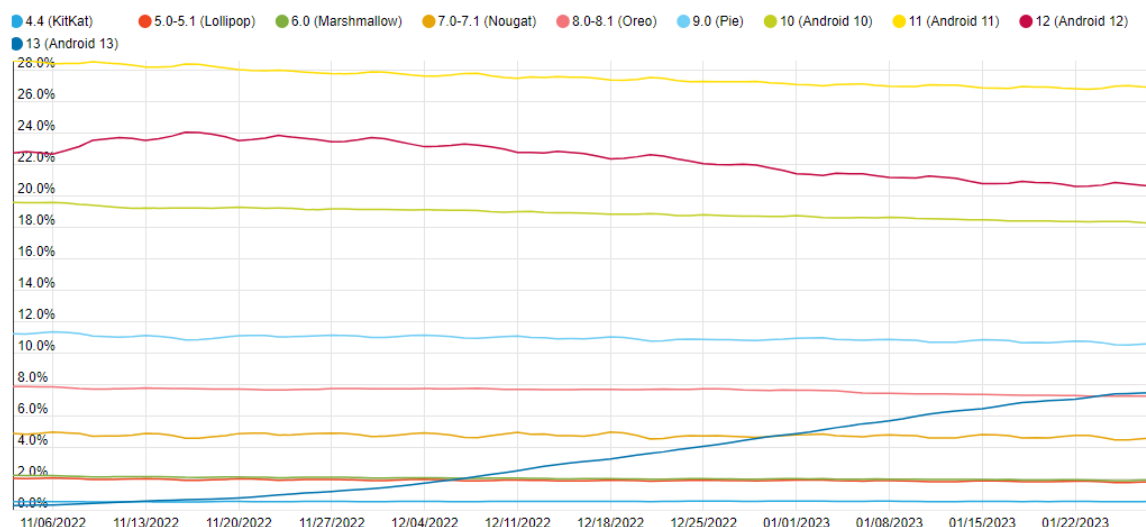


Źródło: <https://www.businessofapps.com/data/app-statistics/> [data odczytu 28.01.2023]

Jak przedstawia powyższy graf popyt na aplikacje oraz gry mobilne z serwisu Google Play jest ogromny. W ostatnich 4 przedstawionych na ilustracji kwartałach, przekracza on nawet 100 miliardów pobrań. Pokazuje to, że popularność na aplikacje mobilne jest duża, a co za tym idzie tworzenie ich profitowe.

Jednak przed stworzeniem aplikacji mobilnej, warto również sprawdzić jakie wersje systemu operacyjnego są aktualnie używane przez posiadaczy smartfonów z systemem Android.

Rysunek 1.5 Najpopularniejsze wersje systemu Android w ostatnich miesiącach



Źródło: <https://www.appbrain.com/stats/top-android-sdk-versions> [data odczytu 29.01.2023]

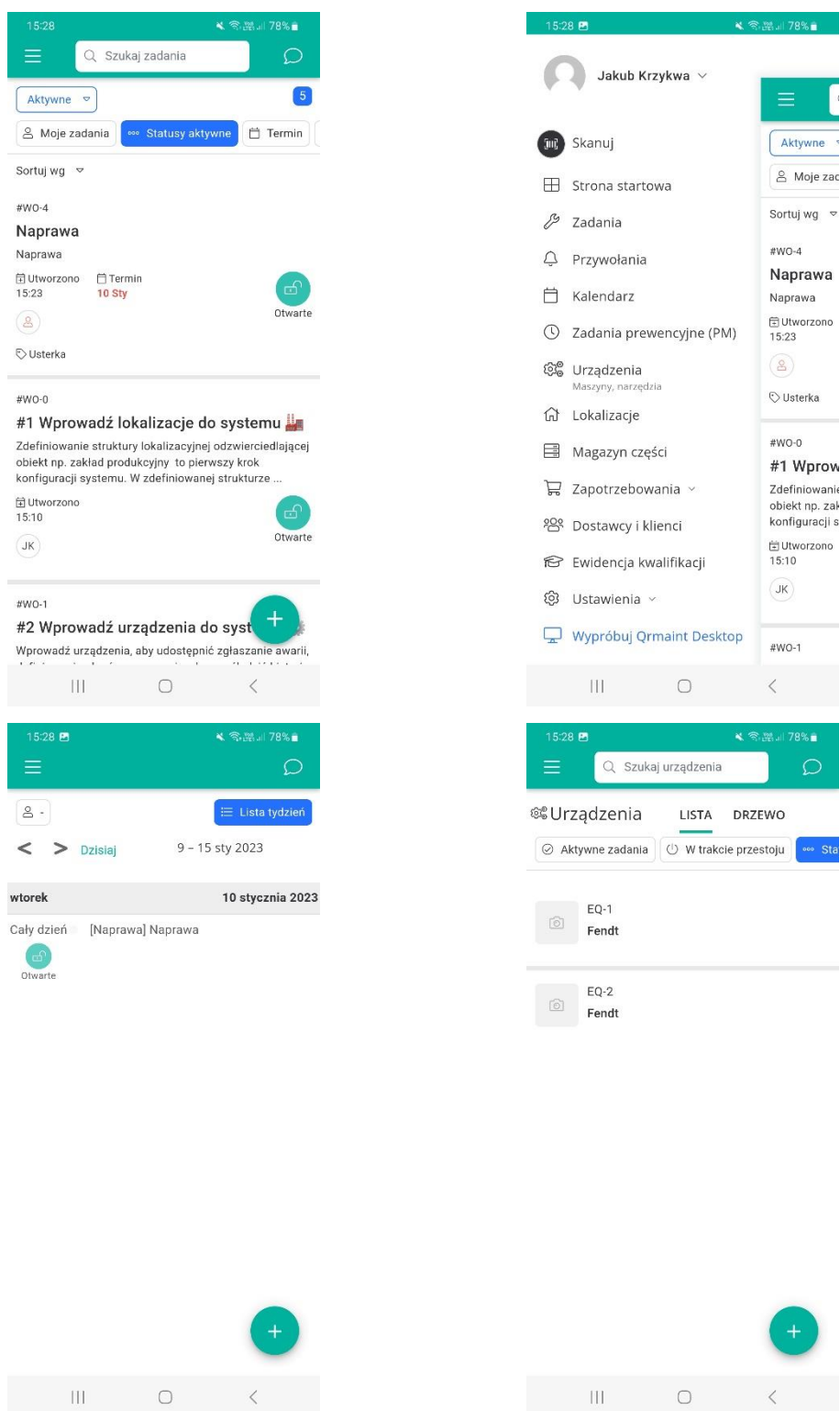
Patrząc na wykres przedstawiający najczęściej używane wersje systemu Android w ostatnich miesiącach można dostrzec, że znacząca większość urządzeń, bo ponad 90%, ma zainstalowany system operacyjny wersji 8.0 lub wyższej.

1.2 Przegląd aktualnych rozwiązań rynkowych

W celu stworzenia aplikacji o tematyce zarządzania parkiem maszynowym w gospodarstwie rolnym należałoby przyjrzeć się również rynkowi i skompletować informację o aktualnych rozwiązaniach. Niniejszy podrozdział poprzez zaprezentowanie funkcjonalności podobnej aplikacji oraz przedstawienie jej wad i zalet, pomoże w tworzeniu docelowego projektu.

Przeszukując usługę Google Play można natrafić na aplikację „QRmaint”. System może działać równocześnie na smartfonie oraz komputerze i jest najlepszym rozwiązaniem do zarządzania na aktualnym rynku.

Rysunek 1.6 Aplikacja QRmaint



15:29
EQ-1 - Fendt

SZCZEGÓŁY
ZADANIA
PLANY PREWENCYJNE

Fendt

Status

● W eksploatacji

Znajduje się/Jest częścią

Gospodarstwo

Opiekun urządzenia

👤 **Jakub Krzykwa**

Dane urządzenia

Producent

Przykładowy producent 1

Model

Fendt

Numer seryjny

NOE 1234

Rodzaj urządzenia

Przykładowy rodzaj urządzenia 1

|||
○
<

15:29
Dodaj nową część
ZAPISZ

Nazwa części

Dodaj tag

☐ Przyjmij do magazynu

Dane części

Dodaj zdjęcie

Rodzaj części +
Wybierz

Producent +
Wybierz

Nr producenta

✓ ZAPISZ

|||
○
<

15:29
Szukaj...

30.01.2023 15:29:15

Dodaj pierwszą część

Dodaj części zamienne, aby monitorować ich ilości i zamówić w odpowiednim momencie.

+ Dodaj nową część

+

|||
○
<

15:29
Szukaj...

Zapotrzebowania

✓ Moje
Status
Rodzaj
Dostawca

PR-0 Zapotrzebowanie na części i materiały

1

Data wprowadzenia
30 sty 15:29

Zaakceptowane

+

|||
○
<

Źródło: Aplikacja QRmaint

Aplikacja cechuje się schludną i przejrzystą szatą graficzną oraz wieloma potrzebnymi funkcjonalnościami do zarządzania np. gospodarstwem czy zakładem produkcyjnym.

Jej głównymi funkcjonalnościami są:

- zgłaszanie awarii usterek w parku maszynowym,
- planowanie przeglądów prewencyjnych,
- dostęp do historii napraw maszyn i urządzeń,
- przydzielanie zleceń pracy/zadań do techników utrzymania ruchu,
- monitorowanie stanów magazynowych części zamiennych,
- śledzenie kluczowych wskaźników utrzymywania ruchu.

Zalety aplikacji:

- lista aktywności/zadań z informacjami o przekroczonych terminach,
- system filtrujący informacje w tabelach,
- kalendarz nadchodzących oraz miniętych aktywności,
- możliwość dodania wielu lokalizacji,
- system logowania i rejestracji,
- umożliwienie komunikacji użytkowników poprzez system,
- analiza kosztów na maszynę,
- udostępnione poradniki do używania oprogramowania,
- dodawania części, maszyn, lokalizacji, pracowników, dostawców,
- udostępniona wersja próbna systemu,
- ciągłe aktualizacje systemu,
- dba o bezpieczeństwo danych.

Wady aplikacji:

- nadmiar informacji i nie potrzebnych funkcjonalności,
- trudna w obsłudze dla osób mających małą styczność z technologią,
- wymagane wdrożenie systemu trwające zależnie od wielkości zakładu,
- brak funkcjonalności monitorowania ilości paliwa w zbiornikach na terenie zakładu,
- konieczność dodania lokalizacji przed dodaniem maszyn do systemu,
- wysoka cena – system nie jest do kupienia za jednorazową umówioną wpłatą, działanie systemu zależy od wybrania jednego z 3 planów

posiadających różne funkcjonalności ostatecznie nie odblokowując całego potencjału aplikacji, jeśli nie zostanie zakupiona wersja „Premium”

Analiza aplikacji do zarządzania parkiem maszynowym pozwoliła na stworzenie wstępnego konceptu projektu, pomogła zrozumieć ważne oraz zbędne elementy systemu, jak i zauważyć te brakujące. Przedstawione oprogramowanie może jedynie pomóc w ewentualnym dalszym rozwoju aplikacji, a do początkowej wersji, została użyta wyłącznie jako inspiracja do zrozumienia działania poszczególnych funkcjonalności.

1.3 Technologie wspomagające budowanie aplikacji mobilnej

W tej części zostaną przedstawione technologie użyte w trakcie tworzenia projektu.

Microsoft SQL Server to relacyjny system zarządzania bazami danych opracowany przez Microsoft. Jest popularnym produktem do zarządzania i przechowywania danych dla aplikacji i usług webowych, który oferuje wiele funkcji, takich jak wysoka wydajność, skalalność, bezpieczeństwo i łatwość użytkowania. Udostępnia narzędzia do tworzenia i administrowania bazami danych, jak SQL Server Management Studio (SSMS) oraz poprzez szereg klas sterowników pozwala na połączenie swojej funkcjonalności z Android Studio.

Android Studio jest narzędziem programistycznym dla systemu Android. Jest to środowisko programistyczne (IDE) zaprojektowane specjalnie do tworzenia aplikacji mobilnych dla systemu Android. Android Studio zapewnia narzędzia do projektowania interfejsu użytkownika, debugowania, testowania i publikowania aplikacji na Google Play. Oferuje także integrację z Android SDK, co umożliwia łatwe i szybkie tworzenie aplikacji dla różnych urządzeń i wersji systemu Android.

Java to obiektowy, wieloplatformowy język programowania. Jest to jeden z najpopularniejszych języków programowania na świecie i jest używany do tworzenia aplikacji desktopowych, serwerowych, mobilnych i gier. Java jest wysoce zorientowana na obiekty, co umożliwia łatwą modyfikację i rozszerzanie kodu. Język programowania Java jest wspierany przez środowisko programistyczne Android Studio.

Git to system kontroli umożliwiający zarządzanie historią zmian w kodzie źródłowym. Git pozwala na śledzenie i przechowywanie wersji kodu, łatwą integrację zmian wywoływanych przez różnych użytkowników oraz możliwość łatwego tworzenia i przełączania się między różnymi gałęziami kodu. Android studio posiada również wtyczkę Gita, która umożliwia łatwe i szybkie aktualizacje projektu

GitHub to platforma internetowa oparta na systemie Git, która umożliwia umieszczanie projektów opartych na Git. GitHub jest bardzo popularnym narzędziem dla programistów i oferuje funkcje, takie jak zarządzanie zadaniami, współdzielenie kodu czy tworzenie i śledzenie problemów.

1.4 Podsumowanie części teoretycznej

Części teoretyczna rozdziału miała za zadanie przybliżenia i analizy trendów technologicznych oraz użytkowych charakterystycznych dla aplikacji mobilnych. Owa dedukcja pomogła w wybraniu odpowiedniego systemu operacyjnego oraz jego najbardziej używanej wersji poprzez analizę grafów wykresów oraz informacji z serwisów o tematyce aplikacji mobilnych. Zbadana została również architektura wybranego systemu w celu lepszego zrozumienia jego działania, co ułatwiło następnie pracę nad projektem. Przedstawione oraz opisane zostały także technologie używane podczas tworzenia aplikacji. Ostatecznie powołując się na przykład podobnego systemu z rynku, zademonstrowano jego funkcjonalności, zalety oraz wady co pomogło w ustaleniu konceptu i zamysłu wytwarzanej aplikacji.

2 Problematyka tworzenia aplikacji zarządzającej parkiem maszynowym

W tym rozdziale zostanie nakreślona problematyka zarządzania parkiem maszynowym oraz wymagania systemu, który mógłby ułatwić pracę nad parkiem maszynowym. Dzięki temu określone zostaną wymagania funkcjonalne i нефункционалне projektu. Rozdział posiada również opisany diagram przypadków użycia oraz modele procesów zachodzących w aplikacji.

2.1 Określenie problemu zarządzania parkiem maszynowym

Przed rozpoczęciem pracy nad aplikacją, ważnym jest ustalenie jej potrzeb oraz problemów z jakimi zmagają się jej użytkownicy. Analizę warto zacząć od zidentyfikowania odbiorców aplikacji co następnie ułatwi pobieranie informacji na ten temat. Najłatwiejszymi formami poznania potrzeb zarządzania parkiem maszynowym jest rozmowa z kompetentną w tej dziedzinie osobą lub własne jego doświadczenie. Autor udał się w tym celu do gospodarstwa rolnego, przeprowadził dogłębną konwersację z osobą osadzoną na odpowiednim stanowisku, zobaczył z czym musi się ona zmagać, jakie ma obowiązki, czego potrzebuje do ułatwienia mu pracy i ostatecznie wysnuł następujące wnioski:

- użytkownik powinien mieć dostęp do informacji na temat parku maszynowego z dowolnego miejsca,
- użytkownik powinien mieć możliwość monitorowania informacji na temat dostępnych maszyn znajdujących się na terenie gospodarstwa,
- użytkownik musi posiadać możliwość kontrolowania ilości paliwa w zbiornikach znajdujących się na terenie ośrodka,
- użytkownik powinien posiadać możliwość przechowywania informacji na temat dostaw paliw do zbiorników oraz zatankowań maszyn,
- użytkownik powinien mieć spis i możliwość archiwizacji prac związanych z obsługą techniczną maszyn takich jak przeglądy, naprawy i wymiany olejów,
- użytkownik powinien posiadać listę zamówień pracowników dotyczących napraw lub ulepszeń maszyn,

- użytkownik powinien mieć możliwość łatwego sprawdzenia numerów telefonów kierowców oraz ich identyfikatorów w gospodarstwie.

2.2 Wymagania funkcjonalne i niefunkcjonalne

Na podstawie wcześniej wysnutych wniosków na temat problematyki zarządzania parkiem maszynowym, opisano wymagania funkcjonalne oraz niefunkcjonalne aplikacji.

Wymagania funkcjonalne:

- przeglądanie listy maszyn – użytkownik powinien mieć wgląd we wszystkie maszyny w gospodarstwie i ich specyfikacje,
- aktualizowanie dat przeglądów maszyn – użytkownik powinien mieć możliwość zmiany daty przeglądu w aplikacji, gdy nastąpi jej zmiana w realnym życiu, aby dane były zawsze aktualne,
- przeglądanie listy zatankowań maszyn – użytkownik powinien posiadać możliwość wglądu w dokonane zatankowania maszyn,
- dodawanie nowego tankowania – użytkownik powinien mieć możliwość dodawania nowych zatankowań, w celu monitorowania ilości paliwa w zbiornikach z ciecżą znajdujących się na terenie gospodarstwa,
- przeglądanie listy zamówień pracowników gospodarstwa – użytkownik powinien posiadać listę zamówień użytkowników,
- dodawanie nowego zamówienia – użytkownik powinien mieć możliwość dodania nowego zamówienia, gdy jest potrzeba kupna nowego przedmiotu/części do maszyny,
- usuwanie dowolnego zamówienia – użytkownik może mieć możliwość usuwania zamówienia z listy, w momencie w którym zamówienie nie jest już aktualne, zaszła pomyłka lub przedmiot został już kupiony i dostarczony odpowiedniej osobie,
- aktualizowanie dowolnego zamówienia – użytkownik powinien posiadać możliwość aktualizacji dowolnego zamówienia w celu dodania do niej ceny oraz daty odbioru przedmiotu,

- przeglądanie listy wymian części oraz olejów – użytkownik powinien mieć dostęp do listy wymian części lub olejów zachodzących w maszynach z gospodarstwa,
- dodawanie nowej wymiany części oraz oleju – użytkownik powinien posiadać możliwość dodawania nowych wymian aby lepiej monitorować zachodzące zmiany maszyn,
- usuwanie dowolnej wymiany – użytkownik może mieć możliwość usuwania dowolnej wymiany w przypadku gdy wymiana nie jest aktualna lub zaszła jakaś pomyłka,
- przeglądanie listy zbiorników – użytkownik powinien posiadać możliwość monitorowania ilości paliwa znajdującego się w zbiornikach na gospodarstwie,
- przeglądanie listy dostaw paliwa dowolnego zbiornika – użytkownik powinien mieć dostęp do listy dostaw paliwa dostarczonego do każdego zbiornika,
- dodawanie nowej dostawy do dowolnego zbiornika – użytkownik powinien posiadać funkcjonalności dodawania nowych dostaw, aby móc zmienić ilość paliwa w zbiorniku na tą rzeczywistą,
- przeglądanie listy kierowców w gospodarstwie – użytkownik powinien posiadać możliwość wglądu w listę kierowców gospodarstwa jak i ich numery telefonów w celu łatwiejszego kontaktu,
- edycja dowolnego kierowcy – użytkownik powinien posiadać funkcjonalność do edycji dowolnego kierowcy w razie zmiany jego rzeczywistego numeru telefonu, imienia lub nazwiska.

Wymagania нефункционалне:

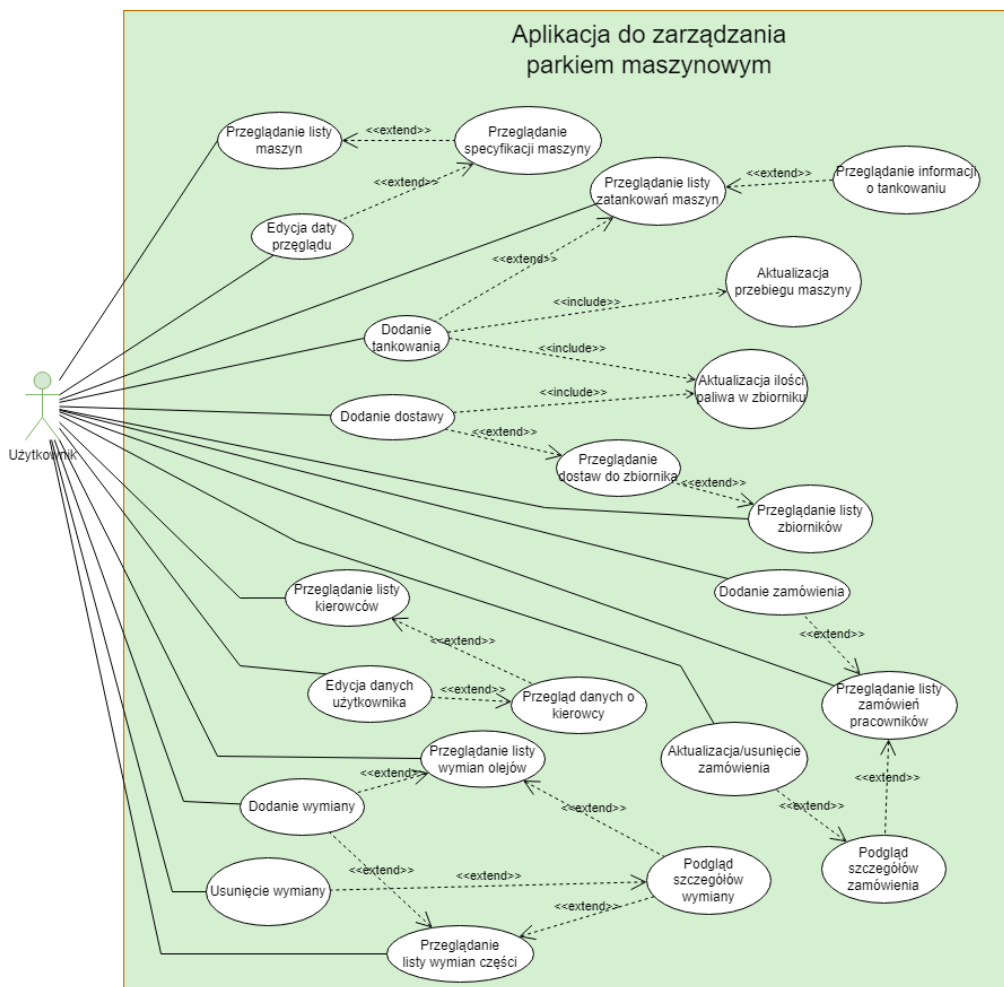
- Kompatybilność aplikacji z odpowiednimi wersjami systemu Android – system powinien być kompatybilny z urządzeniami z systemem Android w wersji 8.0 lub wyżej,
- Dostęp do Internetu – system powinien mieć dostęp do połączenia internetowego, w celu poprawnego działania,
- Koordynacja zmian w bazie danych – system powinien mieć połączenie z bazą danych aby pobierać niezbędne informacje,

2.3 Model aplikacji

W tej części pracy zostanie przedstawiony diagram przypadków użycia oraz procesy realizowane przez aplikację w celu lepszego zobrazowania funkcjonalności jakie będzie wykorzystywała ostateczna wersja systemu.

2.3.1 Diagramy przypadków użycia – UML

Rysunek 2.1 Diagram przypadków użycia



Źródło: opracowanie własne.

Przypadek użycia: Przeglądanie listy maszyn

Aktor: Użytkownik

Opis: Wyświetlenie listy maszyn

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu głównego.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Maszyny”
2. System wyświetla listę wszystkich maszyn

Przypadek użycia: Przeglądanie specyfikacji maszyny

Aktor: Użytkownik

Opis: Wyświetlenie specyfikacji maszyny

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy maszyn.

Przebieg:

1. Użytkownik klika w dowolną maszynę z listy
2. System wyświetla wszystkie informacje o wybranej maszynie

Przypadek użycia: Edycja daty przeglądu

Aktor: Użytkownik

Opis: Edycja daty przeglądu wybranej maszyny

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy maszyn.

Przebieg:

1. Użytkownik klika w dowolną maszynę z listy
2. Użytkownik klika w przycisk „Edytuj datę”
3. Użytkownik zostaje przeniesiony do widoku aktualizacji daty
4. Użytkownik wybiera datę z kalendarza
5. Użytkownik zatwierdza zmiany klikając w przycisk
6. System aktualizuje naniesione zmiany. Użytkownik zostaje przeniesiony do listy maszyn

Przypadek użycia: Przeglądanie listy zatankowań maszyn

Aktor: Użytkownik

Opis: Wyświetlenie listy wszystkich zatankowań

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu głównego.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Zatankuj”
2. System wyświetla listę wszystkich zatankowań

Przypadek użycia: Przeglądanie informacji o tankowaniu

Aktor: Użytkownik

Opis: Wyświetlenie informacji o tankowaniu

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zatankowań.

Przebieg:

1. Użytkownik klika w dowolne tankowanie z listy
2. System wyświetla informacje dotyczące wybranego tankowania

Przypadek użycia: Dodanie tankowania

Aktor: Użytkownik

Opis: Dodanie nowego zatankowania

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zatankowań.

Przebieg:

1. Użytkownik klika w przycisk „Dodaj nowe zatankowanie”
2. Użytkownik zostaje przeniesiony do formularza dodania nowego zatankowania
3. Użytkownik wypełnia formularz danymi
4. Użytkownik zatwierdza naniesione zmiany klikając w przycisk

5. System dodaje tankowanie do listy zatankowań, aktualizuje ilość paliwa z użytego zbiornika, aktualizuje przebieg tankowanej maszyny. Użytkownik zostaje przeniesiony do listy zatankowań.

Przypadek użycia: Przeglądanie listy zbiorników

Aktor: Użytkownik

Opis: Wyświetlenie listy zbiorników

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu głównego.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Zbiorniki”
2. System wyświetla listę wszystkich zbiorników

Przypadek użycia: Przeglądanie dostaw do zbiorników

Aktor: Użytkownik

Opis: Wyświetlenie listę dostaw

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zbiorników.

Przebieg:

1. Użytkownik klika w dowolny zbiornik z listy
2. System wyświetla listę wszystkich dostaw wybranego zbiornika

Przypadek użycia: Dodanie dostawy

Aktor: Użytkownik

Opis: Dodanie dostawy do zbiornika

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy dostaw dowolnego zbiornika

Przebieg:

1. Użytkownik klika w przycisk „Dodaj nową dostawę”

2. Użytkownik zostaje przeniesiony do formularza dodania nowej dostawy
3. Użytkownik wypełnia formularz danymi
4. Użytkownik zatwierdza naniesione zmiany klikając w przycisk
5. System dodaje nową dostawę, aktualizuje ilość paliwa w odpowiednim zbiorniku,
Użytkownik zostaje przeniesiony do listy zbiorników

Przypadek użycia: Przeglądanie listy zamówień pracowników

Aktor: Użytkownik

Opis: Wyświetlenie listy wszystkich zamówień

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu głównego.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Zamów”
2. System wyświetla listę wszystkich zamówień pracowników

Przypadek użycia: Podgląd szczegółów zamówienia

Aktor: Użytkownik

Opis: Wyświetlenie szczegółów wybranego zamówienia

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zamówień.

Przebieg:

1. Użytkownik klika w dowolne zamówienia z listy
2. System wyświetla szczegóły wybranego zamówienia

Przypadek użycia: Dodanie zamówienia

Aktor: Użytkownik

Opis: Dodanie nowego zamówienia

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zamówień.

Przebieg:

1. Użytkownik klika w przycisk „Dodaj nowe zamówienia”
2. Użytkownik zostaje przeniesiony do formularza dodania nowego zamówienia
3. Użytkownik wypełnia formularz danymi
4. Użytkownik potwierdza zmiany klikając w przycisk
5. System dodaje nowe zamówienie, przenosi użytkownika do listy zamówień

Przypadek użycia: Usunięcie zamówienia

Aktor: Użytkownik

Opis: Usunięcie wybranego zamówienia

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zamówień.

Przebieg:

1. Użytkownik klika w dowolne zamówienia z listy
2. Użytkownik wybiera przycisk usunięcia zamówienia
3. System usuwa zamówienia i aktualizuje listę zamówień

Przypadek użycia: Aktualizacja zamówienia

Aktor: Użytkownik

Opis: Aktualizacja wybranego zamówienia

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy zamówień.

Przebieg:

1. Użytkownik klika w dowolne zamówienia z listy
2. Użytkownik wybiera przycisk aktualizacji zamówienia
3. Użytkownik zostaje przeniesiony do formularza aktualizacji wybranego zamówienia z wypełnionymi polami
4. Użytkownik zmienia/wypełnia dowolne pola nowymi danymi

5. System aktualizuje informacje o zamówieniu, przenosi użytkownika do listy zamówień

Przypadek użycia: Przeglądanie listy kierowców

Aktor: Użytkownik

Opis: Wyświetlenie listy wszystkich kierowców

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu głównego.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Kierowcy”
2. System wyświetla listę wszystkich kierowców

Przypadek użycia: Przeglądanie danych o kierowcach

Aktor: Użytkownik

Opis: Wyświetlenie danych wybranego kierowcy

Warunki wstępne: Połączenie z bazą danych. Otwarty widok listy kierowców.

Przebieg:

1. Użytkownik klika w dowolnego kierowcę z listy
2. Użytkownik zostaje przeniesiony do nowego widoku z danymi o wybranym kierowcy

Przypadek użycia: Edycja danych użytkownika

Aktor: Użytkownik

Opis: Edycja danych wybranego użytkownika

Warunki wstępne: Połączenie z bazą danych. Otwarty widok edycji dowolnego kierowcy.

Przebieg:

1. Użytkownik klika w przycisk

2. Użytkownik zostaje przeniesiony do wypełnionego danymi formularza wybranego kierowcy
3. Użytkownik zmienia dowolne pola
4. Użytkownik klika w przycisk do aktualizacji danych
5. System aktualizuje dane o kierowcy, przenosi użytkownika do listy kierowców

Przypadek użycia: Przeglądanie listy wymian części

Aktor: Użytkownik

Opis: Wyświetlenie listy wszystkich wymian części

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu wymian.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Części”
2. System wyświetla listę wszystkich wymian części

Przypadek użycia: Przeglądanie listy wymian olejów

Aktor: Użytkownik

Opis: Wyświetlenie listy wszystkich wymian olejów

Warunki wstępne: Połączenie z bazą danych. Otwarty widok menu wymian.

Przebieg:

1. Użytkownik klika w kafelek podpisany „Oleje”
2. System wyświetla listę wszystkich wymian olejów

Przypadek użycia: Podgląd szczegółów wymiany

Aktor: Użytkownik

Opis: Wyświetlenie szczegółów wybranej wymiany

Warunki wstępne: Połączenie z bazą danych. Otwarty widok wymian olejów lub części

Przebieg:

1. Użytkownik wybiera dowolną wymianę z listy
2. System wyświetla szczegóły wybranej wymiany

Przypadek użycia: Dodanie wymiany

Aktor: Użytkownik

Opis: Dodanie nowej wymiany

Warunki wstępne: Połączenie z bazą danych. Otwarty widok wymian olejów lub części

Przebieg:

1. Użytkownik wybiera przycisk „Dodaj nową wymianę”
2. Użytkownik zostaje przeniesiony do formularza dodania nowej wymiany
3. Użytkownik wypełnia formularz
4. Użytkownik zatwierdza zmiany klikając w przycisk
5. System dodaje nową wymianę, przenosi użytkownika do listy wymian

Przypadek użycia: Usunięcie wymiany

Aktor: Użytkownik

Opis: Usunięcie wybranej wymiany

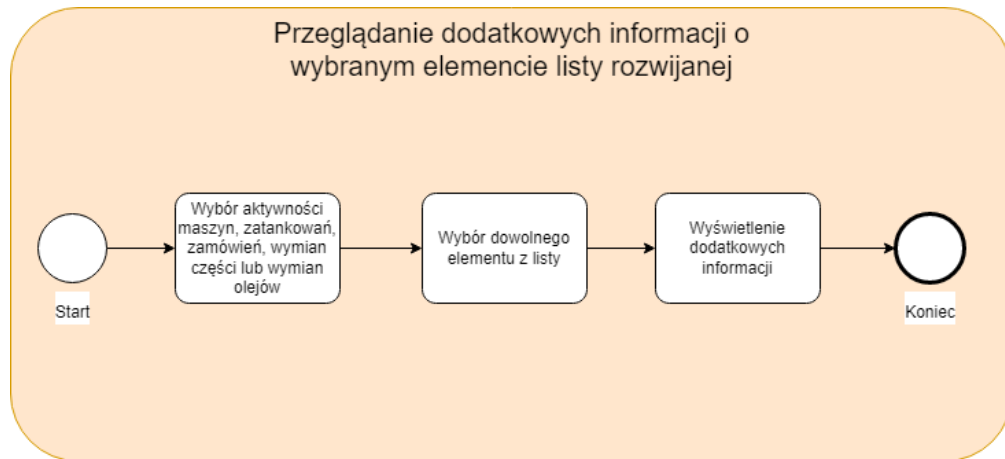
Warunki wstępne: Połączenie z bazą danych. Otwarty widok wymian olejów lub części

Przebieg:

1. Użytkownik wybiera dowolną wymianę z listy
2. Użytkownik klika w przycisk usunięcia wymiany
3. System usuwa wymianę i aktualizuje listę wymian

2.3.2 Modele procesów realizowanych przez aplikację

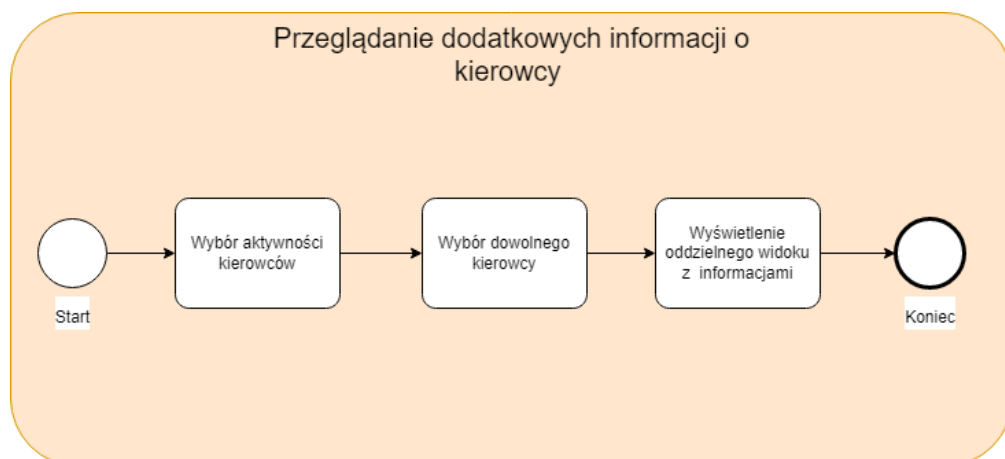
Rysunek 2.2 Model procesu przeglądania dodatkowych informacji o elemencie listy



Źródło: opracowanie własne.

Proces rozpoczyna się od wybrania żądanej aktywności z pośród wymienionych, następnie wybór dowolnego elementu z jej listy. Proces kończy się wyświetleniem dodatkowych informacji na temat wybranego elementu.

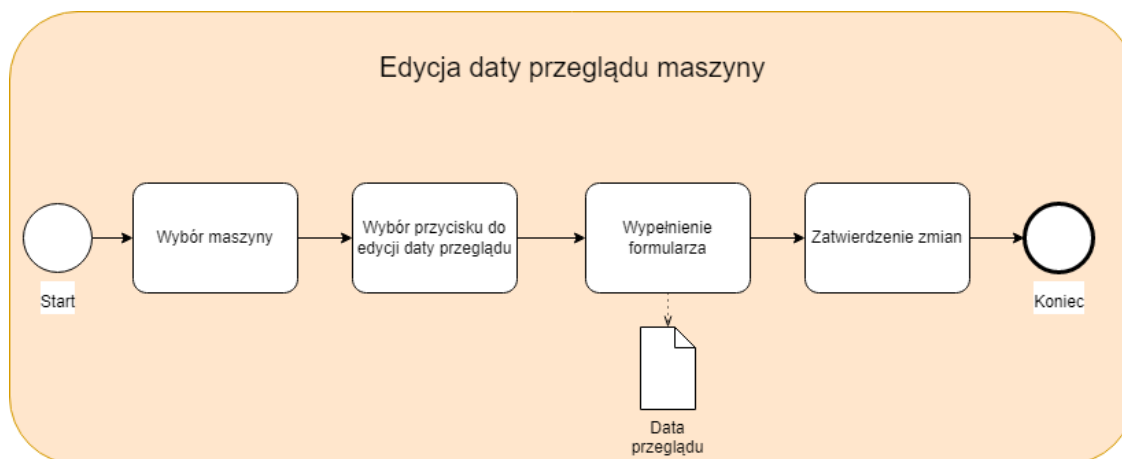
Rysunek 2.3 Model procesu przeglądania dodatkowych informacji o kierowcy



Źródło: opracowanie własne.

Proces rozpoczyna się od wybrania aktywności kierowców, a następnie wybrania dowolnego kierowcy z listy, co doprowadza do wyświetlenia oddzielnego widoku z informacjami o wybranym kierowcy.

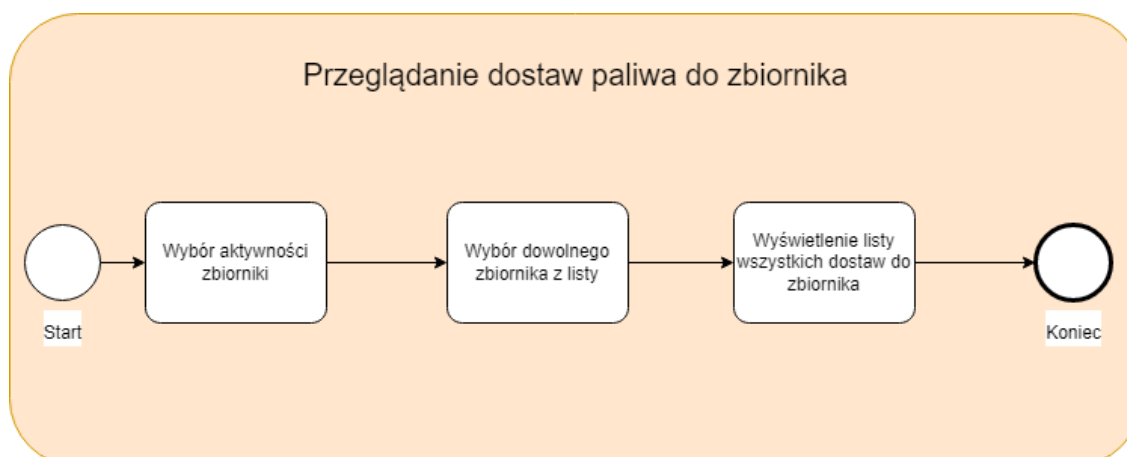
Rysunek 2.4 Model procesu edycji daty przeglądu maszyny



Źródło: opracowanie własne.

Proces rozpoczyna się od wyboru maszyny, w której będzie odbywała się zmiana, następnie wybór przycisku wyświetla formularz w którym należy wypełnić pole daty i zatwierdzić zmiany.

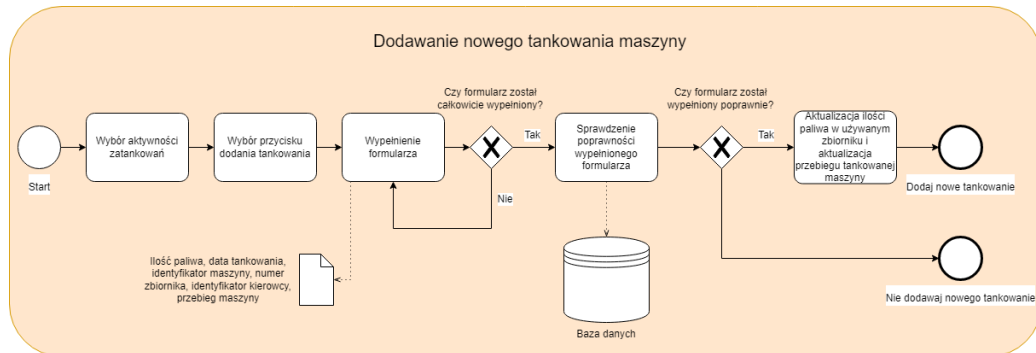
Rysunek 2.5 Model procesu przeglądania dostaw paliwa do wybranego zbiornika



Źródło: opracowanie własne.

Proces rozpoczyna się od wyboru aktywności zbiorników w menu głównym, następnie wyboru dowolnego zbiornika z listy. Skutkuje to wyświetleniem wszystkich dostaw do wybranego zbiornika.

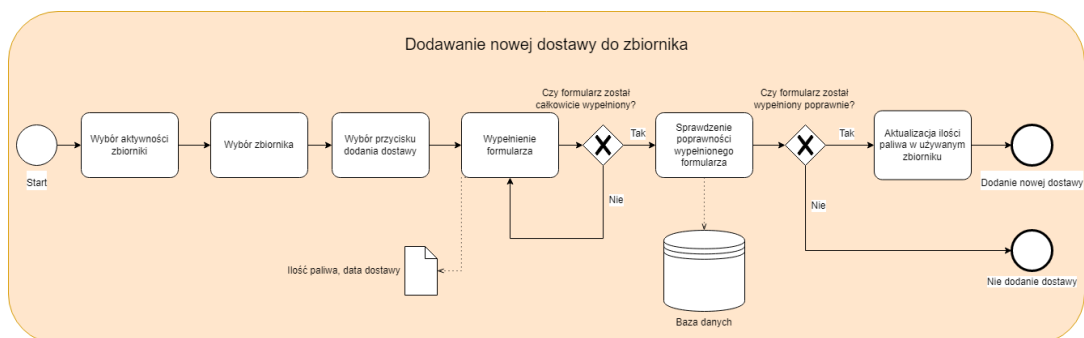
Rysunek 2.6 Model procesu dodania nowego tankowania maszyny



Źródło: opracowanie własne.

Proces rozpoczyna się od wyboru aktywności zatankuj, w którym wybierany jest przycisk do dodania nowego tankowania co wyświetla formularz. Następnie zostaje wypełniony formularz. Jeśli formularz nie został całkowicie wypełniony, należy ponownie wypełnić formularz, jeśli został wypełniony, następuje sprawdzenie poprawności wypełnionych danych (czy paliwo zatankowane nie przekracza pojemności zbiornika maszyny, czy przebieg maszyny nie jest mniejszy od poprzedniego, czy ilość paliwa zatankowanego nie przekracza aktualnej ilości cieczy znajdującej się w rezerwuarze). Jeżeli wszystko się zgadza następuje aktualizacja paliwa w zbiorniku i przebiegu maszyny i nowe tankowanie zostaje dodane, jeśli jednak nie, nic nie ulega zmianie i tankowanie nie dodaje się.

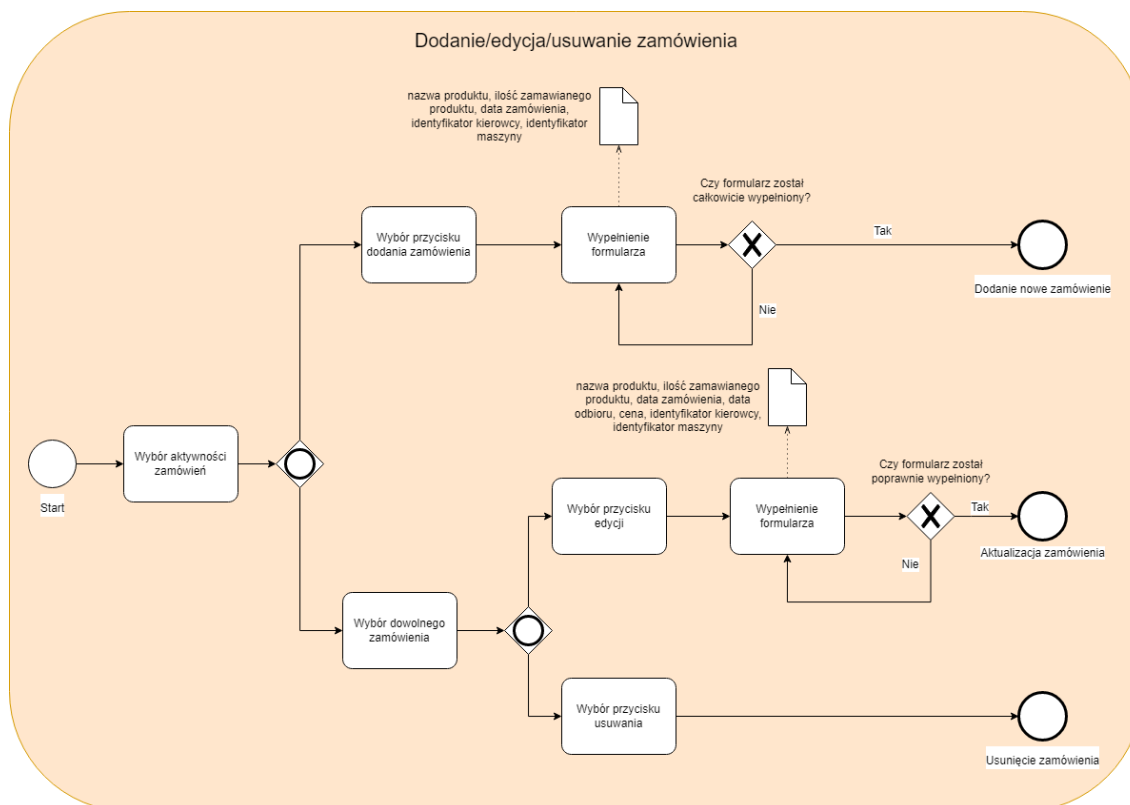
Rysunek 2.7 Model procesu dodania nowej dostawy do zbiornika



Źródło: opracowanie własne.

Proces zaczyna się od wyboru aktywności zbiornika, wyboru odpowiedniego zbiornika z listy, wyboru przycisku dodania dostawy, po czym następuje wypełnienie formularza, na którym zostaje przeprowadzona funkcja sprawdzająca czy cały formularz został wypełniony oraz czy został wypełniony poprawie. W przypadku błędnego wypełnienia formularza, dostawa nie zostaje dodana.

Rysunek 2.8 Model procesu dodania/edycji/usuwania zamówienia



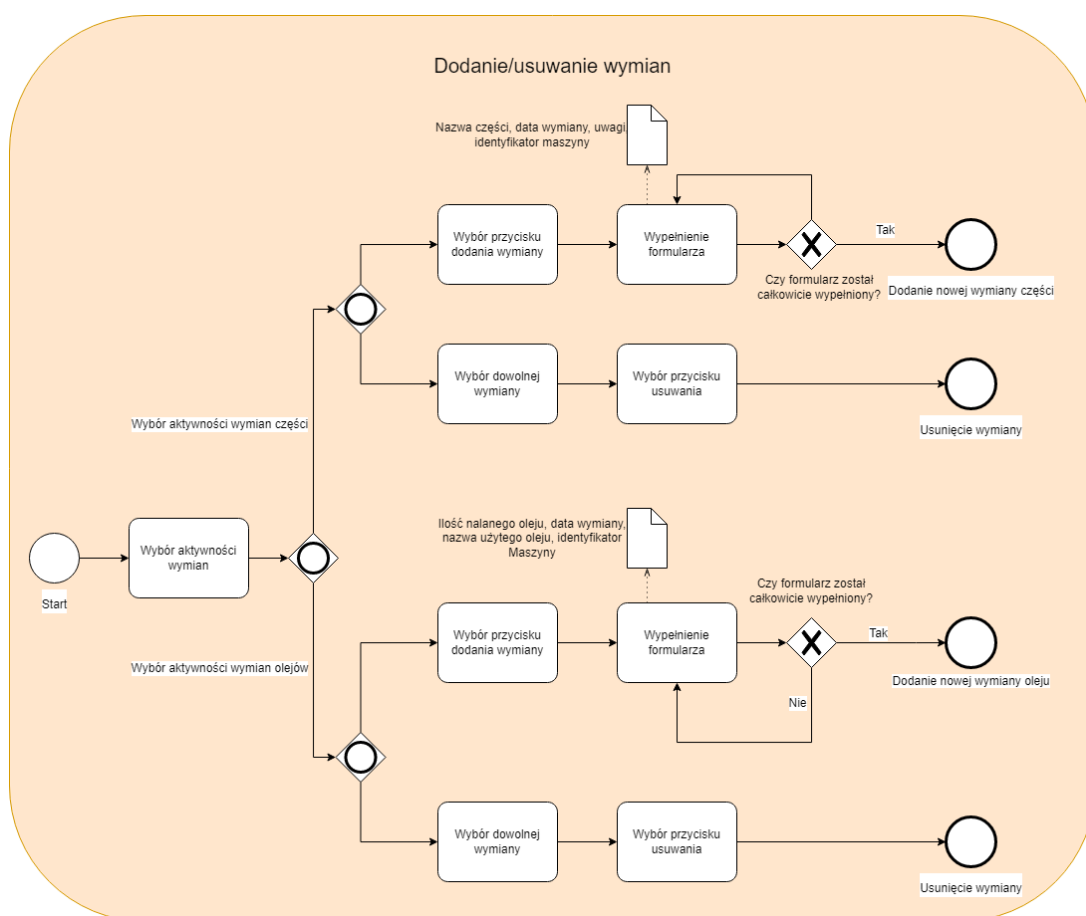
Źródło: opracowanie własne.

Proces rozpoczyna się od wyboru aktywności zamówienia. W przypadku dodania nowej dostawy należy wybrać przycisk dodania zamówienia i wypełnić formularz. Jeżeli formularz został całkowicie wypełniony, zostaje dodane nowe zamówienie.

W przypadku aktualizacji zamówienia, należy dodatkowo wybrać dowolne zamówienie z listy, a następnie przycisnąć przycisk do aktualizacji. Wypełniony formularz następnie zostaje sprawdzony pod względem jego poprawnego wypełnienia. Następnie dochodzi do aktualizacji wybranego zamówienia.

Do usunięcia dochodzi poprzez wybranie dowolnego zamówienia i wybrze przycisku do jego usunięcia.

Rysunek 2.9 Model procesu dodania/usuwania wymian części i olejów



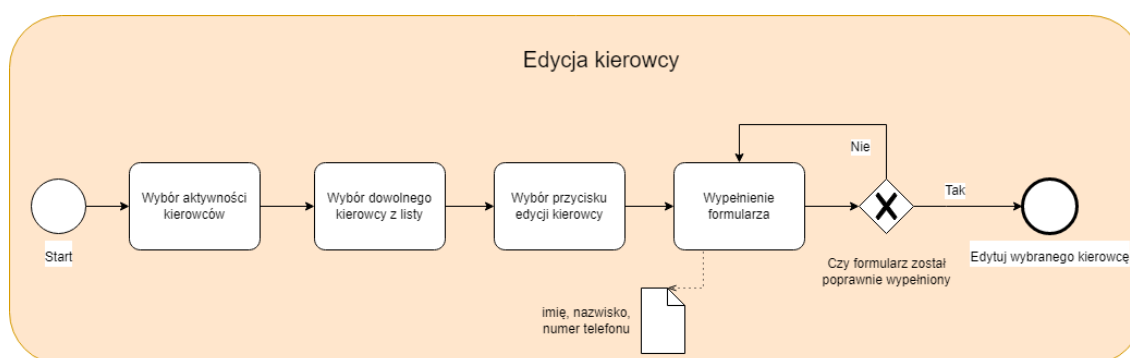
Źródło: opracowanie własne.

Proces rozpoczyna się od wyboru aktywności wymian, następnie, aby dodać nową wymianę części należy przejść do aktywności z listą wymian części. Dalej należy wybrać przycisk dodania wymiany i wypełnić formularz. Ostatecznie sprawdzane jest czy formularz został całkowicie wypełniony, który w przypadku błędu prosi o wypełnienie go ponownie.

Usuwanie wymiany części rozpoczyna się od wyboru aktywności wymian części. Następnie wybrana zostaje dowolna wymiana z listy. Ostatecznie proces kończy wybór przycisku usunięcia.

Adekwatnie do przedstawionych procesów działają wymiany olejów. Zmienia się jedynie początkowe wejście do aktywności wymian olejów.

Rysunek 2.10 Model procesu edycji kierowcy



Źródło: opracowanie własne.

Proces zaczyna się od wyboru aktywności kierowców, a następnie wyboru dowolnego kierowcy z listy. Wybór przycisku do edycji wywołania wypełnienia formularza. Po zmiana odpowiednich danych, system sprawdza czy formularz został poprawnie wypełniony, w przypadku gdy nie został należy ponownie wypełnić formularz. Natomiast jeśli formularz został poprawnie wypełniony wybrany kierowca zostaje edytowany.

2.4 Podsumowanie części problemowej

Rozdział opisuje problem zarządzania parkiem maszynowym na podstawie rozmowy z kompetentną osobą i doświadczeń autora. Zostają wysnute wnioski ułatwiające zrozumienie problemu oraz wyłonione zostają funkcjonalności, które powinna posiadać ostateczna wersja aplikacji. Do lepszego przedstawienia działania produktu powstał również diagram przypadków użycia oraz modele procesów w nim zachodzące. Wszystkie opisane części tego rozdziału pomogły w łatwiejszym i szybszym tworzeniu aplikacji mobilnej.

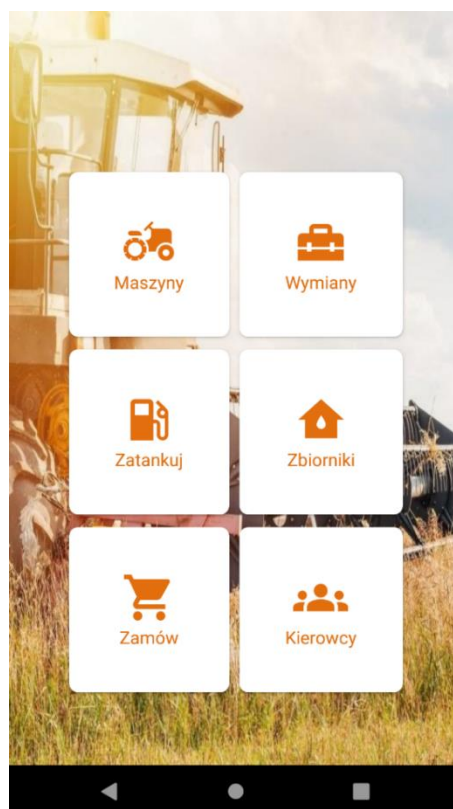
3 Projekt front-endu (interfejsu) aplikacji do zarządzania parkiem maszynowym w gospodarstwie rolnym

W tym rozdziale zostały przedstawione makiety interfejsów aplikacji. Każdy podrozdział prezentuje i opisuje ostateczne widoki systemu, jak również ich funkcjonalności.

3.1 Widok menu głównego

Menu główne składa się tematycznego tła oraz 6 „kafelków” okraszonych w nazwy i małe ikonki, które w przejrzysty sposób informują użytkownika do jakiego widoku zostanie on przeniesiony po kliknięciu w jeden z nich, a są nimi maszyny, wymiany, zatankuj, zbiorniki, zamów oraz kierowcy.

Rysunek 3.1 Widok menu głównego aplikacji



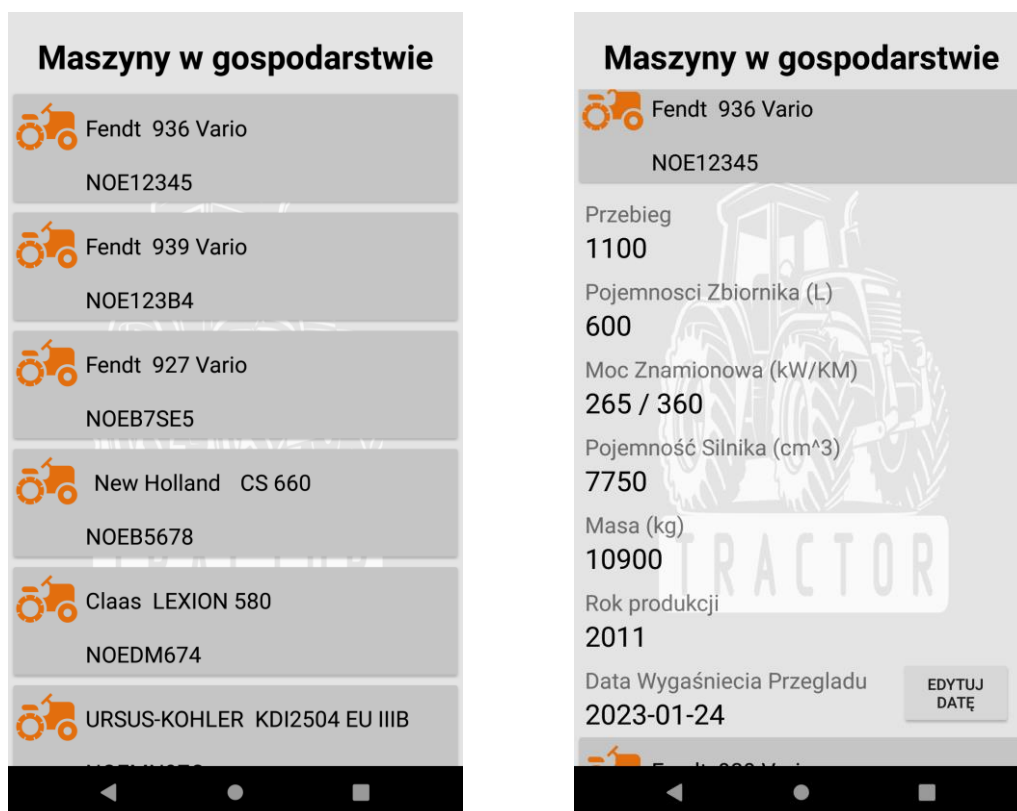
Źródło: opracowanie własne.

3.2 Widok maszyn i ich edycji

Widok (Rysunek 3.2) zawiera nazwę identyfikującą kafelek w górnej części ekranu, jaki został wcześniej kliknięty, listę, z możliwością przewijania w górę lub w dół, wszystkich maszyn w gospodarstwie, podpisanych marką, modelem i numerem rejestracji, oraz przejrzyste tło. Kliknięcie w maszynę powoduje rozwinięcie pozostałych jej parametrów i guzika służącego do aktualizacji przebiegu rozwiniętej maszyny. Szczegóły pojazdu jakie zostają wyświetlone to:

- aktualny przebieg,
- pojemność zbiornika,
- pojemność silnika,
- masą całkowitą,
- rok produkcji,
- data wygaśnięcia przeglądu.

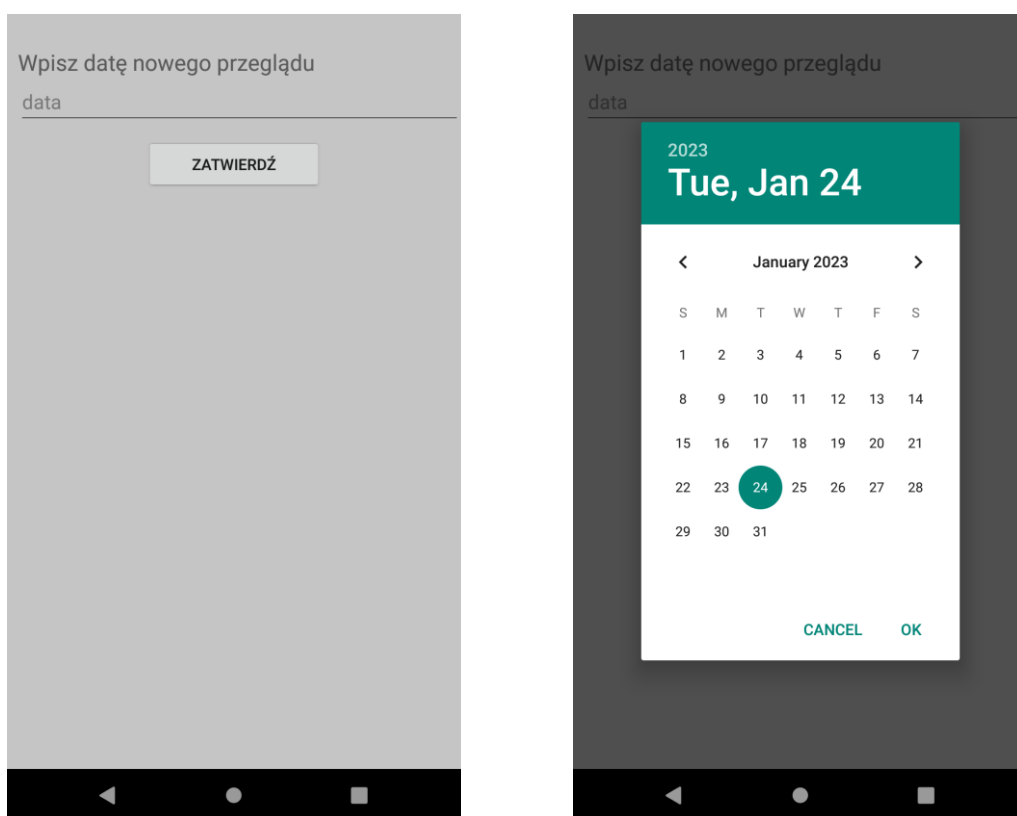
Rysunek 3.2 Widok maszyn



Źródło: opracowanie własne.

W momencie kliknięcia przycisku „Edytuj datę”, odbiorca zostanie przeniesiony do panelu odpowiadającego zmianie daty przeglądu (). Został w nim umieszczony podpis zmienianej wartości, pole po którego kliknięciu skutkuje w pojawieniu się kalendarza, oraz przycisk zatwierdzający zmiany oraz przenoszący użytkownika z powrotem do listy maszyn.

Rysunek 3.3 Zmiana daty przeglądu maszyny

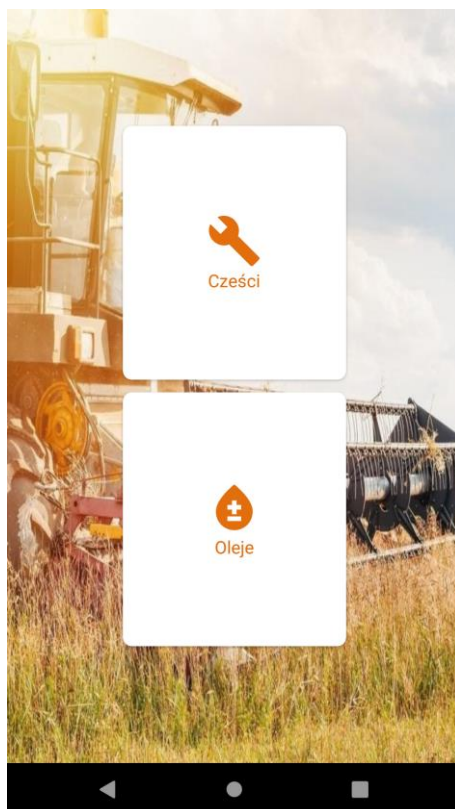


Źródło: opracowanie własne.

3.3 Widok wymian

Widok (Rysunek 3.4) składa się z kolejnych dwóch kafelków z odpowiednio podpisanymi ikonkami, adekwatnie przenoszącymi użytkownika do wymian części lub wymian olejów.

Rysunek 3.4 Widok wymian



Źródło: opracowanie własne.

3.4 Widok wymian olejów

Poniższy rysunek (Rysunek 3.5) przedstawia nazwę panelu, przycisk służący dodawaniu nowej wymiany oleju oraz rozwijaną listę wszystkich zadeklarowanych rekordów, która przed kliknięciem w wybrany element pokazuje nazwę wymienianego oleju oraz datę jej przeprowadzenia, natomiast po, przycisk do usunięcia danej wymiany oraz wszystkie szczegóły jej dotyczące.

Rysunek 3.5 Wymiany olejów

Wymiany Oleju

DODAJ NOWĄ WYMIANĘ

Nazwa Oleju	Data
Fendt Extra Trans 10W/40	2022-10-12

Wymiana w maszynie:
Fendt 936 Vario

Przebieg maszyny w momencie wymiany
1100

Ilość wlanego oleju (L)
10

USUŃ

Nazwa Oleju	Data
HYDRAL STOU 10W40	2022-10-12

Nazwa Oleju	Data
Fendt Extra Trans 10W/40	2022-06-20

Nazwa Oleju	Data
-------------	------

Źródło: opracowanie własne.

Ilustracja (Rysunek 3.6) przedstawia, w górnej części ikonę identyfikującą widok, niżej listę namiarów do wypełnienia, a pod tym przycisk zatwierdzający operację i przenoszący użytkownika do widoku (Rysunek 3.5). Widok pojawia się za pośrednictwem naciśnięcia przycisku nazwanego „Dodaj nową wymianę”, z poprzedniej aktywności (Rysunek 3.5). Widok uwzględnia wszystkie edytowalne pola potrzebne do dodania nowej wymiany oleju, którymi są:

- ilość nalanego oleju,
- data wymiany,
- nazwę oleju,
- unikatowy identyfikator maszyny.

Pola przyjmują różne typy wartości, przez co zostały zaimplementowane odpowiednie dla nich klawiatury numeryczne, tekstowe lub kalendarze, aby pomóc użytkownikowi w wypełnieniu formularza bez wcześniejszego ich sprawdzenia. Rysunek (Rysunek 3.7) pokazuje pojawienie się klawiatury tekstowej po wyborze pola

edycji „nazwa”, (Rysunek 3.8) prezentuje odpowiedni dla typu wartości widok po kliknięciu w „ilość” lub „id”, natomiast (Rysunek 3.9) ilustruje okno kalendarzu po wcześniejszym wyborze pola „Data wymiany”.

Rysunek 3.6 Widok dodania nowej wymiany

Rysunek 3.7 Widok edycji pola tekstowego

Rysunek 3.8 Widok edycji pola numerycznego

Rysunek 3.9 Widok edycji daty

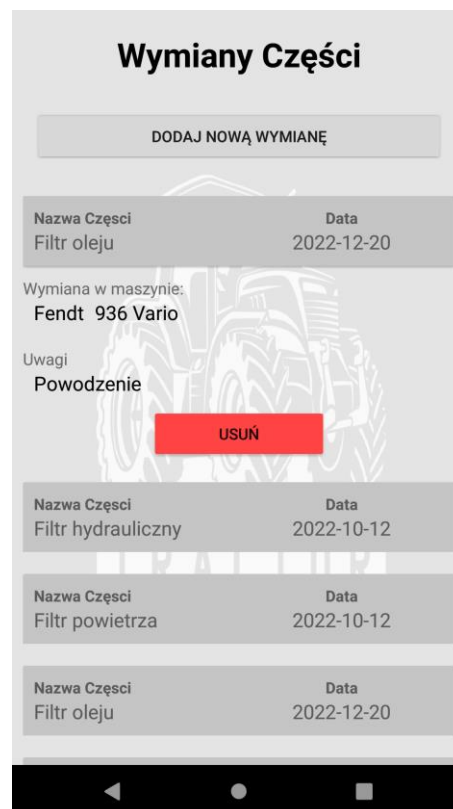
Źródło: opracowanie własne.

3.5 Widok wymian części

Widok (Rysunek 3.10) zawiera nazwę panelu, aby użytkownik nie miał wątpliwości gdzie się znajduje, przycisk służący dodawaniu nowej wymiany części oraz rozwijaną listę wszystkich wymian, która przed kliknięciem w wybrany element pokazuje nazwę wymienianej części oraz datę jej przeprowadzenia, natomiast po, przycisk do usunięcia danej operacji oraz wszystkie szczegóły jej dotyczące. Składają się na nie:

- marka i model maszyny, w której zaszła wymiana,
- uwagi dotyczące wymiany.

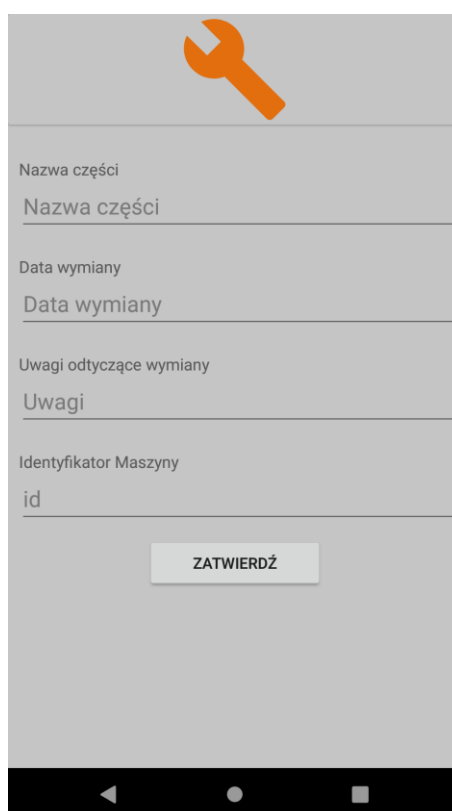
Rysunek 3.10 Wymiany części



Źródło: opracowanie własne.

Rysunek (Rysunek 3.11) przedstawia wygląd widoku dodania nowej wymiany części. Podobnie jak w nowej wymianie oleju (Rysunek 3.6), odpowiednie pola skutkują w pojawieniu się klawiatury numerycznej, tekstowej lub kalendarza. Pierwszy z nich wyświetla się gdy zostanie kliknięte „id”, drugi w przypadku naciśnięcia „Nazwy części” lub „Uwagi”, a trzeci tak jak na ilustracji (Rysunek 3.9) „Daty wymiany”. Ponownie, zatwierdzenie i przejście do panelu z wszystkimi wymianami następuje po wybraniu przycisku „Zatwierdź”.

Rysunek 3.11 Widok dodania nowej wymiany części



Nazwa części
Nazwa części

Data wymiany
Data wymiany

Uwagi odtyczące wymiany
Uwagi

Identyfikator Maszyny
id

ZATWIERDŹ

Źródło: opracowanie własne.

3.6 Widok tankowania

Pokazany poniżej widok (Rysunek 3.12) znowu przedstawia nazwę panelu, przycisk służący dodawaniu nowego tankowania pojazdu, oraz rozwijaną listę wszystkich operacji napełniania zbiorników maszyn. Na wszystkie informacje o poszczególnym tankowaniu składa się:

- imię, i nazwisko kierowcy pojazdu do którego zostaje nalewane paliwo,
- ilość zatankowanego paliwa,
- data tankowania,
- marka, model oraz przebieg maszyny,
- numer użytego zbiornika.

Rysunek 3.12 Widok tankowania wszystkich maszyn w gospodarstwie

Tankowanie

DODAJ NOWE TANKOWANIE

Imie	Nazwisko	Ilosc paliwa(L)	Data
Jakub	Krzykwa	100	2023-01-15

Marka
Fendt

Model
936 Vario

Przebieg maszyny przed tankowaniem
1100

Numer użytego zbiornika
1

Imie	Nazwisko	Ilosc paliwa(L)	Data
Jakub	Krzykwa	100	2023-01-15

Imie	Nazwisko	Ilosc paliwa(L)	Data
Jakub	Krzykwa	40	2023-01-03

Imie	Nazwisko	Ilosc paliwa(L)	Data
Karol	Jasiński	250	2022-12-20

Źródło: opracowanie własne.

Rysunek 3.13 Widok dodania nowego tankowania



Źródło: opracowanie własne.

Widok dodania nowego tankowania (Rysunek 3.13) podobnie jak dodawanie nowych wymian, przedstawia pola do wypełnienia z odpowiednio, do wartości, wyświetlanymi klawiaturami lub kalendarzem. Jak na wcześniejszych tego typu panelach użytkownik posiada również przycisk do zatwierdzenia i dodania kolejnego zdarzenia, po czym zostaje przeniesiony do zaktualizowanej listy wszystkich zatankowań.

3.7 Widok zbiorników w gospodarstwie

Panel ze zbiornikami (Rysunek 3.14) został przedstawiony w prosty sposób, poprzez dodanie ich poszczególnych wartości do listy, i opisanie ich ponad nią. Widok ma na celu przedstawienie ilości paliwa w litrach, jakie każdy zbiornik posiada w środku z określonej jego pojemności. Kliknięcie w dowolny zbiornik z listy, skutkuje pojawieniem się unikalnego dla wybranego rezerwuaru widoku (Rysunek 3.15) z dostawami. Użytkownik ma dzięki temu wgląd we wszystkie dostawy, ich daty oraz ilość paliwa, które zostają dodane przez niego poprzez kliknięcie w przycisk „Dodaj nową dostawę”. Przenosi go to do widoku formularza, z tymi samymi funkcjonalnościami

co poprzednie, tym razem wyłącznie z możliwością wypełnienia ilości dostarczonego paliwa i daty.

Rysunek 3.14 Widok Zbiorników w gospodarstwie

Zbiorniki w gospodarstwie		
Nr.	Ilość paliwa(L)	Pojemność zbiornika(L)
1	25200 /	30000
2	11100 /	40000
3	11000 /	20000

Rysunek 3.15 Widok dostaw paliwa po kliknięciu w dowolny zbiornik

Dostawy paliw	
DODAJ NOWĄ DOSTAWĘ	
Data	Ilość paliwa(L)
2023-01-16	200
2023-01-15	800
2023-01-15	600
2023-01-14	2000
2023-01-14	2000
2023-01-01	100
2023-01-01	100
2022-08-25	10000

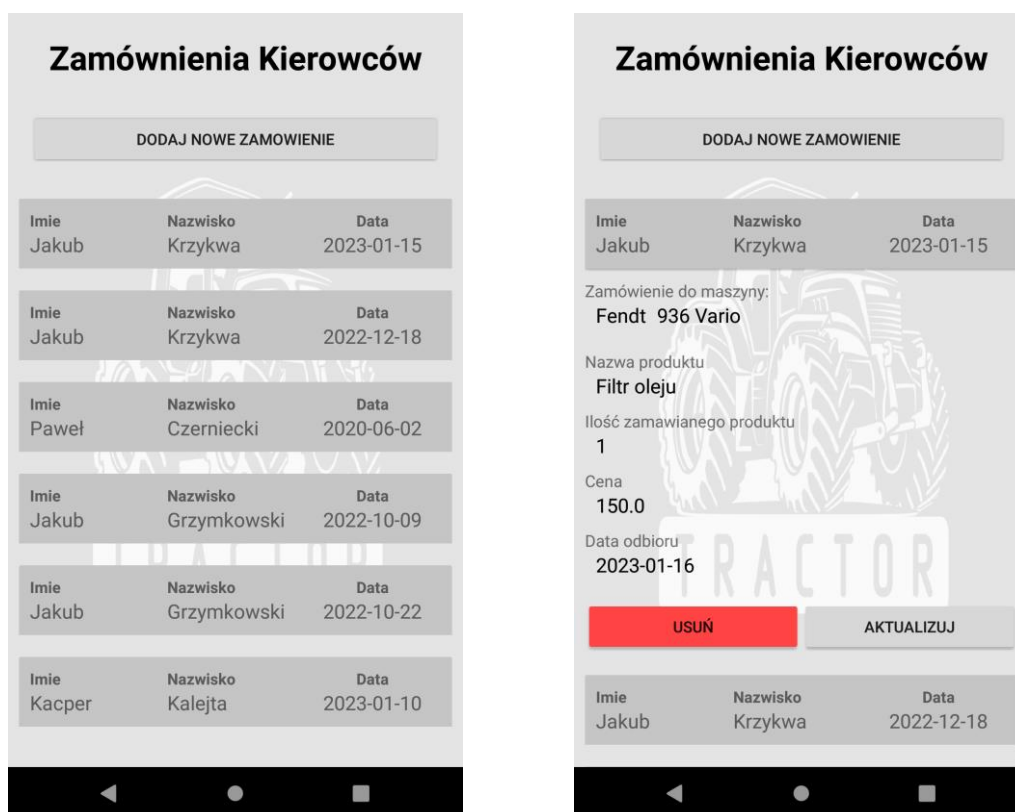
Źródło: opracowanie własne.

3.8 Widok zamówień kierowców

Widok zamówień kierowców (Rysunek 3.16) został przedstawiony podobnie do innych panelów z rozwijanymi listami. Jego celem jest pokazanie wszystkich danych dotyczących zamówienia, dodawanie nowych zamówień, ale także możliwość aktualizowania i usuwania tych powstałych. Formularz dodania nowego zamówienia wyglądają podobnie do (Rysunek 3.6), adekwatnie posiadając inne dane jakie powinny zostać wypełnione. Usuwanie klikniętej pozycji, usuwa ją i odświeża stronę. Natomiast przycisk „Aktualizuj”, przenosi użytkownika do widoku (Na obrazku **(Błąd! Nieprawidłowy odsyłacz do zakładki: wskazuje na nią sama.)**) można zauważyć, że pola, zostały automatycznie wypełnione informacjami o wybranym wcześniej zamówieniu. Pomoże to użytkownikowi w odpowiednim wypełnieniu formularza. W przypadku świeżo dodanego rekordu pole „Data odbioru” będzie puste.

Rysunek 3.17).

Rysunek 3.16 Widok zamówień kierowców



Źródło: opracowanie własne.

Na obrazku (**Błąd! Nieprawidłowy odsyłacz do zakładki: wskazuje na nią samą.**) można zauważyć, że pola, zostały automatycznie wypełnione informacjami o wybranym wcześniej zamówieniu. Pomoże to użytkownikowi w odpowiednim wypełnieniu formularza. W przypadku świeżo dodanego rekordu pole „Data odbioru” będzie puste.

Rysunek 3.17 Aktualizacja zamówienia

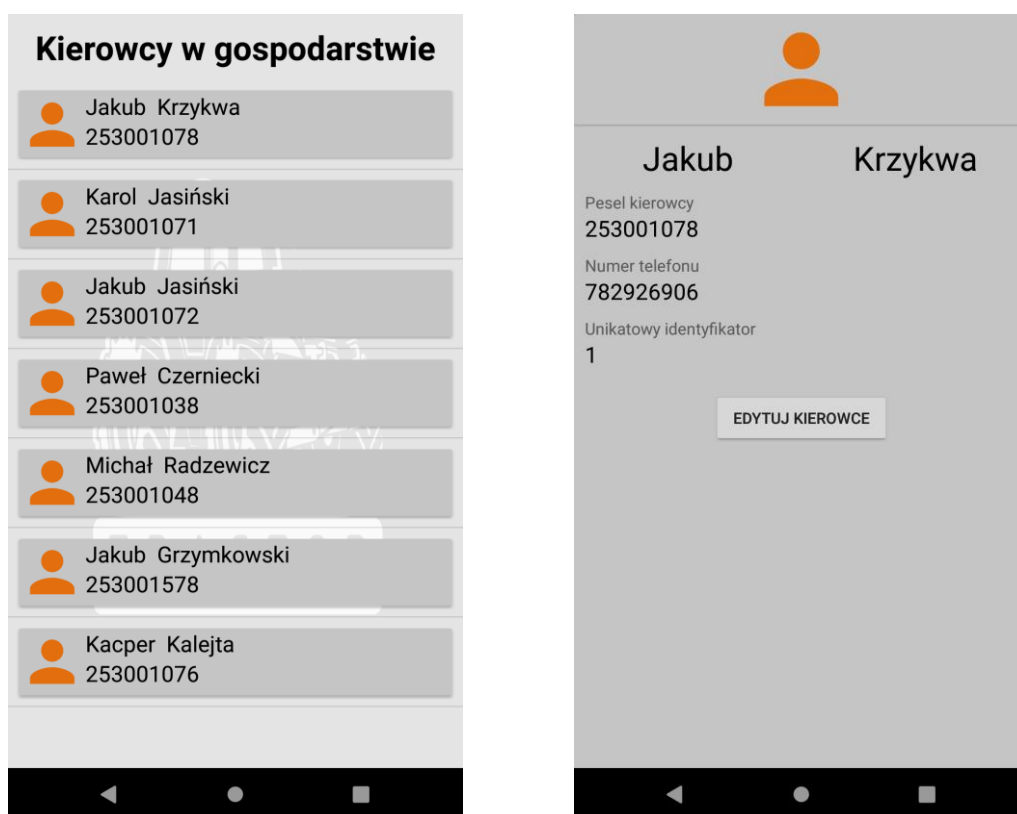
Nazwa produktu	Filtr oleju
Ilość zamawianego produktu	1
Data zamówienia	2023-01-15
Data odbioru	2023-01-16
Cena	150.0
Identyfikator Kierowcy	1
Identyfikator Maszyny	5
<button>ZATWIERDŹ</button>	

Źródło: opracowanie własne.

3.9 Widok kierowców w gospodarstwie

Rysunek po lewej przedstawia listę kierowców w której każdy z nich ma wypisane imię, nazwisko i pesel. Natomiast prawa ilustracja przedstawia widok, który pojawia się po kliknięciu w dowolną osobę, gdzie wypisane są wszystkie najistotniejsze informacje o pracowniku, wraz z przyciskiem umożliwiającym jego edycję. Edycja kierowcy działa podobnie do aktualizacji zamówień, gdzie pola są już wypełnione a użytkownik musi zmienić wyłączenie wybrane przez niego pola i zatwierdzić zmiany. Tym razem użytkownik, może zmienić jedynie imię, nazwisko lub numer telefonu. Zatwierdzenie zmian w tym przypadku, przenosi odbiorcę aplikacji z powrotem do szczegółów wybranego pracownika.

Rysunek 3.18 Widok kierowców i ich szczegółów



Źródło: opracowanie własne.

3.10 Ocena przygotowanych widoków interfejsu

Zaprezentowane projekty interfejsów aplikacji zostały stworzone w prosty i intuicyjny sposób, aby osoba nią operująca nie miała wątpliwości gdzie znajduje się w danym momencie, bez problemu przemieszczała się po niej, a także dodawała i aktualizowała potrzebne, a usuwała niepotrzebne mu informacje. Wszystkie modele są przejrzyste, a wielkość i kolorystyka czcionek odpowiednio dopasowane do całego wystroju aplikacji.

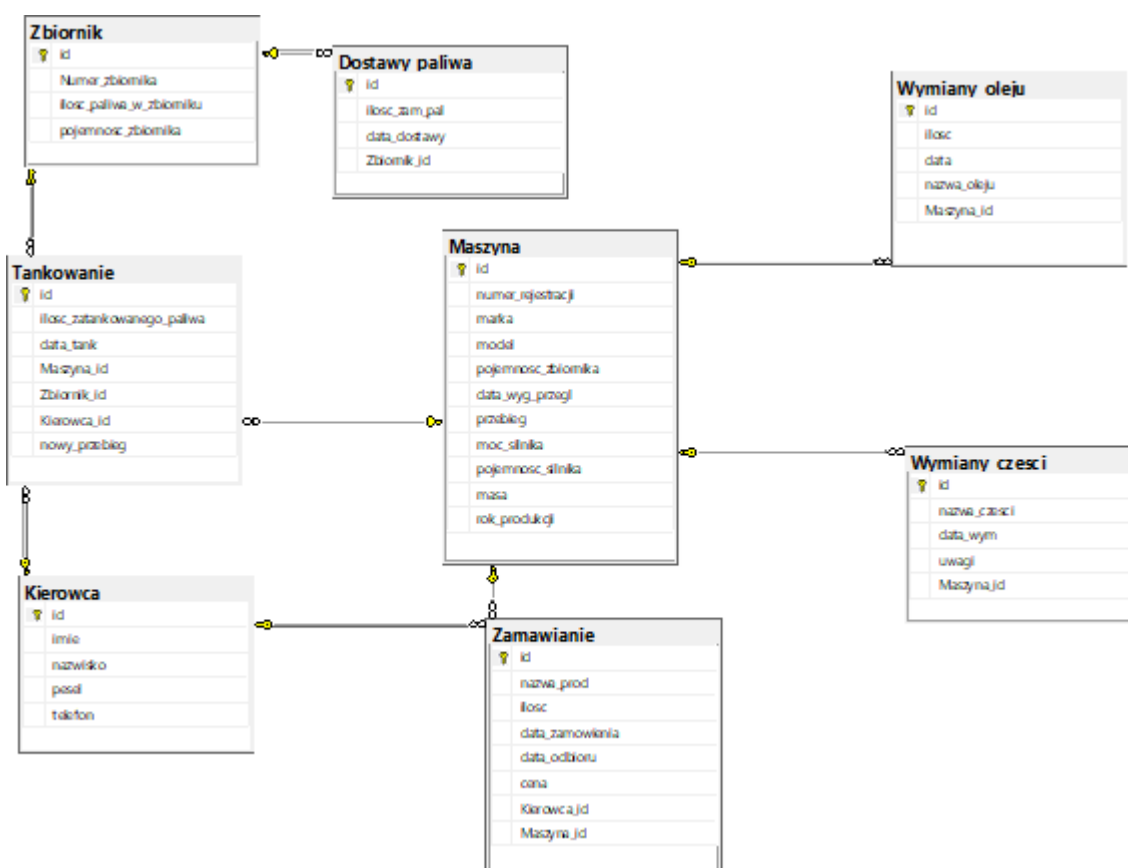
4 Projektowanie back-endu aplikacji do zarządzania parkiem maszynowym

Celem niniejszego rozdziału jest pokazanie wersji back-endowej aplikacji. Zostanie w nim przedstawiony proces implementacji bazy danych oraz funkcjonalności, a ich działanie będą poparte przykładami zawartymi w aplikacji.

4.1 Implementacja bazy danych

Jak zostało przedstawione na rysunku (Rysunek 4.1) baza danych składa się z 8 połączonych tabel. W pierwszej kolejności należy stworzyć ową bazę danych, a następnie dodać tabele główne, czyli te które nie mają żadnych kluczy obcych. W tym przypadku były nimi maszyny, kierowcy i zbiorniki.

Rysunek 4.1 Model bazy danych



Źródło: opracowanie własne.

Każda tabela zawiera nazwy kolumny i typy ich wartości. Przykładowy kod identyfikujący dodanie nowej tabeli głównej wygląda w następujący sposób (Rysunek 4.2), dodanie do niej nowego rekordu przedstawia rysunek (Rysunek 4.3), a widok po jej wywołaniu (Rysunek 4.4).

Rysunek 4.2 Tworzenie tabeli głównej w bazie danych

```
CREATE TABLE Maszyna
(
    id INTEGER NOT NULL ,
    numer_resjestracji CHAR (8) NOT NULL ,
    marka VARCHAR (50) NOT NULL ,
    model VARCHAR (50) NOT NULL ,
    pojemnosc_zbiornika INTEGER NOT NULL ,
    data_wyg_przegl DATE NOT NULL ,
    przebieg INTEGER NOT NULL ,
    moc_silnika VARCHAR (20) ,
    pojemnosc_silnika VARCHAR (30) ,
    masa INTEGER ,
    rok_produkcji INTEGER
    Primary key(id)
)
ALTER TABLE Maszyna ADD CONSTRAINT Maszyna__UN UNIQUE NONCLUSTERED (numer_resjestracji)
```

Źródło: opracowanie własne.

Rysunek 4.3 Dodanie nowego rekordu do tabeli głównej

```
INSERT INTO Maszyna(id,
    numer_rejestracji,
    marka,
    model,
    pojemnosc_zbiornika,
    data_wyg_przegl,
    przebieg,
    moc_silnika,
    pojemnosc_silnika,
    masa,
    rok_produkcji)
VALUES (1, 'NOE12345', 'Fendt', '936 Vario', 600,
    '2023-01-09', 924, '265 / 360', '7750', 10900, 2011)
```

Źródło: opracowanie własne.

Rysunek 4.4 Widok wywołanej tabeli

	id	numer_rejestracji	marka	model	pojemnosc_zbiornika	data_wyg_przebieg	przebieg	moc_silnika	pojemnosc_silnika	masa	rok_produkcji
1	1	NOE12345	Fendt	936 Vario	600	2023-01-09	1100	265 / 360	7750	10900	2011
2	2	NOE123B4	Fendt	939 Vario	600	2021-09-23	327	287 / 390	7750	10900	2013
3	3	NOEB7SE5	Fendt	927 Vario	600	2023-12-21	1167	199 / 270	7750	10830	2010
4	4	NOEB5678	New Holland	CS 660	580	2022-11-30	400	207 / 281	7474	12000	2006
5	5	NOEDM674	Claas	LEXION 580	980	2021-01-09	543	362 / 492	15928	16500	2005
6	6	NOEMN970	URSUS-KOHLER	KDI2504 EU IIIB	115	2023-05-20	342	54,5 / 74	2500	3750	2017
7	7	NOE9DTMY	New holland	Boomer 55	47	2023-06-09	130	35/57	1879	1720	2022
8	8	NOEE098X	John Deere	7930	392	2020-01-09	872	220 / 310	6788	7900	2008

Źródło: opracowanie własne.

Do typu kolumny z utworzonej tabeli zostały dodane odpowiednie wartości. W przypadku aktualizowania pola typu VARCHAR, CHAR i DATE należy użyć, okraszzonego w pojedyncze apostrofy na początku i końcu, łańcucha znaków, natomiast do pola typu INTEGER zwykłego ciągu cyfr. Pojawienie się ograniczenia NOT NULL, przy typie kolumny, identyfikuje czy pole wymaga wypełnienia w przypadku aktualizacji lub dodania nowego rekordu do tabeli. Ostatnią funkcją wprowadzoną do tabel było zaktualizowanie niektórych ich kolumn do unikatowych, zapewniając aby, tak jak przykładowo w przypadku tabeli „Maszyna”, dodając nowy rekord, numer rejestracji nigdy się nie powtórzył, gdyż każdy pojazd powinien mieć swoją własną unikatową rejestrację.

Następnym krokiem było utworzenie dodatkowych tabel funkcjonalnych, które zostały połączone z tabelami głównymi poprzez dodanie do nich kluczy obcych, będącymi odpowiednio unikatowymi i głównymi kluczami w tabelach głównych. Ponieważ będą one stale aktualizowane, poprzez dodawanie do nich nowych rekordów, przy tworzeniu owych tabel, których kod mógł wyglądać w następujący sposób (Rysunek 4.6), numery główne zostały wyposażone w dodatkową właściwość „IDENTITY (1,1)” znaną jako auto inkrementacja. Pierwsza liczba w nawiasie określa numer od którego auto inkrementacja będzie rozpoczynać wyliczanie, natomiast druga jest wartością przyrostową, która zostanie dodana do poprzedniego załadowanego wiersza. Dzięki niej uzupełnianie tabeli o nowy rekord nie wymaga od użytkownika sprecyzowania klucza głównego co również prezentuje ilustracja (Rysunek 4.5).

Rysunek 4.5 Utworzenie tabeli funkcjonalnej oraz dodanie do niej rekordu

```
CREATE TABLE Tankowanie
(
    id INTEGER NOT NULL IDENTITY(1,1),
    ilosc_zatankowanego_paliwa INTEGER NOT NULL ,
    data_tank DATE NOT NULL ,
    Maszyna_id INTEGER NOT NULL ,
    Zbiornik_id INTEGER NOT NULL ,
    Kierowca_id INTEGER NOT NULL ,
    nowy_przebieg INTEGER NOT NULL,
    Primary key(id)
)
ALTER TABLE Tankowanie ADD CONSTRAINT Tankowanie_Kierowca_FK FOREIGN KEY (Kierowca_id) REFERENCES Kierowca (id)
ALTER TABLE Tankowanie ADD CONSTRAINT Tankowanie_Maszyna_FK FOREIGN KEY (Maszyna_id) REFERENCES Maszyna (id)
ALTER TABLE Tankowanie ADD CONSTRAINT Tankowanie_Zbiornik_FK FOREIGN KEY (Zbiornik_id) REFERENCES Zbiornik (id)
INSERT Tankowanie(ilosc_zatankowanego_paliwa, data_tank, Maszyna_id, Zbiornik_id, Kierowca_id )
VALUES (200, '2022-11-12', 1, 1, 1)
```

Źródło: opracowanie własne.

Przez wszystkie dotychczas wykonane kroki, osiągnięto model całej bazy danych (Rysunek 4.1). Celem stworzenia owej infrastruktury była możliwość funkcjonowania tabel między sobą i połączenia ich w sposób który pozwoli później użytkownikowi na odczyt potrzebnych mu danych. Koncept można zaprezentować na podstawie przykładu „Zbiorników” i „Dostaw paliw”, mianowicie w przypadku dodania nowej dostawy paliwa w gospodarstwie, następuje dolanie cieczy do odpowiedniego zbiornika. Użytkownik będzie musiał w tym celu wypełnić ilość dostarczonego paliwa, datę oraz numer zbiornika, do którego zostaje wlana ciecz aby dodać nowy rekord do listy i zaktualizować ilość paliwa w podanym zbiorniku. Dlatego potrzebna jest implementacja klucza obcego w tabeli dostaw, którego odpowiednikiem jest klucz główny w tabeli zbiorników. Pomimo połączenie tych dwóch tabel, dodanie nowej dostawy paliwa, nie zmieni ilości paliwa w zbiorniku. W tym celu został utworzony wyzwalacz.

Rysunek 4.6 Wyzwalacz aktualizujący zbiornik

```
CREATE TRIGGER Update_Zbiornik
ON Dostawy_paliwa
for INSERT
AS
BEGIN
    declare @ilosc_pal int, @id int, @ilosc_pal_zbior int
    Select @ilosc_pal = z.ilosc_paliwa_w_zbiorniku + i.ilosc_zam_pal, @id = i.Zbiornik_id, @ilosc_pal_zbior = z.pojemnosc_zbiornika
    from inserted as i, Zbiornik as z where i.Zbiornik_id = z.id
    if(@ilosc_pal_zbior<@ilosc_pal)
    begin
        Print 'NO'
        rollback
    end
    else
        Update Zbiornik set ilosc_paliwa_w_zbiorniku = @ilosc_pal where id = @id
end
```

Źródło: opracowanie własne.

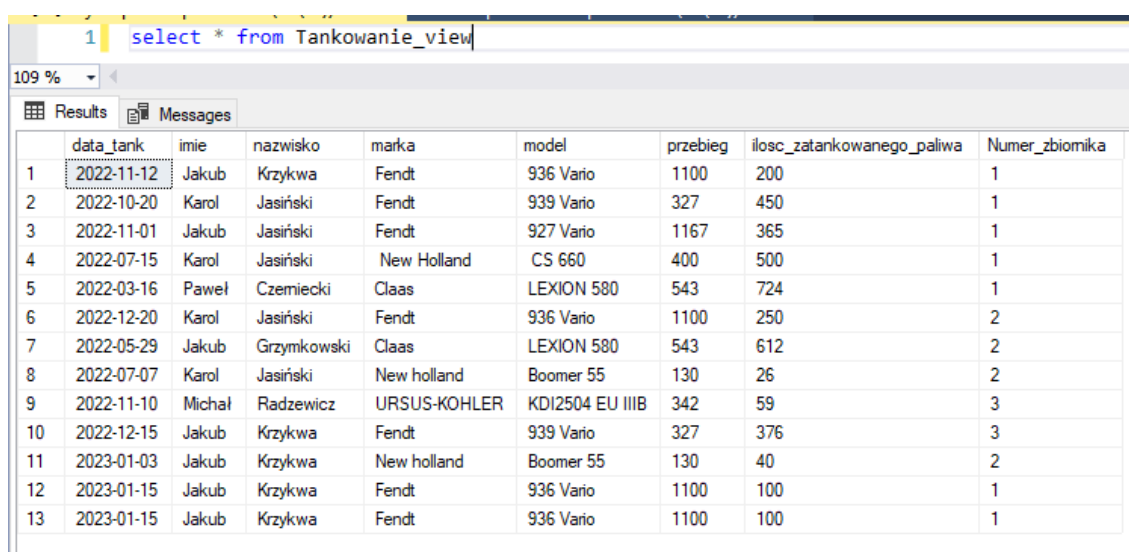
Powyższa ilustracja (Rysunek 4.6), przedstawia wyzwalacz o nazwie „Update_Zbiornik”, który samoistnie uruchamia się w momencie dodania nowego rekordu do tabeli „Dostawy paliwa”. Kod polega na zadeklarowaniu zmiennych i ich typów, następnie odpowiednim ich przypisaniu i poprzez wcześniejsze sprawdzenie instrukcją „if”, czy ilość nalewanego paliwa, po dodaniu aktualnego stanu cieczy, nie przekracza pojemności zbiornika, aktualizuje ilości cieczy w zbiornik o odpowiednim numerze identyfikacyjnym. Moduł „rollback” w instrukcji warunkowej, usuwa wszystkie modyfikacje danych dokonane od początku wyzwalacza i zwalnia zasoby utrzymywane przez niego. Bardzo podobny wyzwalacz został zaimplementowany również w przypadku dodawania nowego tankowanie maszyny, lecz tym razem ilość paliwa w zbiorniku zmniejsza się po wykonaniu akcji i nie może być mniejsza od zera z oczywistych przyczyn.

Rysunek 4.7 Utworzenie wizualizacji

```
Create View Tankowanie_view AS
SELECT T.data_tank,
K.imie,
K.nazwisko ,
M.marka, M.model,
M.przebieg,
T.ilosc_zatankowanego_paliwa,
Z.Numer_zbiornika
FROM Tankowanie AS T, Maszyna AS M, Zbiornik AS Z, Kierowca AS K
WHERE T.Kierowca_id = K.id AND T.Maszyna_id = M.id AND T.Zbiornik_id = Z.id
GO
```

Źródło: opracowanie własne.

Rysunek 4.8 Wygląd wizualizacji



	data_tank	imie	nazwisko	marka	model	przebieg	ilosc_zatankowanego_paliwa	Numer_zbiornika
1	2022-11-12	Jakub	Krzykwa	Fendt	936 Vario	1100	200	1
2	2022-10-20	Karol	Jasiński	Fendt	939 Vario	327	450	1
3	2022-11-01	Jakub	Jasiński	Fendt	927 Vario	1167	365	1
4	2022-07-15	Karol	Jasiński	New Holland	CS 660	400	500	1
5	2022-03-16	Paweł	Czemiecki	Claas	LEXION 580	543	724	1
6	2022-12-20	Karol	Jasiński	Fendt	936 Vario	1100	250	2
7	2022-05-29	Jakub	Grzymkowski	Claas	LEXION 580	543	612	2
8	2022-07-07	Karol	Jasiński	New holland	Boomer 55	130	26	2
9	2022-11-10	Michał	Radzewicz	URSUS-KOHLER	KD12504 EU IIIB	342	59	3
10	2022-12-15	Jakub	Krzykwa	Fendt	939 Vario	327	376	3
11	2023-01-03	Jakub	Krzykwa	New holland	Boomer 55	130	40	2
12	2023-01-15	Jakub	Krzykwa	Fendt	936 Vario	1100	100	1
13	2023-01-15	Jakub	Krzykwa	Fendt	936 Vario	1100	100	1

Źródło: opracowanie własne.

Do ułatwienia odczytu poszczególnych funkcjonalności bazy, utworzone zostały również ich wizualizacje (Rysunek 4.7), pozwalające poprzez połączenia tabel, pokazanie najważniejszych informacji potrzebnych dla użytkownika. Natomiast do szybszego dodania, bądź aktualizacji rekordu, procedur, skracających i optymalizujących wykonanie wymienionych operacji.

Przykładowy kod tworzący i wykonujący procedurę dodania nowego rekordu:

Rysunek 4.9 Utworzenie i wykonanie procedury dodania nowego rekordu

```

create proc Nowa_Dostawa
@ilosc_zam_pal int,
@data_dostawy date,
@Zbiornik_id int
as
begin
insert into Dostawy_paliwa values(@ilosc_zam_pal,@data_dostawy,@Zbiornik_id)
end
GO

exec Nowa_Dostawa 1000, '2023-01-02', 3

```

Źródło: opracowanie własne.

Przykładowy kod tworzący i wykonujący procedurę aktualizacji istniejącego rekordu:

Rysunek 4.10 Kod tworzący i wywołujący procedurę aktualizującą rekord

```
create proc Update_Zamawianie
@id int,
@nazwa_prod varchar(50),
@ilosc int,
@data_zamowienia date,
@data_odbioru date,
@cena real,
@kierowca_id int,
@maszyna_id int
as
begin
Update Zamawianie set nazwa_prod=@nazwa_prod, ilosc=@ilosc,
data_zamowienia=@data_zamowienia,
data_odbioru = @data_odbioru, cena = @cena,
Kierowca_id = @kierowca_id , Maszyna_id = @maszyna_id
where id = @id
end
GO

exec Update_Zamawianie 13, 'Zestaw rozrusznika', 1, '2023-01-10', '2023-01-15', 840, 7, 8
```

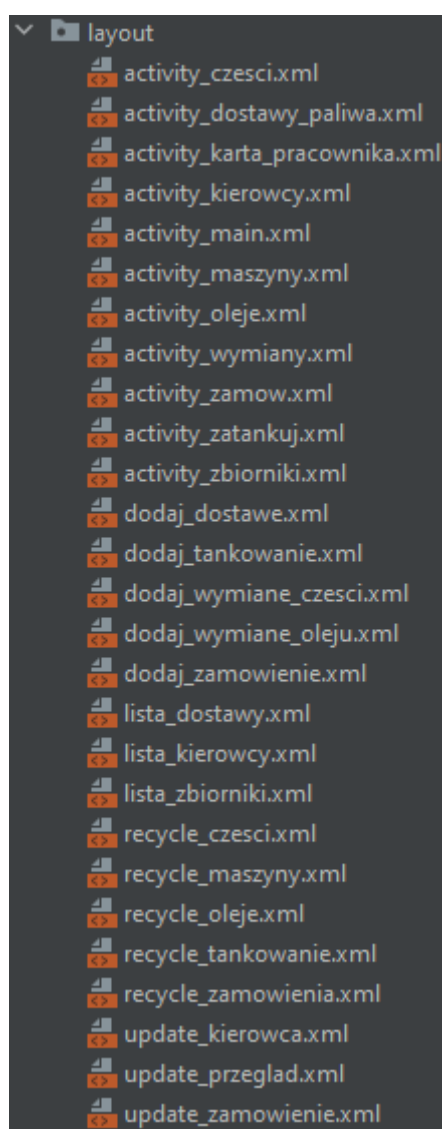
Źródło: opracowanie własne.

4.2 Implementacja aplikacji

4.2.1 Widoki, komponenty

Aktywności i widoki są jednymi z najważniejszych elementów każdej aplikacji mobilnej przeznaczonej na platformę Android. Celem tworzenia aktywności jest zwizualizowanie potrzebnych użytkownikowi informacji na widokach, które poprzez wszelkiego rodzaju przyciski, listy, typografię, ikony, ilustracje czy kolory będą odpowiadać za komunikację z odbiorcą. Jest to możliwe przy pomocy plików z rozszerzeniem xml. Widoki interfejsu użytkownika powstały przy pomocy RelativeLayout, LinearLayout, GridLayout, jak również w oparciu o ich elementy dostępne w AndroidSDK, które zaprojektowane w odpowiedni sposób dodają prostoty, intuicyjności i lepszej wydajności, zwiększając tym samym satysfakcję z korzystania z produktu. W oparciu o to zostały zdefiniowane wizualizacje list oraz ich komponentów przedstawionych na rysunku (Rysunek 4.11).

Rysunek 4.11 Widoki wykorzystane w aplikacji



Źródło: opracowanie własne.

4.2.2 Połączenie bazy danych z aplikacją

Aby połączyć aplikację utworzoną w Android Studio, należy wykorzystać sterownik z rozszerzeniem typu jar. Projekt został rozszerzony o plik jtids-1.3.1.jar umieszczony w folderze libs projektu. Liczba 1.3.1 odpowiada wersji pliku. Jest to najnowszy sterownik typu open source dla Microsoft SQL Servera, wspierający komunikację z bazą danych i implementujący wszystkie metody DatabaseMetaData oraz ResultSetMetaData. W następnej kolejności aby dodać plik jtids do aplikacji, plik build.gradle (Module: „nazwa aplikacji”.app) musi zostać rozszerzony o kod:

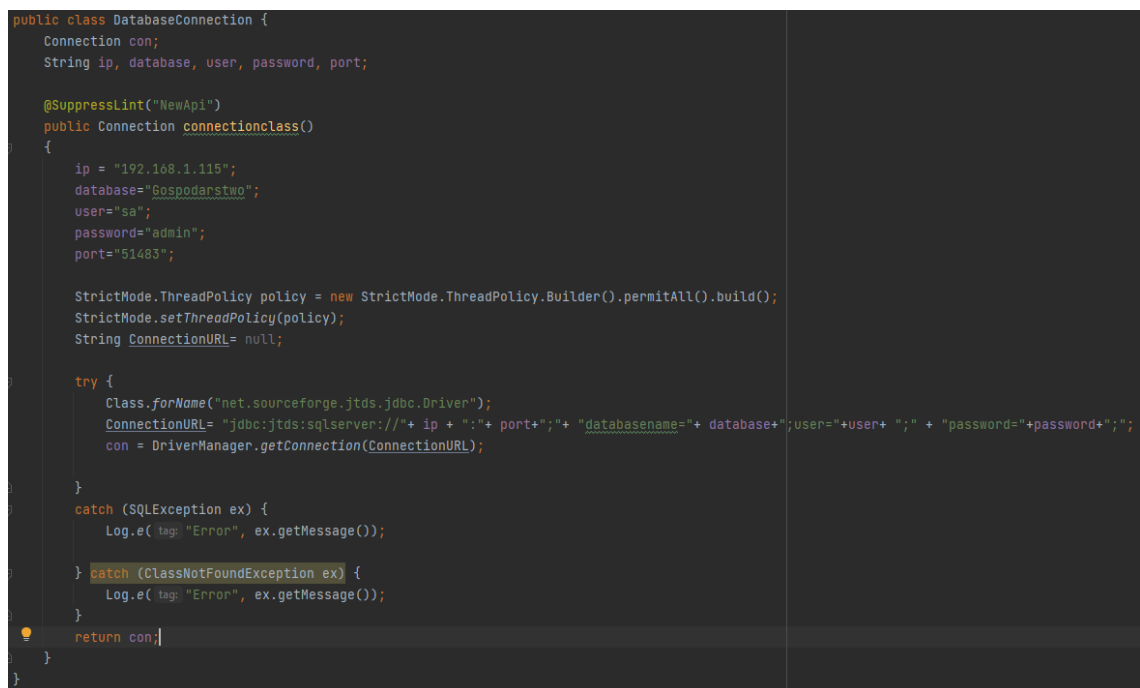
implementation files('lokalizacja pliku jar')

Natomiast w celu pobierania i aktualizacji rekordów z utworzonej wcześniej bazy potrzebne jest połączenie z internetem, dlatego plik AndroidManifest powinien zostać powiększony o następujące uprawnienia:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
```

Ostatnią czynnością jaką należało wykonać, było stworzenie klasy łączącej z bazą danych (Rysunek 4.12).

Rysunek 4.12 Kod klasy DatabaseConnection

The image shows a screenshot of a code editor with a dark theme. It displays the Java code for a class named DatabaseConnection. The code includes a constructor annotated with @SuppressWarnings("NewApi") that initializes several fields: ip, database, user, password, and port. It also sets a StrictMode.ThreadPolicy and attempts to establish a JDBC connection to a SQL Server. The connection URL is constructed using the class's fields. The code includes try-catch blocks for SQLException and ClassNotFoundException, logging errors with Log.e(). The constructor returns the Connection object 'con'.

```
public class DatabaseConnection {
    Connection con;
    String ip, database, user, password, port;

    @SuppressWarnings("NewApi")
    public Connection connectionclass()
    {
        ip = "192.168.1.115";
        database="Gospodarstwo";
        user="sa";
        password="admin";
        port="51483";

        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);
        String ConnectionURL= null;

        try {
            Class.forName("net.sourceforge.jtds.jdbc.Driver");
            ConnectionURL= "jdbc:jtds:sqlserver://" + ip + ":" + port + ";" + "databasename=" + database + ";user=" + user + ";" + "password=" + password + ";";
            con = DriverManager.getConnection(ConnectionURL);
        }
        catch (SQLException ex) {
            Log.e( tag: "Error", ex.getMessage());
        } catch (ClassNotFoundException ex) {
            Log.e( tag: "Error", ex.getMessage());
        }
        return con;
    }
}
```

Źródło: opracowanie własne.

Klasa DatabaseConnection składa się z:

- pola „con” i metody „connectionclass” typu Connection,
- pól „ip”, „database”, „user”, „password” i „port” typu String

Do wcześniej zadeklarowanych zmiennych typu String, metoda przypisuje informacje o połączeniu internetowym oraz bazie danych:

- ip, przechowuje adres IPv4 połączenia internetowego,
- database, nazwa bazy danych,
- user, nazwa administratora bazy danych,
- password, hasło administratora bazy danych,
- port, port używany przez MSSQL Server.

Zmienna „policy” typu StrictMode.ThreadPolicy zastosowano w celu ustawienia jakich problemów ma za zadanie szukać aktualny wątek. Użycie w tym przypadku publicznej metody „permitAll()”, zapobiega wszystkim wykrywanom. Następnie zadeklarowano zmienną „ConnectionString”, aby przechowywać adres połączenia z bazą danych.

Blok „try” rozpoczyna się od wywołania funkcji „Class.forName()”, która dzięki metodzie „forName()” pobiera instancję klasy określonej poprzez tekst znajdujący się w nawiasach. Tym razem jest nim Driver, sterownik, który jest potrzebny do połączenia z bazą danych.

W kolejnym kroku do zmiennej „connectURL” zostaje przypisany adres wykorzystujący wcześniej zadeklarowane zmienne, aby został ostatecznie użyty w funkcji „DriverManager.getConnection()”, która podejmie próbę załadowania wcześniej utworzonego sterownika w celu uzyskania dostępu do odczytu i pobierania danych z bazy. Ostatecznie blok „try” zwróci pole „con” przypisane do ostatniej wymienionej funkcji.

4.2.3 Przechodzenie pomiędzy głównymi widokami aplikacji

Panel nawigacji pomiędzy najważniejszymi funkcjonalnościami aplikacji został zaimplementowany w klasie „MainActivity”. Wygląd menu głównego został stworzony za pomocą GridView, tworzącej układ ze stałą liczbą kafelków w tak zwanej siatce. Następnie każde pole w siatce zostało wypełnione komponentem CardView, któremu został przypisany identyfikator w pliku XML, służący do znalezienia określonego widoku. Przykładem kodu umożliwiającego przechodzenie pomiędzy funkcjonalnościami może być:

Rysunek 4.13 Przykładowy kod umożliwiający przechodzenie pomiędzy menu a aktywnością

```
CardView Zatankuj = findViewById(R.id.Zatankuj);  
Zatankuj.setOnClickListener((View v) ->{  
    Intent intent = new Intent( packageContext: MainActivity.this, ZatankujActivity.class);  
    startActivity(intent);  
});
```

Źródło: opracowanie własne.

Rysunek 4.14 Kod przykładowego CardView w menu głównym

```
<androidx.cardview.widget.CardView  
    android:id="@+id/Zatankuj"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_columnWeight="1"  
    android:layout_rowWeight="1"  
    android:layout_row="1"  
    android:layout_column="0"  
    android:layout_gravity="fill"  
    app:cardCornerRadius="8dp"  
    app:cardUseCompatPadding="true">
```

Źródło: opracowanie własne.

Aby przejście do innej aktywności było możliwe, należało użyć metody „setOnClickListener()”, która zostaje wywołana w momencie gdy użytkownik wybierze jeden z elementów na panelu. W przypadku kodu z rysunku (Rysunek 4.13) CardView zostaje nadana zmienna, a następnie poprzez metodę „findViewById()” znalezienia odpowiedniego identyfikatora komponentu zadeklarowanej w pliku XML, jak pokazuje przykład (Rysunek 4.14). Następnie stworzona jest nowa zmienna typu „Intent”, umożliwiającą uruchomienie odpowiedniej aktywności, którą w przykładowym kodzie jest „ZatankujActivity”.

4.2.4 Działanie i implementacja listy RecyclerView

RecyclerView ułatwia kreatywne wyświetlanie większych zestawów danych. Deweloper może określić wygląd listy, dopasować elementy do własnych potrzeb, a następnie wypełnić pola potrzebnymi danymi. Działanie i implementacje biblioteki można przedstawić na podstawie aktywności zatankowań w gospodarstwie, które potrzebuje dużej ilości informacji.

Rysunek 4.15 Wygląd elementu przykładowej listy RecyclerView z zatankowaniami w gospodarstwie

The image shows a screenshot of a single item in a RecyclerView list. The item is a light blue rectangular card with rounded corners. At the top, there is a table with four columns: 'Imie', 'Nazwisko', 'Ilosc paliwa(L)', and 'Data'. Below the table, there are several text input fields: 'Marka', 'Model', 'Przebieg maszyny w trakcie tankowaniem', 'Przebieg maszyny', 'Numer użytego zbiornika', and 'Numer zbiornika'. The 'Przebieg maszyny' field has a small white square slider control next to it. The entire card is set against a dark blue background.

Źródło: opracowanie własne.

Wszystkie podpisane elementy listy to pola typu TextView. Najważniejsze informacje dotyczące tankowania zostały umieszczone na samej górze, a elementy rozwijane poniżej aby uniknąć przesyty informacji po otwarciu aktywności.

Następnym krokiem do poprawnego działania listy było stworzenie klasy „GetTankowanie” przechowującej i zwracającej odpowiednie informacje o każdym komponencie elementu listy.

Rysunek 4.16 Budowa przykładowej klasy Get

```
public class GetTankowanie {
    String data_tank, imie, nazwisko, ilosc_zatankowanego_paliwa, marka, model, przebieg, numer_zbiornika;
    boolean isVisible;

    public GetTankowanie(String imie, String nazwisko, String ilosc_zatankowanego_paliwa,
        String data_tank, String marka, String model, String przebieg, String numer_zbiornika, boolean isVisible) {
        this.data_tank = data_tank;
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.ilosc_zatankowanego_paliwa = ilosc_zatankowanego_paliwa;
        this.marka = marka;
        this.model = model;
        this.przebieg = przebieg;
        this.numer_zbiornika = numer_zbiornika;
        this.isVisible = false;
    }

    public String getData_tank() { return data_tank; }
    public String getImie() { return imie; }
    public String getNazwisko() { return nazwisko; }
    public String getIlosc_zatankowanego_paliwa() {...}
    public String getMarka() { return marka; }
    public String getModel() { return model; }
    public String getPrzebieg() { return przebieg; }
    public String getNumer_zbiornika() { return numer_zbiornika; }
    public boolean isVisible() { return isVisible; }
    public void setVisible(boolean visible) { isVisible = visible; }
}
```

Źródło: opracowanie własne.

W klasie (Rysunek 4.16) zostały zaimplementowane pola typu String, przechowujące informacje tekstowe oraz jedno pole typu logicznego isVisible, odpowiadające rozwijaniu się listy. Następnie został utworzony konstruktor i zbiór getterów, w których pierwszy z nich ustawia konkretne wartości pól, a drugi zwraca owe informacje i pozwala je pobrać w innych klasach.

Następnie, w celu połączenia napływających danych, przechowywanych w ArrayList, w tym przypadku typu GetTankowanie, z komponentem RecyclerView, w klasie z aktywnością, została zaimplementowana klasa Adapter, która pomoże w osiągnięciu tego celu poprzez wbudowane w nią niezbędne metody onCreateViewHolder, onBindViewHolder, getItemCount. Dodatkowo Adapter został wyposażony w pole typu Context zwracający informacje na jakim widoku obecnie pracuje aplikacja.

Rysunek 4.17 Przykładowa klasa ViewHolder

```
public class ViewHolder extends RecyclerView.ViewHolder
{
    TextView imie_tank, nazw_tank, pal_tank, data_tank, marka_tank, model_tank, przebieg_tank, nr_zbior_tank;
    CardView card_view3;
    LinearLayout lin_visible;
    View layout;

    public ViewHolder(View v)
    {
        super(v);
        layout = v;
        imie_tank = v.findViewById(R.id.imie_tank);
        nazw_tank = v.findViewById(R.id.nazw_tank);
        pal_tank = v.findViewById(R.id.pal_tank);
        data_tank = v.findViewById(R.id.data_tank);
        marka_tank = v.findViewById(R.id.marka_tank);
        model_tank = v.findViewById(R.id.model_tank);
        przebieg_tank = v.findViewById(R.id.przebieg_tank);
        nr_zbior_tank = v.findViewById(R.id.nr_zbior_tank);
        card_view3 = v.findViewById(R.id.card_view3);
        lin_visible = v.findViewById(R.id.lin_visible);

        card_view3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v){
                GetTankowanie getTankowanie = itemsArrayList.get(getAbsoluteAdapterPosition());
                getTankowanie.setVisible(!getTankowanie.isVisible());
                notifyItemChanged(getAbsoluteAdapterPosition());
            }
        });
    }
}
```

Źródło: opracowanie własne.

Rysunek 4.18 Funkcje onCreateViewHolder, onBindViewHolder, getItemCount

```
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)
{
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View v = inflater.inflate(R.layout.recycle_tankowanie, parent, attachToRoot false);
    ViewHolder vh = new ViewHolder(v);
    return vh;
}

@Override
public void onBindViewHolder(ViewHolder holder, final int position){
    final GetTankowanie getTankowanie = values.get(position);
    holder.imie_tank.setText(getTankowanie.getImie());
    holder.nazw_tank.setText(getTankowanie.getNazwisko());
    holder.pal_tank.setText(getTankowanie.getIlosc_zatankowanego_paliwa());
    holder.data_tank.setText(getTankowanie.getData_tank());
    holder.marka_tank.setText(getTankowanie.getMarka());
    holder.model_tank.setText(getTankowanie.getModel());
    holder.przebieg_tank.setText(getTankowanie.getPrzebieg());
    holder.nr_zbior_tank.setText(getTankowanie.getNumer_zbiornika());

    boolean isVisible = itemsArrayList.get(position).isVisible();
    holder.lin_visible.setVisibility(isVisible ? View.VISIBLE : View.GONE);
}

@Override
public int getItemCount() { return values.size(); }
```

Źródło: opracowanie własne.

Najpierw została stworzona publiczna klasa ViewHolder (Rysunek 4.17), która ma za zadanie znalezienie widoku wszystkich elementów listy, jak również daje możliwość rozwijania i zwijania elementów poprzez naciśnięcie w CardView.

Następnie do Adaptera dodawano kolejno niezbędne metody (Rysunek 4.18):

- onCreateViewHolder, mający za cel zwracanie nowego ViewHolder'a, czyli utworzonego wcześniej widoku pojedynczego elementu,
- onBindViewHolder, używa elementy nadane w klasie ViewHolder i pobiera odpowiednie wartości z klasy GetTankowanie, a ostatecznie odpowiada za chowanie zadeklarowanego komponentu listy,
- getItemCount – zwraca liczbę elementów aktualnie dostępnych w adapterze.

Rysunek 4.19 Przykładowa metoda „doInBackground” w aplikacji

```
protected String doInBackground(String... strings)
{
    try
    {
        Connection con = databaseConnection.connectionClass();
        if (con == null)
        {
            success = false;
        }
    }
    else
    {
        String query = "SELECT * from Tankowanie_view ORDER BY data_tank DESC";
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        if (rs != null)
        {
            while (rs.next())
            {
                try {
                    itemsArrayList.add(new GetTankowanie(rs.getString(® "imie"), rs.getString(® "nazwisko"),
                    rs.getString(® "ilosc_zatankowanego_paliwa"),
                    rs.getString(® "data_tank"),
                    rs.getString(® "marka"),
                    rs.getString(® "model"),
                    rs.getString(® "przebieg"),
                    rs.getString(® "numer_zbiornika"),
                    isVisible: false));
                } catch (Exception ex){
                    ex.printStackTrace();
                }
            }
            msg = "Found";
            success = true;
        }else{
            msg = "No Data Found!";
            success = false;
        }
    }
}
} catch (Exception e)
{
    e.printStackTrace();
    Writer writer = new StringWriter();
    e.printStackTrace(new PrintWriter(writer));
    msg = writer.toString();
    success = false;
}
return msg;
```

Źródło: opracowanie własne.

Rysunek 4.20 Przykładowa metoda onPostExecute() w aplikacji

```
@Override
protected void onPostExecute(String msg)
{
    if(success)
    {
        myAdapter = new MyAdapter(itemsArrayList, context: ZatankujActivity.this);
        recyclerView.setAdapter(myAdapter);
    }
}
```

Źródło: opracowanie własne.

Ostatecznie do klasy Tankowanie, została dodana prywatna klasa „AsyncTask”, do której następnie wprowadzono metodę asynchroniczną „doInBackground()”, która pobiera rekordy z bazy danych. Klasa pobiera owe rekordy w tle na oddzielnym wątku, następnie aktualizując wątek główny. Metoda początkowo łączy się z bazą danych za pomocą stworzonej wcześniej klasy DatabaseConnection. Następnie, poprzez zapytanie zaciąga zawartość z odpowiedniej tabeli.

W następnej kolejności, aby wypełnić tabelę informacjami z bazy, w bloku „try” każdy rekord zostaje dodany jako pojedynczy element do wcześniej utworzonej w Adapterze ArrayListy, co po udanym wczytaniu rekordów skutkuje w zmianie pola „success” na „true”. Owa zmiana zostaje następnie wykorzystana w metodzie „onPostExecute()”, z ilustracji (Rysunek 4.20), w której pole myAdapter, typu klasy MyAdapter, zostaje wypełnione ArrayListą posiadającą już rekordy z bazy danych, oraz odpowiednim widokiem któremu ma przekazać dane. Ostatecznie zostaje przypisana, do pola „recyclerView”, jako adapter.

4.2.5 Działanie listy ListView

Rysunek 4.21 Klasa Adapter listy typu ListView

```
public class Adapter extends BaseAdapter
{
    public class ViewHolder{
        TextView numer_zbiornika, ilosc_paliwa, poj_zbior_paliwa;
    }

    public Context context;
    public ArrayList<GetZbiornik> arrayList;

    private Adapter(ArrayList<GetZbiornik> Gospodarstwo, Context context){
        this.arrayList = Gospodarstwo;
        this.context = context;
    }

    @Override
    public int getCount() { return arrayList.size(); }
    @Override
    public Object getItem(int position) { return position; }
    @Override
    public long getItemId(int position) { return position; }
    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {

        View rowView = convertView;
        SyncData.Adapter.ViewHolder viewHolder = null;
        if(rowView == null)
        {
            LayoutInflater inflater = getLayoutInflater();
            rowView = inflater.inflate(R.layout.lista_zbiorniki, parent, attachToRoot: false);
            viewHolder = new SyncData.Adapter.ViewHolder();
            viewHolder.numer_zbiornika = rowView.findViewById(R.id.numer_zbiornika);
            viewHolder.ilosc_paliwa = rowView.findViewById(R.id.ilosc_paliwa);
            viewHolder.poj_zbior_paliwa = rowView.findViewById(R.id.poj_zbior_paliwa);
            rowView.setTag(viewHolder);
        } else {
            viewHolder = (SyncData.Adapter.ViewHolder) convertView.getTag();
        }

        viewHolder.numer_zbiornika.setText(arrayList.get(position).getNumer_zbiornika());
        viewHolder.ilosc_paliwa.setText(arrayList.get(position).getIlosc_paliwa_w_zbiorniku());
        viewHolder.poj_zbior_paliwa.setText(arrayList.get(position).getPojemnosc_zbiornika());

        return rowView;
    }
}
```

Źródło: opracowanie własne.

Ilustracja przedstawia kod adapteru listy typu „ListView”. Działanie jest bardzo podobne do tego z RecyclerView, czyli pełni funkcje wyświetlania danych pobranych do ArrayListy w widoku posiadającym odpowiednią listę ListView. Klasa posiada niezbędne do jej działania metody:

- getCount(), zwraca ilość elementów listy,
- getItem(), zwraca obiekt danego elementu, umożliwia uzyskiwanie danych w adapterze,
- getItemId(), zwraca numer (pozycje) wybranego elementu listy,
- getView(), generuje widoki elementów listy.

W celu połączenia się z danymi z bazy danych została, tak jak w przypadku listy RecyclerView, klasa SyncData z metodą asynchroniczną „doInBackground”, pozwalająca na załadowanie informacji z tabel w tle. Możliwość uzyskania większej ilości informacji o elemencie z listy, została zapewniona przez naciśnięcie na jednego z nich. W tym celu użyto metody „onItemClickListener()”, przedstawionej na rysunku (). Jego zadaniem jest przeniesienie użytkownika do odpowiedniej aktywności, która umożliwi wyświetlenie danych dla odpowiedniego elementu, poprzez przekazanie następnej klasie numeru identyfikującego wybranego elementu.

Rysunek 4.22 Metoda setOnItemClickListener()

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> adapterView, View view, int position, long id) {  
        Intent intent = new Intent( packageContext: ZbiornikiActivity.this, DostawyPaliwaActivity.class);  
        String id_zbiornik = itemsArrayList.get(position).getNumer_zbiornika();  
        intent.putExtra( name: "id_zbiornik", id_zbiornik);  
        startActivity(intent);  
        finish();  
    }  
});
```

Źródło: opracowanie własne.

4.2.6 Dodawanie, usuwanie i aktualizacja danych w aplikacji

W widokach aplikacji, można zauważyć różne funkcjonalności. Dla przykładu, panel tankowania maszyn, oprócz wypisywania wszystkich rekordów znajdujących się w bazie, umożliwia użytkownikowi dodawanie kolejnych zatankowań maszyn, poprzez przycisk „Dodaj nowe tankowanie”, podobne do tego z ilustracji (Rysunek 3.5). Przycisk zwyczajnie przenosi użytkownika do formularza, w którym ten musi wypełnić pola odpowiednimi wartościami. Następnie po naciśnięciu w przycisk zatwierdzający zmiany zostaje przeniesiony z powrotem do listy zatankowań.

Rysunek 4.23 Przykładowy kod dodania nowego rekordu do bazy

```
add.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String paliwo = paliwo_tank.getText().toString();
        String data = data_tank.getText().toString();
        String id_m = id_maszynny_tank.getText().toString();
        String id_z = id_zbiornik_tank.getText().toString();
        String id_k = id_kierowcy_tank.getText().toString();
        String n_przebieg = nowy_przebieg_tank.getText().toString();

        DatabaseConnection databaseConnection = new DatabaseConnection();
        Connection conn = databaseConnection.getConnection();
        try {
            if (conn != null) {
                if (paliwo.equals("") || data.equals("") || id_m.equals("") || id_z.equals("") || id_k.equals("") || n_przebieg.equals("")){
                    Toast.makeText(getApplicationContext(), "Pola nie mogą być puste!", Toast.LENGTH_LONG).show();
                } else {
                    Intent intent = new Intent( packageContext.DodajTankowanie.this, ZatankujActivity.class);
                    startActivity(intent);
                    finish();
                    String query = "exec Nowe_Tankowanie " + paliwo + "," + data + "," + id_m + "," + id_z + "," + id_k + "," + n_przebieg;
                    Statement st = conn.createStatement();
                    ResultSet rs = st.executeQuery(query);
                }
            }
        } catch (Exception ex){
            ex.printStackTrace();
        }
    }
});
```

Źródło: opracowanie własne.

Rysunek (Rysunek 4.23) przedstawia kod inicjalizujący dodanie nowego tankowania w gospodarstwie. Cała sekwencja zasilenia bazy o nowy rekord zaczyna się od przypisania przycisku i dodanie do niego metody `setOnClickListener`. Następnie zadeklarowane zostały pola typu `String`, przechowujące wartości podane przez użytkownika w formularzu. Każde pola zostało przypisane do odpowiednio znalezionych, przed wywołaniem metody, edytowalnego pola tekstowego. Kolejno następuje połączenie się z bazą. Zaimplementowany blok „try”, na początku sprawdza czy połączenie jest odpowiednie, a następnie poprzez klauzulę „if”, upewnia się, czy wszystkie pola zostały wypełnione i nie pozwoli użytkownikowi zatwierdzić zmian, dopóki to nie nastąpi. Odbywa się to poprzez metodę „equals()”, która została przypisana każdemu zadeklarowanemu wcześniej polu i w przypadku gdy jakkolwiek pozycja w formularza jest pusta wypisuje informuje odbiorcę o błędzie. Jeśli jednak wszystko przebiegło poprawnie, zostaje wywołana procedura „Nowe_Tankowanie”, pobierająca wszystkie wartości podane przez użytkownika i inicjuje dodanie rekordu. W między czasie aplikacja zmienia aktywność na wcześniej widoczną listę zatankowań.

W przypadku tankowania zostały również dodane 2 wyzwalacze, w którym pierwszy z nich odpowiada odejmowaniu zatankowanego przez maszynę paliwa, a drugi za aktualizację maszyny i przypisanie jej nowego podanego przez użytkownika przebiegu. Zostały one utworzone na wzór wcześniej omawianych wyzwalaczy.

Widok zamówień posiada również dodatkowe funkcjonalności jakimi są, usuwanie rekordów z listy oraz ich aktualizacja (Rysunek 3.16).

Usuwanie osiągnięto poprzez następujący kod:

Rysunek 4.24 Przykładowe usuwanie rekordu z listy

```
usun.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        GetZamowienie getZamowienie = itemsArrayList.get(getAbsoluteAdapterPosition());
        DatabaseConnection databaseConnection = new DatabaseConnection();
        Connection conn = databaseConnection.connectionclass();
        try {
            if (conn != null) {
                String query = "delete from Zamawianie where id= " + getZamowienie.getId();
                Statement st = conn.createStatement();
                ResultSet rs = st.executeQuery(query);
            }
            else
            {
                msg = "No connection";
            }
        } catch (Exception ex){
            ex.printStackTrace();
        }
        Intent intent = new Intent( packageContext ZamowActivity.this, ZamowActivity.class);
        startActivity(intent);
        finish();
    }
});
```

Źródło: opracowanie własne.

Przedstawiony kod odnosi się do przycisku „usuń” pojawiający się przy każdym rozwijanym elemencie listy. Usuwanie jest osiągnięte poprzez metodę „setOnClickListener()”. Metoda poza znanym już schematem łączenia się z bazą, wywołaniem odpowiedniego zapytania do bazy danych i przeniesieniem użytkownika do właściwej aktywności, rozpoczyna działanie przypisując zmiennej typu klasy GetZamowienie, pozycję adaptera elementu reprezentowanego przez ViewHolder

w odniesieniu do adaptera RecyclerView. Jest to następnie użyte poprzez zwrócenie właściwego klucza głównego zamówienia, dzięki czemu aplikacja wie który rekord usunąć.

Rysunek 4.25 Przejście do aktywności aktualizacji

```
aktualizuj.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        GetZamowienie getZamowienie = itemsArrayList.get(getAbsoluteAdapterPosition());  
        Intent intent = new Intent( packageContext: ZamowActivity.this, UpdateZamowienie.class);  
        String id = getZamowienie.getId();  
        intent.putExtra( name: "id", id);  
        startActivity(intent);  
        finish();  
    }  
});
```

Aktualizacja elementu listy odbywa się za pośrednictwem oddzielnej klasy i widoku. W celu przejście do nowej aktywności znowu została użyta metoda „setOnClickListener()” (Rysunek 4.25). Po raz kolejny, tak jak w przypadku usuwania, będzie potrzebny identyfikator rekordu, który użytkownik może chcieć zaktualizować, dlatego znowu numer identyfikacyjny zostaje pobrany i przypisany do zmiennej. Tym razem, do użycia odpowiedniego klucza głównego w kolejnej aktywności, użyta została metoda „putExtra()” pozwalająca na rozszerzenie operacji „Intent” o dodatkową, dowolnie nazwaną wartość.

Kolejno, aktywność „UpdateZamowienie”, posiada metodę „getZamowienieInfo” służącą wypisywaniu rekordów wcześniej rozwiniętego elementu z listy, aby użytkownik nie musiał wpisywać wcześniej ustalonych wartości.

Rysunek 4.26 Wczytywanie rekordów do formularza aktualizacji

```
private void getZamowienieInfo () {
    String id = getIntent().getExtras().getString( key: "id");
    try {
        DatabaseConnection databaseConnection = new DatabaseConnection();
        Connection conn = databaseConnection.connectionClass();
        if (conn!=null){
            String q = "SELECT * FROM Zamawianie WHERE id="+id;
            Statement stmt1 = conn.createStatement();
            ResultSet rs1 = stmt1.executeQuery(q);
            while (rs1.next()) {
                produkt_akt.setText(rs1.getString( s: "nazwa_prod"));
                ilosc_akt.setText(rs1.getString( s: "ilosc"));
                data_akt.setText(rs1.getString( s: "data_zamowienia"));
                data_obioru_akt.setText(rs1.getString( s: "data_odbioru"));
                cena_akt.setText(rs1.getString( s: "cena"));
                id_kierowcy_akt.setText(rs1.getString( s: "Kierowca_id"));
                id_maszyzny_akt.setText(rs1.getString( s: "Maszyna_id"));
            }
        }
        else
        {
            ConnectionResult="Check conn";
        }
    } catch (Exception ex){
        ex.printStackTrace();
    }
}
```

Źródło: opracowanie własne.

Metoda zaciąga dodany we wcześniejszej fazie numer identyfikacyjny zamówienia, który po połączeniu się z bazą zostaje użyty w zapytaniu, które wypisze wyłącznie informacje o zamówieniu o podanym „id”. Następnie przypisuje odpowiednie dane komponentom elementu formularza.

Następnie zostaje wywołana metoda „setOnClickListener()” poprzez przycisk zatwierdzający operacje. Tak jak w przypadku dodania nowego rekordu, metoda zwyczajnie sprawdza czy wszystkie pola zostały wypełnione a następnie przy pomocy zapytania do bazy danych, aktualizuje odpowiedni rekord. Jedyną zmianą jest podanie unikatowego numeru identyfikacyjnego w zapytaniu, pobranego we wcześniejszej aktywności.

4.2.7 Pobieranie daty z kalendarza

W celu łatwego i intuicyjnego wyboru daty, odbiorcy aplikacji wyświetla się kalendarz, w którym użytkownik może znaleźć i wybrać odpowiadający mu rok, miesiąc oraz dzień. Stąd do projektu została dodana nowa klasa DatePicker, posiadająca metodę „onCreateDialog” typu „Dialog”.

Rysunek 4.27 Metoda onCreateDialog w klasie DatePicker

```
@NonNull
@Override
public Dialog onCreateDialog(Bundle savedInstanceState){
    Calendar c = Calendar.getInstance();
    int year = c.get(Calendar.YEAR);
    int month = c.get(Calendar.MONTH);
    int day = c.get(Calendar.DAY_OF_MONTH);

    return new DatePickerDialog(getActivity(), (DatePickerDialog.OnDateSetListener) getActivity(), year, month, day);
}
```

Źródło: opracowanie własne.

Metoda składa się z:

- c, typu Calendar, pobierającym instancje kalendarza,
- year, month, day, zmiennych przechowujących wybrany kolejno rok, miesiąc i dzień miesiąca,
- new DatePickerDialog, inicjalizujący pojawienie się kalendarza i przyjmujący, kontekst aktywności, w tym przypadku będzie pobierał aktualnie używaną aktywność, następnie aktywność przypisaną do „DatePickerDialog.OnDateSetListenera”, która wskazuje, że użytkownik wybrał datę i ostatecznie wcześniej zadeklarowane wartości.

W kolejnym kroku należało wdrożyć możliwość korzystania z owej funkcji w odpowiednich klasach i stworzyć w nich metodę „onDateSet”. Metoda pobiera wybrane przez użytkownika dane, formatuje sposób w jaki będzie wyświetlała się data i przypisuje tą wartość do pola tekstowego. Do pojawienia się kalendarza została użyta metoda „setOnClickListener()” na odpowiednim edytowalnym polu tekstowym z datą, która wywołuje klasę DatePicker.

Rysunek 4.28 Metoda kalendarza onDateSet()

```
@Override
public void onDateSet(android.widget.DatePicker view, int year, int month, int dayOfMonth) {
    Calendar c = Calendar.getInstance();
    c.set(Calendar.YEAR, year);
    c.set(Calendar.MONTH, month);
    c.set(Calendar.DAY_OF_MONTH, dayOfMonth);
    SimpleDateFormat format = new SimpleDateFormat( pattern: "YYYY-MM-DD");
    String strDate = format.format(c.getTime());
    TextView data_tank = (TextView) findViewById(R.id.data_tank);
    data_tank.setText(strDate);
}
```

Źródło: opracowanie własne.

4.3 Testy

Testy funkcjonalne zostały przeprowadzone w programie „Android Studio”, umożliwiającym kontrolowanie działania na wbudowanych w program emulatorach, z możliwością wyboru różnych rozdzielczości oraz wersji systemu Android. Testy użytkowe i działania bazy danych były przeprowadzane przez autora równolegle z rozwojem aplikacji. Każda funkcjonalność została sprawdzona pod względem poprawnego pobierania i wyświetlania się odpowiednich danych.

Jedna z ostatecznych wersji aplikacji została przetestowana również przez docelowego jej użytkownika, z którym autor projektu ustalił wcześniej wstępne jego założenia. Z doniesień testera wynika, że program przeprowadzał wszystkie działania zgodnie z oczekiwaniami. Aplikacja została pochwalona pod względem wizualnym. Wszystkie widoki odpowiednio się wyświetlały oraz w przejrzysty sposób informowały testera o tym gdzie w danej chwili się znajduje, a adnotacje o błędnym wypełnieniu formularza pomogły mu w szybkim i odpowiednim jego poprawieniu. Pomimo, iż testerem była starsza i mniej doświadczona technologicznie osoba, nie miała ona problemów z poruszaniem się po aplikacji i jej odczytem. Ewentualne uwagi dotyczące programu zostały przedstawione po zakończeniu użytkowania, a następnie poprawione przez autora w następnych jego wersjach.

W trakcie testowania przeprowadzono testy:

- Funkcjonalne
 - testy poprawnego wyświetlania się elementów w aplikacji,
 - testy dodawania, usuwania i aktualizacji danych w aplikacji,
 - testy poprawnego działania aplikacji na różnych rozdzielczościach ekranu telefonu.
- Użytkowe
 - zweryfikowanie czy aplikacja spełnia nadane cele.
- Wydajnościowe
 - testy połączenia z bazą danych,
 - testy przechodzenia między panelami.

Przetestowane rozdzielczości:

- Pixel 1080x1920,
- Pixel 4 1080x2280,
- Pixel XL 1440x2560,
- Nexus 4 768x1280.

Zakończenie

Celem niniejszej pracy inżynierskiej było zaprojektowanie oraz implementacja aplikacji mobilnej, na system Android w wersji 8.0 lub wyższej, pomagającej w zarządzaniu parkiem maszynowym w gospodarstwie rolnym. Czynnikiem, dla którego powstała owa aplikacja, było zauważenie przez autora problemu oraz zmagania osoby, starającej się zarządzać parkiem maszyn i braku intuicyjnej aplikacji o podobnym charakterze. Rozmowa z doświadczoną w tej dziedzinie osobą z gospodarstwa rolnego oraz obserwacje własne autora pomogły ustalić koncept oraz potrzebne funkcjonalności programu.

Aplikacja w obecnej chwili pozwala na przeglądanie listy maszyn z ich specyfikacjami oraz dodawanie nowych wymian i zatankowań w nich zachodzących. Ponadto system umożliwia monitorowanie poziomu paliwa w zbiornikach znajdujących się na terenie gospodarstwa, jak również przeglądanie i dodawanie nowych dostaw cieczy do wybranego rezerwuaru. Program automatycznie aktualizuje ilość paliwa w zbiorniku w momencie dodania nowego zatankowania maszyny lub dostawy paliwa. Dodatkowo aplikacja daje możliwość wglądu w listę kierowców z niezbędnymi informacjami o nich oraz możliwość ich ewentualnej edycji. Ostatnią funkcjonalnością jest posiadanie przez system sekcji zamówień, edytowalnej listy przedmiotów lub części do maszyn w gospodarstwie potrzebnych pracownikom w danym momencie.

Jest to dopiero początkowa lecz działająca wersja aplikacji, a w przyszłości można ją będzie rozwijać na przeróżne sposoby. Jednym z nich mogłoby być wprowadzenie alertów informujących użytkownika o małym stanie paliwa w zbiorniku, czy zbliżającego się terminu przeglądu maszyny lub konieczności wymiany filtrów w maszynie, które powinny być wymieniane, w zależności od filtra, od 300 do 500 motogodzin maszyny. Kolejnym ciekawym rozwiązaniem mogłoby być wprowadzenie dokładniejszej analizy kosztów przeznaczonych na maszynę, z warunkiem na wymieniane w niej części czy oleje oraz ilość zatankowanego paliwa w danym okresie. Pomogłoby to nie tylko osobie zajmującej się zarządzaniem parkiem maszynowym, ale również całemu gospodarstwu/firmie w podejmowaniu decyzji, co do ewentualnego pozbycia się maszyny, która jest nieopłacalna. Nowoczesne maszyny posiadają również urządzenia do śledzenia np. ilości zasianego zboża. Aplikacja mogłaby używać owych danych poprzez

połączenie się z urządzeniem, a tym samym ułatwić analizę corocznych zbiorów. W przyszłości równie ciekawym rozwiązaniem, byłoby wprowadzenie systemu logowania dla różnych użytkowników gospodarstwa, przez co kierowcy mogliby sami wypełniać formularze dotyczące potrzebnych produktów, które następnie byłyby dodawane do listy zamówień w panelu przeznaczonym wyłącznie dla zarządzającego parkiem maszynowym.

Spis rysunków

Rysunek 1.1 Użytkownicy smartfonów względem populacji.....	8
Rysunek 1.2 Udziały w rynku mobilnych systemów operacyjnych na świecie od 2012 do 2022 roku	9
Rysunek 1.3 Architektura systemu Android	10
Rysunek 1.4 Kwartalne pobrania aplikacji oraz gier z Google Play w latach 2015-2022	12
Rysunek 1.5 Najpopularniejsze wersje systemu Android w ostatnich miesiącach	13
Rysunek 1.6 Aplikacja QRmaint	14
Rysunek 2.1 Diagram przypadków użycia	22
Rysunek 2.2 Model procesu przeglądania dodatkowych informacji o elemencie listy.....	31
Rysunek 2.3 Model procesu przeglądania dodatkowych informacji o kierowcy	31
Rysunek 2.4 Model procesu edycji daty przeglądu maszyny	32
Rysunek 2.5 Model procesu przeglądania dostaw paliwa do wybranego zbiornika	32
Rysunek 2.6 Model procesu dodania nowego tankowania maszyny	33
Rysunek 2.7 Model procesu dodania nowej dostawy do zbiornika	33
Rysunek 2.8 Model procesu dodania/edycji/usuwania zamówienia.....	34
Rysunek 2.9 Model procesu dodania/usuwania wymian części i olejów	35
Rysunek 2.10 Model procesu edycji kierowcy	36

Rysunek 3.1 Widok menu głównego aplikacji.....	37
Rysunek 3.2 Widok maszyn	38
Rysunek 3.3 Zmiana daty przeglądu maszyny	39
Rysunek 3.4 Widok wymian	40
Rysunek 3.5 Wymiany olejów	41
Rysunek 3.6 Widok dodania nowej wymiany	42
Rysunek 3.7 Widok edycji pola tekstowego	42
Rysunek 3.8 Widok edycji pola numerycznego	42
Rysunek 3.9 Widok edycji daty.....	42
Rysunek 3.10 Wymiany części.....	43
Rysunek 3.11 Widok dodania nowej wymiany części	44
Rysunek 3.12 Widok tankowania wszystkich maszyn w gospodarstwie.....	45
Rysunek 3.13 Widok dodania nowego tankowania	46
Rysunek 3.14 Widok Zbiorników w gospodarstwie	47
Rysunek 3.15 Widok dostaw paliwa po kliknięciu w dowolny zbiornik	47
Rysunek 3.16 Widok zamówień kierowców	48
Rysunek 3.17 Aktualizacja zamówienia.....	49
Rysunek 3.18 Widok kierowców i ich szczegółów	50
Rysunek 4.1 Model bazy danych.....	52
Rysunek 4.2 Tworzenie tabeli głównej w bazie danych	53

Rysunek 4.3 Dodanie nowego rekordu do tabeli głównej	53
Rysunek 4.4 Widok wywołanej tabeli	54
Rysunek 4.5 Utworzenie tabeli funkcjonalnej oraz dodanie do niej rekordu	55
Rysunek 4.6 Wyzwalacz aktualizujący zbiornik	55
Rysunek 4.7 Utworzenie wizualizacji.....	56
Rysunek 4.8 Wygląd wizualizacji.....	57
Rysunek 4.9 Utworzenie i wykonanie procedury dodania nowego rekordu	57
Rysunek 4.10 Kod tworzący i wywołujący procedurę aktualizującą rekord	58
Rysunek 4.11 Widoki wykorzystane w aplikacji	59
Rysunek 4.12 Kod klasy DatabaseConnection	60
Rysunek 4.13 Przykładowy kod umożliwiający przechodzenie pomiędzy menu a aktywnością	62
Rysunek 4.14 Kod przykładowego CardView w menu głównym.....	62
Rysunek 4.15 Wygląd elementu przykładowej listy RecyclerView z zatankowaniami w gospodarstwie	63
Rysunek 4.16 Budowa przykładowej klasy Get	64
Rysunek 4.17 Przykładowa klasa ViewHolder	65
Rysunek 4.18 Funkcje onCreateViewHolder, onBindViewHolder, getItemCount	65
Rysunek 4.19 Przykładowa metoda „doInBackground” w aplikacji	66
Rysunek 4.20 Przykładowa metoda onPostExecute() w aplikacji	67

Rysunek 4.21 Klasa Adapter listy typu ListView	68
Rysunek 4.22 Metoda setOnItemClickListener()	69
Rysunek 4.23 Przykładowy kod dodania nowego rekordu do bazy	70
Rysunek 4.24 Przykładowe usuwanie rekordu z listy	71
Rysunek 4.25 Przejście do aktywności aktualizacji	72
Rysunek 4.26 Wczytywanie rekordów do formularza aktualizacji	73
Rysunek 4.27 Metoda onCreateDialog w klasie DatePicker	74
Rysunek 4.28 Metoda kalendarza onDateSet()	75

Bibliografia

1. Mobirank, Digital i mobile w 2022 roku: <https://mobirank.pl/2022/01/27/digital-i-mobile-w-2022-roku/> [dostęp 28.01.2023]
2. Digital Information World, Google Android vs Apple iOS: Data Reveals Which OS Dominates Market Share: <https://www.digitalinformationworld.com/2022/04/google-android-vs-apple-ios-data.html> [dostęp 28.01.2023]
3. Android, Platform Architecture: <https://developer.android.com/guide/platform> [dostęp 28.01.2023]
4. BusinessOfApps, App Download Data (2023) <https://www.businessofapps.com/data/app-statistics/> [dostęp 28.01.2023]
5. AppBrain, Top Android OS versions: <https://www.appbrain.com/stats/top-android-sdk-versions> [dostęp 28.01.2023]
6. Android, Documentation: <https://developer.android.com/docs> [dostęp 28.01.2023]
7. Microsoft, SQL Server Documentation: <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16> [dostęp 28.01.2023]

8. Android Authority, From Android Market to Google Play: a brief history of the Play Store: <https://www.androidauthority.com/android-market-google-play-history-754989/> [dostęp 28.01.2023]
9. Drip, 13 Key Mobile Marketing Statistics You Need to Know in 2022: <https://www.drip.com/blog/mobile-marketing-statistics> [dostęp 28.01.2023]
10. The jTDS Project, Documentation: <https://jtds.sourceforge.net/doc.html> [dostęp 28.01.2023]
11. Katarzyna Śledziwska i Renata Włoch (2020). *Gospodarka cyfrowa. Jak nowe technologie zmieniają świat*. Warszawa: Wydawnictwa Uniwersytetu Warszawskiego.
12. Jan L. Harrington (2016). *Relational Database Design and Implementation*. New York City: Apress.
13. J. Paul Cardle (2017). *Android App Development in Android Studio: Java + Android Edition for Beginners*. Scotts Valley: CreateSpace
14. Joyce Farrell (2015). *Java Programming*. Boston: Cengage Learning.
15. Pablo Perea i Pau Giner (2017). *UX Design for Mobile*. Birmingham: Packt Publishing.
16. Stanisław Legutko (2007). *Podstawy eksploatacji maszyn i urządzeń*. Warszawa: Wydawnictwa Szkolne i Pedagogiczne.
17. Craig Larman (2005). *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development*. Hoboken: Prentice Hall PTR.
18. Mike Halsey (2019). *The IT Support Handbook: A How-To Guide to Providing Effective Help and Support to IT Users*. New York City: Apress.