

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA
INŻYNIERSKA

Aplikacja mobilna wspomagająca zarządzanie
kosztami eksploatacji pojazdów

Mobile application supporting management of
vehicles operating costs

AUTOR:

Jakub Zagrobelny

PROWADZĄCY PRACĘ:

dr inż. Jarosław Mierzwa, Wydział Elektroniki

OCENA PRACY:

WROCŁAW, 2016

Spis treści

Spis rysunków	5
Spis tabel	6
Spis listingów	7
Skróty	8
1. Wprowadzenie.....	9
1.1. Wstęp.....	9
1.2. Cel i zakres pracy	9
2. Istniejące rozwiązania	10
2.1. Fuelio	10
2.2. Drivvo – Zarządzani pojazdami	11
2.2.1. Przykładowe widoki	12
2.3. Zużycie paliwa - Fuel Manager	12
3. Wymagania funkcjonalne i нефункционалне.....	14
3.1. Wymagania funkcjonalne	14
3.2. Wymagania нефункционалне	14
3.2.1. Wymagania efektywnościowe i niezawodnościowe	14
3.2.2. Wymagania dotyczące bezpieczeństwa.....	15
4. Użyte technologie i narzędzia	16
4.1. Android Studio	16
4.2. Papyrus	16
4.3. SQLite.....	16
4.4. MPAndroidChart	16
5. Projekt	17
5.1. Diagram i opis przypadków użycia	17
5.2. Projekt bazy danych.....	21

5.2.1.	Tabela user	21
5.2.2.	Tabela vehicle	22
5.2.3.	Tabela cost.....	22
5.3.	Architektura aplikacji	23
5.4.	Interfejs graficzny	23
5.4.1.	Fragment główny - HomeFragment	24
5.4.2.	Fragment dodawania kosztu - AddCostFragment	24
5.4.3.	Fragment historia - HistoryFragment	25
5.5.	Diagram klas	26
5.5.1.	Interfejsy.....	27
5.5.2.	Klasy testujące.....	27
5.5.3.	Pozostałe klasy	27
6.	Implementacja	28
6.1.	Opis kluczowych klas aplikacji	28
6.1.1.	Cost.....	28
6.1.2.	ChartEntry	28
6.1.3.	DatabaseHelper	29
6.1.4.	Klasy i interfejsy do testów	29
6.2.	Algorytmy obliczeniowe	30
6.2.1.	Obliczanie całkowitego kosztu kilometra	30
6.2.2.	Średnie spalanie.....	30
6.2.3.	Obliczanie kosztu kilometra z ostatnich trzydziestu dni	30
6.3.	Komunikacja z bazą danych	30
6.3.1.	Utworzenie bazy.....	30
6.3.2.	Nawiązanie połączenia z bazą danych	31
6.3.3.	Pobranie kosztów z ostatnich 30 dni	31
6.3.4.	Dodanie nowego kosztu	31

6.3.5.	Aktualizacja wybranego kosztu	32
6.3.6.	Usunięcie wybranego kosztu.....	32
6.3.7.	Import i eksport bazy danych	32
7.	Testy aplikacji	33
7.1.	Testy jednostkowe z wykorzystaniem Mockito	33
7.1.1.	Opis wykonywanych testów.....	33
7.1.2.	Wnioski z testów	33
7.2.	Testy manualne na emulatorze	34
7.2.1.	Opis wykonywanych testów.....	34
7.2.2.	Wnioski z testów	34
7.3.	Testy manualne na urządzeniu mobilnym	34
7.3.1.	Opis wykonywanych testów.....	34
7.3.2.	Wnioski z testów	35
8.	Podsumowanie	36
8.1.	Wnioski.....	36
8.2.	Możliwości rozwoju	36
9.	Literatura i źródła	37
10.	Załącznik – instrukcja	38
10.1.	Zawartość płyty CD	38
10.2.	Instrukcja korzystania z aplikacji.....	38
10.2.1.	Intro	38
10.2.2.	Ekran główny.....	40
10.2.3.	Dodawanie tankowania i kosztu	40
10.2.4.	Historia	41
10.2.5.	Edycja kosztu.....	42
10.2.6.	Import i Eksport.....	42

Spis rysunków

Rysunek 1 Widok strony głównej	11
Rysunek 2 Widok historii.....	11
Rysunek 3 Widok menu bocznego	12
Rysunek 4 Widok Historii	12
Rysunek 5 Widok ekranu głównego	13
Rysunek 6 Widok menu bocznego	13
Rysunek 7 Diagram przypadków użycia.....	17
Rysunek 8 Schemat bazy danych	21
Rysunek 9 HomeFragment	24
Rysunek 10 AddCostFragment	25
Rysunek 11 HistoryFragment.....	25
Rysunek 12 Wybór kategorii filtrowania	25
Rysunek 13 Diagram zależności klas	26
Rysunek 14 Intro - ekran pierwszy	38
Rysunek 15 Intro - Uprawnienia	38
Rysunek 16 Intro - Zapytanie o uprawnienia	39
Rysunek 17 Intro - Wprowadzenie danych użytkownika	39
Rysunek 18 Intro - Wprowadzenie danych pojazdu	39
Rysunek 19 Ekran główny	40
Rysunek 20 Menu boczne	40
Rysunek 21 Dodaj tankowanie.....	41
Rysunek 22 Dodaj koszt.....	41
Rysunek 23 Widok historii.....	41
Rysunek 24 Okno dialogowe wyboru kategorii	41
Rysunek 25 Edytuj koszt.....	42
Rysunek 26 Import i Eksport	42

Spis tabel

Tabela 1 Utwórz konto	18
Tabela 2 Dodaj pojazd.....	18
Tabela 3 Dodaj koszt.....	19
Tabela 4 Edytuj koszt.....	19
Tabela 5 Usuń koszt	20
Tabela 6 Eksportuj dane	20
Tabela 7 Importuj dane	21

Spis listingów

Listing 1 Konstruktor klasy Cost	28
Listing 2 Klasa ChartEntry	29
Listing 3 Uzyskanie połączenia z bazą danych	31
Listing 4 Wyznaczenie zakresu dat	31
Listing 5 Dodawanie nowego kosztu	31
Listing 6 Usunięcie kosztu za pomocą id	32
Listing 7 Import pliku bazy danych	32

Skróty

API (ang. *Application Programming Interface*)

XML (ang. *eXtensible Markup Language*)

IDE (ang. *Integrated Development Environment*)

1. Wprowadzenie

1.1. Wstęp

Rzeczony technologii komputerowych rozpoczynał się od ogromnych komputerów zajmujących powierzchnię nawet kilku pokoi, a obecnie urządzenia kilkaset razy mniejsze i jednocześnie kilkaset razy bardziej wydajne znajdują się w kieszeni prawie każdego człowieka. Popularność technologii mobilnych sprawiła, że człowiek może w każdej chwili wyszukać informację w sieci lub skorzystać z pomocnej aplikacji. Właśnie ta powszechność smartfonów, a także fakt niezadowolenia z dostępnych na rynku aplikacji wspomagających prowadzenie budżetu pojazdu, wpłynęła na mój wybór tematu pracy inżynierskiej.

1.2. Cel i zakres pracy

Celem pracy jest stworzenie aplikacji mobilnej działającej na smartfonach z systemem Android oraz udokumentowanie kolejnych etapów, czyli projektowania, implementacji oraz testowania. Celem pracy jest także poznanie podstaw technologii programowania na system Android w oparciu o język Java, a także zaznajomienie się ze środowiskiem programowania i jego narzędziami.

Niniejsza praca zawiera dokumentację projektu aplikacji mobilnej, opis jej testów oraz instrukcję obsługi. Na pracę składa się dziesięć rozdziałów. Pierwszy rozdział zawiera wstęp pracy. W rozdziale drugim znajduje się przedstawienie trzech już istniejących rozwiązań. Rozdziały od drugiego do piątego zawierają projekt techniczny aplikacji, czyli wymagania funkcjonalne i нефункционалне, omówienie technologii oraz sam projekt struktury aplikacji oraz jej funkcji. Rozdziały szósty i siódmy omawiają implementację oraz testowanie aplikacji. W rozdziale ósmym jest podsumowanie projektu, a w dziewiątym rozdziale znajduje się wykorzystana literatura. Ostatni rozdział zawiera opis zawartości załączonej do pracy płyty CD oraz instrukcję instalacji oraz korzystania z aplikacji.

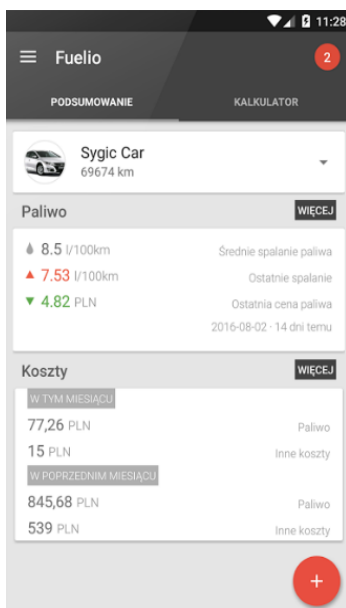
2. Istniejące rozwiązania

W niniejszym rozdziale przedstawię trzy wybrane popularne aplikacje zmierzające się tematem podobnym do mojego, dostępne w Sklepie Play. Na końcu podsumuję, które elementy poszczególnych aplikacji znajdują się w mojej aplikacji, a których nie będzie oraz co nowego wprowadzam, co nie znajduje się w omówionych rozwiązaniach.

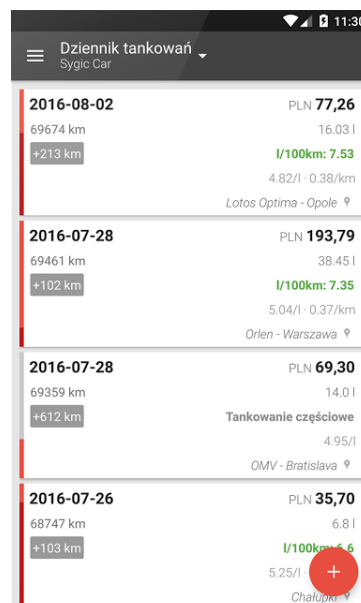
2.1. Fuelio

Jest to aplikacja oparta na stylu aplikacji Material Art promowanym przez twórcę systemu Android, czyli firmę Google. Przykładowe widoki aplikacji są widoczne na rysunkach 1 i 2. Posiada ona następujące funkcjonalności:

- a) wgląd w statystyki liczbowe i wykresy dla osobno tankowań oraz innych kosztów podzielonych na kategorie,
- b) wgląd osobno w dziennik tankowań i dziennik kosztów,
- c) generowanie raportów zawierających dane wybrane przez użytkownika,
- d) import i eksport manualny lub automatyczny przy wykorzystaniu Google Drive, Dropbox albo lokalnie pliku CSV,
- e) kalkulator kosztów podróży,
- f) możliwość wyboru jednostek miar i waluty, języka, formatu daty, motywu kolorystycznego,
- g) opiniowanie stacji benzynowych z wglądem na ich lokalizację na mapie,
- h) wgląd w zestawienie tankowań na mapie ze względu na lokalizację stacji benzynowej,
- i) przypomnienia o przedłużeniu ubezpieczenia, wykonaniu przeglądu rejestracyjnego lub innego zdefiniowanego kosztu,
- j) możliwość dodawania wielu pojazdów i wyboru dla każdego osobnej formy zasilania samochodu, ilości źródeł, a także jednostek odległości, pojemności i średniego spalania.



Rysunek 1 Widok strony głównej



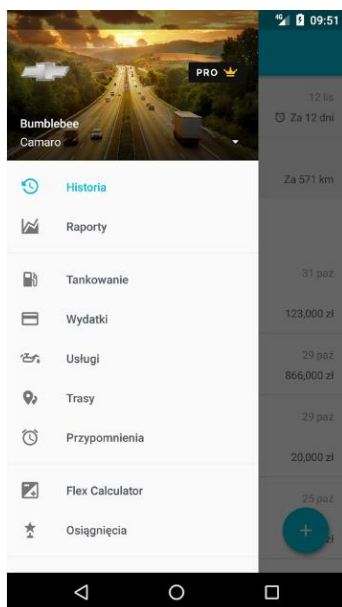
Rysunek 2 Widok historii

2.2. Drivvo – Zarządzani pojazdami

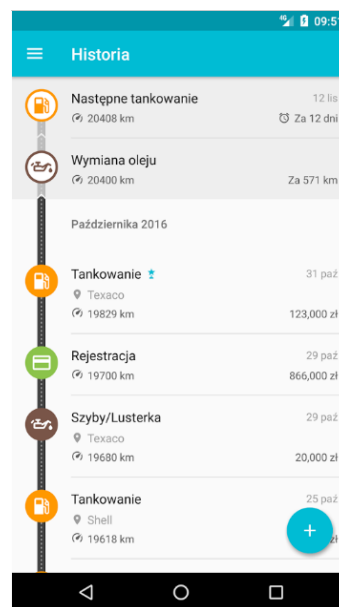
Jest to aplikacja oparta na stylu aplikacji Material Art promowanym przez twórcę systemu Android, czyli firmę Google. Przykładowe widoki aplikacji są widoczne na rysunkach 3 i 4. Zawiera następujące funkcjonalności:

- wgląd w raporty liczbowe i wykresy ogólne oraz osobno dla tankowań, wydatków i usług,
- wgląd w historię ogólną oraz osobno w historię tankowań, wydatków i usług,
- import i eksport automatyczny poprzez synchronizację za pomocą konta Google, Facebook lub konta własnego Drivvo,
- zbieranie danych na temat podróży,
- możliwość wyboru jednostek miar i waluty, języka, formatu daty, motywu kolorystycznego,
- przypomnienia dla wydatków lub usług,
- możliwość dodawania wielu pojazdów i wyboru dla każdego osobnej formy zasilania samochodu, ilości źródeł, a także jednostek odległości, pojemności i średniego spalania.

2.2.1. Przykładowe widoki



Rysunek 3 Widok menu bocznego



Rysunek 4 Widok Historii

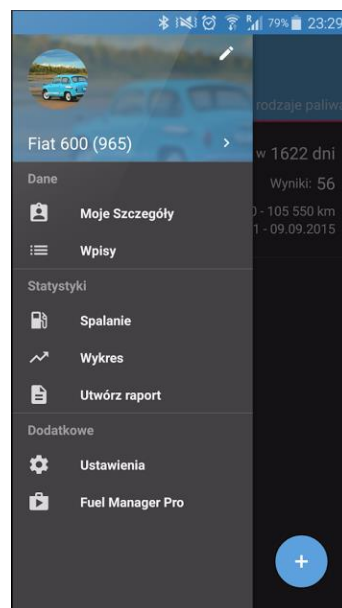
2.3. Zużycie paliwa - Fuel Manager

Jest to aplikacja oparta na stylu aplikacji Material Art promowanym przez twórcę systemu Android, czyli firmę Google. Zawiera następujące funkcjonalności:

- wgląd w raporty liczbowe i wykresy ogólne,
- wgląd w historię wydatków,
- funkcja kalkulatora podróży,
- szacowanie ilości paliwa pozostałego w baku,
- import i eksport automatyczny poprzez synchronizację za pomocą konta Google, Facebook lub konta własnego Drivvo,
- możliwość wyboru jednostek miar i waluty, języka, formatu daty, motywu kolorystycznego,
- możliwość dodawania wielu pojazdów i wyboru dla każdego osobnej formy zasilania samochodu, ilości źródeł, a także jednostek odległości, pojemności i średniego spalania.



Rysunek 5 Widok ekranu głównego



Rysunek 6 Widok menu bocznego

3. Wymagania funkcjonalne i нефunkcjonalne

3.1. Wymagania funkcjonalne

Aplikacja realizowana w ramach projektu będzie posiadała następujące funkcjonalności:

- a) zbieranie danych o kosztach związanych z eksploatacją pojazdu użytkownika, danych samego użytkownika,
- b) analiza danych dotyczące kosztów i wyświetlanie podstawowych statystyk oraz wykresu na ekranie głównym,
- c) przeglądanie i edycja dodanych kosztów,
- d) eksportu do pliku stanu całej bazy danych w dowolnym momencie,
- e) import stanu bazy danych i zastąpienie obecnego.

3.2. Wymagania нефunkcjonalne

3.2.1. Wymagania efektywnościowe i niezawodnościowe

Główne założenia efektywnościowe i niezawodnościowe aplikacji mobilnej realizowanej w ramach projektu:

- a) przystosowanie do wyświetlania na smartfonach z systemem mobilnym Android w wersji 4.1 (API 16) i wyższej. Aplikacja nie będzie przystosowana do wyświetlania na tabletach, więc uruchomienie na urządzeniu tego typu może spowodować nieestetyczne lub nieczytelne przeskalowanie elementów – dotyczy szczególnie tabletów z domyślnym trybem poziomym,
- b) prostota i intuicyjność widoków aplikacji umożliwiające sprawne zapoznanie się z aplikacją i wygodne korzystanie z niej,
- c) uruchomienie aplikacji powinno trwać możliwie krótko - nie dłużej niż 10 sekund,
- d) przechodzenie pomiędzy widokami powinno być płynne i szybkie,
- e) poprawność wprowadzanych danych powinna być kontrolowana,
- f) import i eksport całej bazy danych powinien odbywać się szybko i bez błędów.

3.2.2. Wymagania dotyczące bezpieczeństwa

Aplikacja ze względu na przechowywane dane będzie posiadała szyfrowaną bazę danych. Wejście do aplikacji nie będzie wymagało hasła, ale użytkownik może wykorzystać zewnętrzne aplikacji do hasłowania dostępu do aplikacji oraz szyfrowane odblokowanie telefonu.

Aplikacja nie będzie nigdzie wysyłać danych podawanych przez użytkownika. Eksport i import odbywa się wyłącznie lokalnie.

4. Użyte technologie i narzędzia

4.1. Android Studio

Zintegrowane środowisko programowania służące do tworzenia aplikacji mobilnych na system Android stworzone przez Google oparte na platformie IntelliJ. Środowisko zawiera narzędzia wspierające tworzenie aplikacji takie jak graficzny edytor widoków, zintegrowany emulator urządzeń mobilnych (domyślnie urządzenia serii Nexus) z możliwością pobrania emulatora dowolnego urządzenia z wybranym systemem, debbuger. Jest to jedno z najbardziej popularnych środowisk służących do programowania na system Android. Głównymi zaletami tego środowiska jest jego popularność, dzięki czemu dostępne jest wiele poradników programowania i obsługi środowiska, a także czytelna funkcjonalność debbugera.

4.2. Papyrus

Środowisko służące do tworzenia diagramów UML oparte na środowisku programistycznym Eclipse. Do zalet można zaliczyć prostotę interfejsu oraz czytelność funkcjonalności. Podczas projektowania korzystałem z wersji Papyrus Neon (4.6.0).

4.3. SQLite

Technologia bazodanowa przechowująca dane w jednym odpowiednio sformatowanym pliku. Biblioteki służące do obsługi SQLite zostały dołączone do API już od jego pierwszej wersji. Zaletą tej technologii jest brak konieczności instalowania serwera.

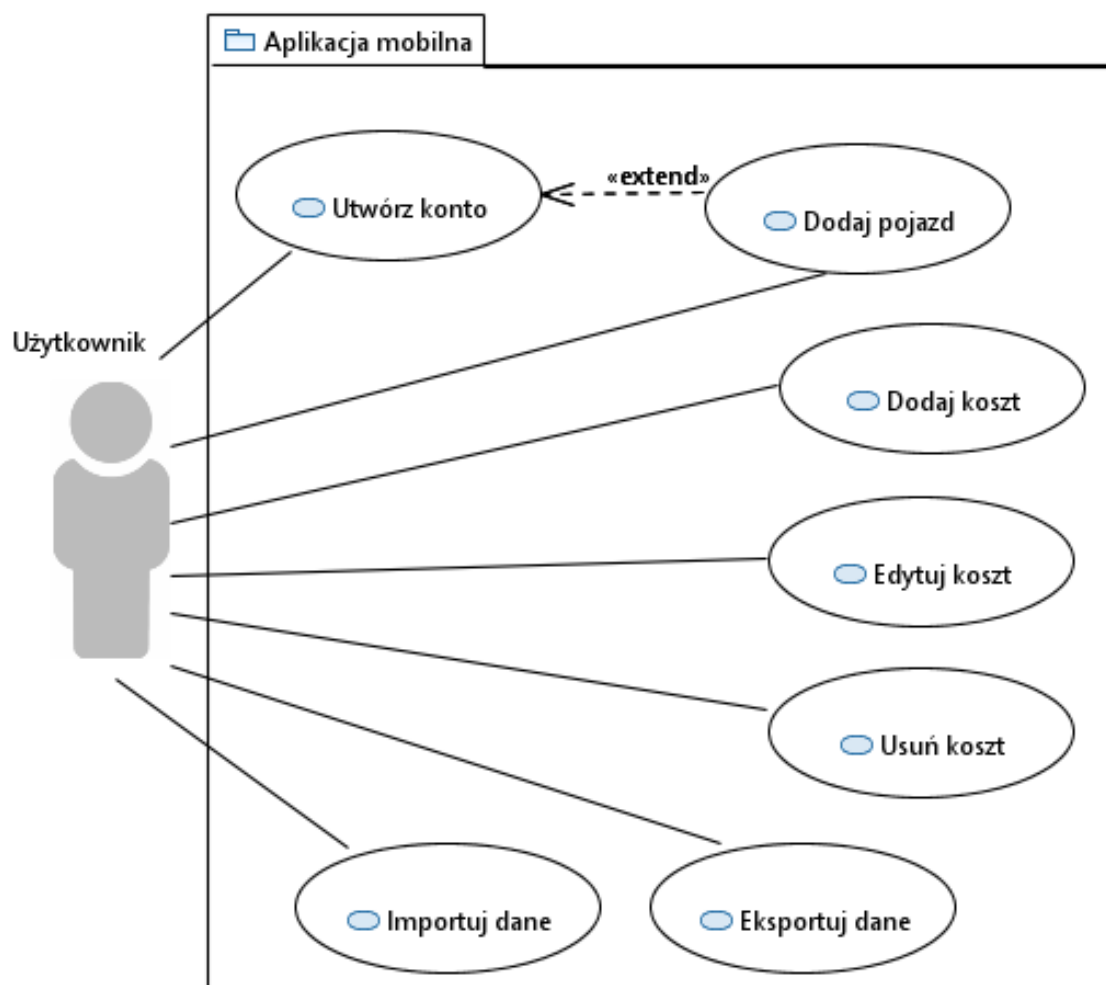
4.4. MPAndroidChart

Biblioteka zawierająca gotowe klasy umożliwiające wygodne tworzenie estetycznych wykresów różnego typu tworzona przez Philippa Jahode. Do zalet biblioteki można zaliczyć wygląd wykresów, czytelną dokumentację oraz popularność. Podczas implementacji aplikacji korzystałem z wersji biblioteki MPAndroidChart v2.1.6. Biblioteka jest dostępna w repozytorium <https://github.com/PhilJay/MPAndroidChart>.

5. Projekt

5.1. Diagram i opis przypadków użycia

Funkcje aplikacji zostały zaprojektowane i przedstawione na poniższym diagramie przypadków użycia. Opis widocznych przypadków użycia znajduje się w dalszej części podrozdziału w osobnych tabelach.



Rysunek 7 Diagram przypadków użycia

Przypadek użycia o nazwie „Utwórz konto” ma miejsce podczas intro początkowego w widoku Użytkownik. Został opisany w poniższej tabeli.

Tabela 1 Utwórz konto

Kontekst zdaniowy	Pierwsze uruchomienie aplikacji i utworzenie bazy danych.
Warunki wstępne	Aplikacja nie została wcześniej uruchomiona lub nie istnieje baza danych aplikacji.
Warunek pomyślnego zakończenia	Utworzenie całej bazy danych przygotowanej pod dodawanie pojazdów wraz danymi użytkownika.
Warunek niepomyślnego zakończenia	Nieutworzenie bazy danych.
Aktor	Użytkownik.
Wyzwalacz	Pierwsze uruchomienie aplikacji.
Główny przebieg	1. Użytkownik przechodzi do widoku dodawania użytkownika w intro. 2. Użytkownik wpisuje swoje imię oraz zaznacza wybrane pola i zatwierdza przyciskiem dalej. 3. Dane zostają zapisane.
Rozszerzenie	4. Po przycisku dalej aplikacja przechodzi do przypadku użycia dodawania pojazdu (Extend::Dodaj pojazd).

Przypadek użycia o nazwie „Utwórz konto” ma miejsce podczas intro początkowego w widoku Pojazd. Został opisany w poniższej tabeli.

Tabela 2 Dodaj pojazd

Kontekst zdaniowy	Dodanie nowego pojazdu.
Warunki wstępne	Baza danych jest utworzona.
Warunek pomyślnego zakończenia	Dodanie pojazdu do bazy.
Warunek niepomyślnego zakończenia	Niedodanie pojazdu do bazy.
Aktor	Użytkownik.
Wyzwalacz	Pierwsze uruchomienie aplikacji.
Główny przebieg	1. Użytkownik przechodzi do widoku dodawania pojazdu w intro. 2. Użytkownik wpisuje nazwę pojazdu, oraz wybiera rodzaj paliwa/paliw, pojemność baku/baków, ilość baków i klika Zakończ 3. Następuje utworzenie bazy danych i wprowadzenie do niej danych z przypadku użycia Utwórz konto.
Rozszerzenie	Brak.

Kolejna tabela opisuje przypadek użycia Dodaj koszt. Jest on uruchamiany z poziomu fragmentu głównego – rysunek 9 – poprzez odpowiednie przyciśnięcie przycisku pływającego „plus”.

Tabela 3 Dodaj koszt

Kontekst zdaniowy	Dodanie nowego kosztu.
Warunki wstępne	Baza danych została założona.
Warunek pomyślnego zakończenia	Dodanie kosztu do bazy danych.
Warunek niepomyślnego zakończenia	Niedodanie kosztu do bazy danych.
Aktor	Użytkownik.
Wyzwalacz	Krótkie lub długie kliknięcie przycisku „plus” widocznego w głównym widoku.
Główny przebieg	<ol style="list-style-type: none"> 1. Użytkownik krótko klika przycisk plus. 2. Przejście do ekranu dodawania tankowania. 3. Użytkownik wpisuje wymagane wartości parametrów kosztu w zależności od wybranej kategorii i zatwierdza dodanie kosztu. 4. Weryfikacja poprawności danych. 5. (Dane poprawne) Dodanie tankowania do bazy. 5. (Dane niepoprawne) Powrót do punktu 3.
Alternatywny przebieg	<ol style="list-style-type: none"> 1. Użytkownik długo klika przycisk plus. 2. Przejście do ekranu dodawania kosztu. 3. Użytkownik wpisuje wymagane wartości parametrów kosztu w zależności od wybranej kategorii i zatwierdza dodanie kosztu. 4. Weryfikacja poprawności danych 5. (Dane poprawne) Dodanie tankowania do bazy. 5. (Dane niepoprawne) Powrót do punktu 3.

Poniższa tabela opisuje przypadek użycia Edytuj koszt uruchamiany z widoku Historia – rysunek 11.

Tabela 4 Edytuj koszt

Kontekst zdaniowy	Edycja parametrów wybranego kosztu.
Warunki wstępne	Baza danych została założona, koszt istnieje w bazie danych.
Warunek pomyślnego zakończenia	Zaktualizowanie parametrów kosztu w bazie.
Warunek niepomyślnego zakończenia	Niezaktualizowanie parametrów kosztu w bazie.
Aktor	Użytkownik.
Wyzwalacz	Wybranie kosztu z listy wyświetlanej w zakładce Historia, a następnie kliknięcie przycisku edycji.
Główny przebieg	<ol style="list-style-type: none"> 1. Użytkownik klika wybrany koszt z listy w zakładce Historia. 2. Użytkownik zmienia wybrane wartości parametrów kosztu, które zależą od kategorii kosztu, a następnie klika przycisk Zapisz. 3. Koszt zostaje zaktualizowany w bazie danych.
Alternatywny przebieg	<ol style="list-style-type: none"> 2. Użytkownik zmienia wybrane wartości parametrów kosztu, które zależą od kategorii kosztu, ale nie zatwierdza zmian lub naciska przycisk wstecz. 3. Koszt nie zostaje zaktualizowany w bazie danych.

Przypadek użycia Usun koszt – opisany poniżej – uruchamia się identycznie jak poprzedni przypadek użycia, ponieważ funkcja usuwania kosztu znajduje się w tym samym widoku co funkcja edycji – wygląd został przedstawiony i opisany w podrozdziale 10.2.5.

Tabela 5 Usun koszt

Kontekst zdaniowy	Usunięcie wybranego kosztu.
Warunki wstępne	Koszt znajduje się w bazie danych.
Warunek pomyślnego zakończenia	Usunięcie kosztu z bazy danych.
Warunek niepomyślnego zakończenia	Nieusunięcie kosztu z bazy danych.
Aktor	Użytkownik.
Wyzwalacz	Wybranie kosztu z listy wyświetlanej w zakładce Historia, a następnie kliknięcie przycisku usuwania.
Główny przebieg	1. Użytkownik klika wybrany koszt z listy w zakładce Historia. 2. Użytkownik klika przycisk Usun. 3. Koszt zostaje usunięty z bazy danych.
Alternatywny przebieg	2. Użytkownik anuluje usuwanie kosztu poprzez przycisk wstecz. 3. Koszt nie zostaje usunięty z bazy danych.

Eksportowanie odbywa się we fragmencie Import/Eksport, którego opis i wygląd znajduje się w podrozdziale 10.2.6. Opis tego przypadku użycia znajduje się w poniższej tabeli.

Tabela 6 Eksportuj dane

Kontekst zdaniowy	Eksport bazy danych do pliku.
Warunki wstępne	Baza danych istnieje.
Warunek pomyślnego zakończenia	Wyeksportowanie pliku bazy danych do pliku dostępnego lokalnie.
Warunek niepomyślnego zakończenia	Niewyeksportowanie bazy danych.
Aktor	Użytkownik.
Wyzwalacz	Wybranie przycisku Eksport w zakładce Import/Eksport.
Główny przebieg	1. Użytkownik wybiera przycisku Eksport w zakładce Import/Eksport. 2. Baza danych zostaje wyeksportowana do pliku dostępnego lokalnie.
Rozszerzenie	Brak

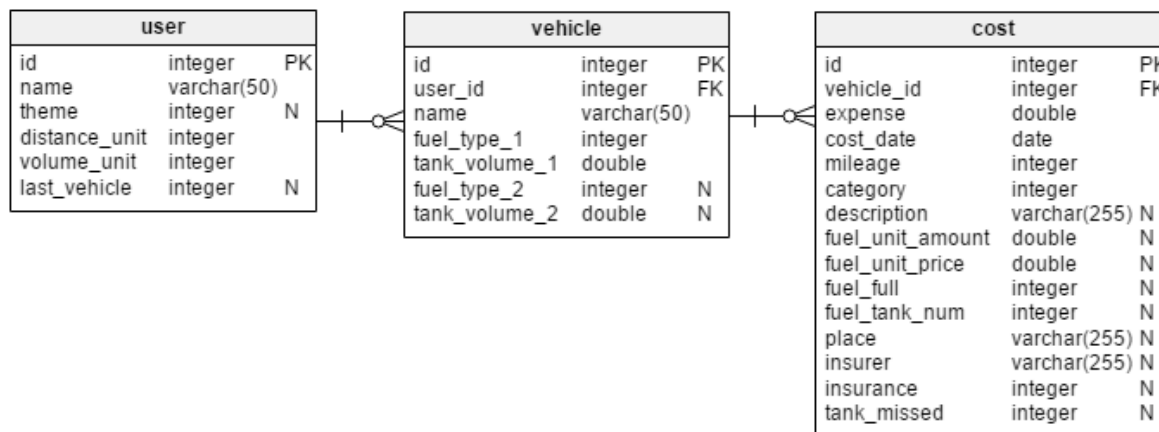
Ostatnim przypadkiem użycia jest importowanie danych. Dostęp do tej funkcji jest analogiczny do eksportowania i znajduje się w tym samym fragmencie. Został on opisany w poniższej tabeli.

Tabela 7 Importuj dane

Kontekst zdaniowy	Import bazy danych z pliku lokalnego.
Warunki wstępne	Plik do importu znajduje się w lokalizacji /sdcard/CarCost
Warunek pomyślnego zakończenia	Zaimportowanie pliku do domyślnej ścieżki bazodanowej.
Warunek niepomyślnego zakończenia	Niezaimportowanie wszystkich danych z archiwum zip do bazy danych.
Aktor	Użytkownik.
Wyzwalacz	Wybranie przycisku Import w zakładce Import/Eksport.
Główny przebieg	1. Użytkownik wybiera przycisku Import w zakładce Import/Eksport. 3. Baza danych zostaje zaimportowana.
Rozszerzenie	2.a. Użytkownik anuluje import w oknie dialogowym. 3.a. Baza danych nie zostaje zaimportowana.

5.2. Projekt bazy danych

Baza danych została zaprojektowana w oparciu o technologię bazodanową SQLite. Z założenia aplikacja ma być prosta i czytelna, więc także baza danych składa się z tylko trzech powiązanych ze sobą tabel: user, vehicle oraz cost. Poniżej znajduje się schemat bazy danych wykorzystywanej w aplikacji.



Rysunek 8 Schemat bazy danych

5.2.1. Tabela user

Tabela user zawiera tylko jeden wiersz i zawiera dane, które opisują użytkownika następującymi parametrami:

- a) name - nazwa użytkownika,

- b) theme – numer motywu wybranego przez użytkownika (nie ma zastosowania w obecnej wersji aplikacji – do wykorzystania przy dalszym rozwoju),
- c) distance_unit – wybrana przez użytkownika jednostka odległości,
- d) volume_unit – wybrana przez użytkownika jednostka objętości,
- e) last_vehicle – numer ostatnio wybranego pojazdu (nie ma zastosowania w obecnej wersji aplikacji – do wykorzystania przy dalszym rozwoju).

5.2.2. Tabela vehicle

Tabela vehicle w obecnej wersji aplikacji zawiera tylko jeden wiersz. Została stworzona z myślą o rozwoju aplikacji pod kątem prowadzenia kilku pojazdów. Zawiera następujące kolumny:

- a) user_id – id użytkownika z którym jest powiązany pojazd,
- b) name – nazwa pojazdu,
- c) fuel_type_1 – typ paliwa przechowywany w głównym baku,
- d) tank_volume_1 – pojemność głównego baku,
- e) fuel_type_2 – (opcjonalny) typ paliwa przechowywany w dodatkowym baku,
- f) tank_volume_2 – (opcjonalny) pojemność dodatkowego baku.

5.2.3. Tabela cost

Najbardziej istotną tabelą jest tabel cost. Zawiera ona dane wszystkich dodanych przez użytkownika kosztów z uwzględnieniem kategorii oraz charakterystycznych dla kategorii parametrów. Zawiera następujące kolumny:

- a) vehicle_id – id pojazdu z którym jest powiązany koszt,
- b) expense – całkowita wartość kosztu,
- c) cost_date – data poniesienia kosztu,
- d) mileage – przebieg w momencie poniesienia kosztu,
- e) category – numer kategorii kosztu,
- f) description – (opcjonalny) opis kosztu,
- g) fuel_unit_amount – ilość zatankowanego paliwa,
- h) fuel_unit_price – cena za jednostkę zatankowanego paliwa
- i) fuel_full – informacja, czy tankowanie było do pełna
- j) fuel_tank_num – numer baku, który był napełniany
- k) place – (opcjonalny) miejsce poniesienia kosztu
- l) insurer – nazwa ubezpieczyciela

- m) insurance – typ ubezpieczenia
- n) tank_missed – czy pominięto poprzednie tankowanie

5.3. Architektura aplikacji

Aplikacja realizowana w ramach pracy dyplomowej opiera się na architekturze fragmentów. Istnieją dwie aktywności: IntroActivity oraz MainActivity.

Pierwsza z wymienionych aktywności uruchamia się tylko raz przy pierwszym uruchomieniu aplikacji po instalacji. Warunkiem jej uruchomienia jest brak utworzonej bazy danych. Poprzez cztery fragmenty prosi o uprawnienia do pisania i czytania do pamięci, a także gromadzi dane użytkownika oraz pojazdu, które następnie umieszcza w nowoutworzonej bazie danych.

MainActivity jest natomiast główną aktywnością obsługującą pozostałe fragmenty aplikacji zapewniające dodawanie, edycję i usuwanie kosztów, oraz wyświetlanie ich na wykresie. W tej aktywności odbywa się zmiana poszczególnych fragmentów, a także w jej kontekście odbywają się zdarzenia.

Za kontakt z bazą danych odpowiedzialna jest klasa DatabaseHelper. Dodatkowo zostały utworzone klasy wspomagające tymczasowe przechowywanie danych potrzebnych np. do wyświetlenia wykresu.

5.4. Interfejs graficzny

Interfejs graficzny aplikacji zgodnie z założeniem jest prosty i czytelny. Składa się na niego sześć fragmentów przynależnych do MainActivity:

- o) AddFuelFragment – fragment odpowiedzialny za dodawanie kosztów tankowania,
- p) EditCostFragment – fragment używany pod czas edycji kosztu – wywoływany przez HistoryFragment,
- q) ImExFragment – fragment obsługuje import i eksport bazy danych,
- r) HomeFragment,
- s) HistoryFragment,
- t) AddCostFragment.

Pozostałe fragmenty nie posiadają własnych plików Java, ale są to widoki XML, z który IntroActivity pobiera wcześniej opisane dane.

W dalszej części rozdziału omówię bardziej szczegółowo trzy najważniejsze fragmenty, oraz zaprezentuję zrzuty ekranu przedstawiające ich wygląd.

5.4.1. Fragment główny - HomeFragment

Fragment główny – rysunek 9 – uruchamia się na początku regularnego uruchomienia aplikacji i tuż po zakończeniu IntroActivity przy pierwszym uruchomieniu aplikacji. Są na nim widoczne trzy podstawowe statystyki: średni koszt przejechanego kilometra na podstawie wszystkich danych z bazy, średnie spalanie silnika na 100 kilometrów, średni przejechanego koszt kilometra na podstawie wszystkich danych z zeszłego miesiąca.



Rysunek 9 HomeFragment – fragment główny

5.4.2. Fragment dodawania kosztu - AddCostFragment

Fragment ten widoczny na rysunku 10 służy do dodawania kosztu z uwzględnieniem wyboru kategorii:

- a) konserwacja i naprawy,
- u) przegląd,
- v) ubezpieczenie,
- w) inne.

W zależności od wybranej kategorii, zmieniają się dostępne pola do wypełnienia np. w przypadku kategorii Ubezpieczenie pojawiają się pola podana nazwy ubezpieczyciela oraz typu ubezpieczenia – OC/AC.

Rysunek 10 AddCostFragment – fragment dodawania kosztu

5.4.3. Fragment historia - HistoryFragment

Ten fragment przedstawia listę dodanych przez użytkownika kosztów z możliwością filtrowania ich ze względu na kategorie oraz z uwzględnienie ram czasowych. Domyślnie ustawione są wszystkie kategorie oraz zakres dat od obecnej daty do miesiąca wstecz. Z poziomu tego fragmentu można także kliknąć dowolny koszt i edytować jego właściwości lub usunąć z bazy danych.

Historia	
Od:	2017-01-03
Do:	2017-02-03
KATEGORIE	
SZUKAJ	
Tankowanie	50.0
2017-02-03	50
Konservacja i naprawy	20.0
2017-02-03	300

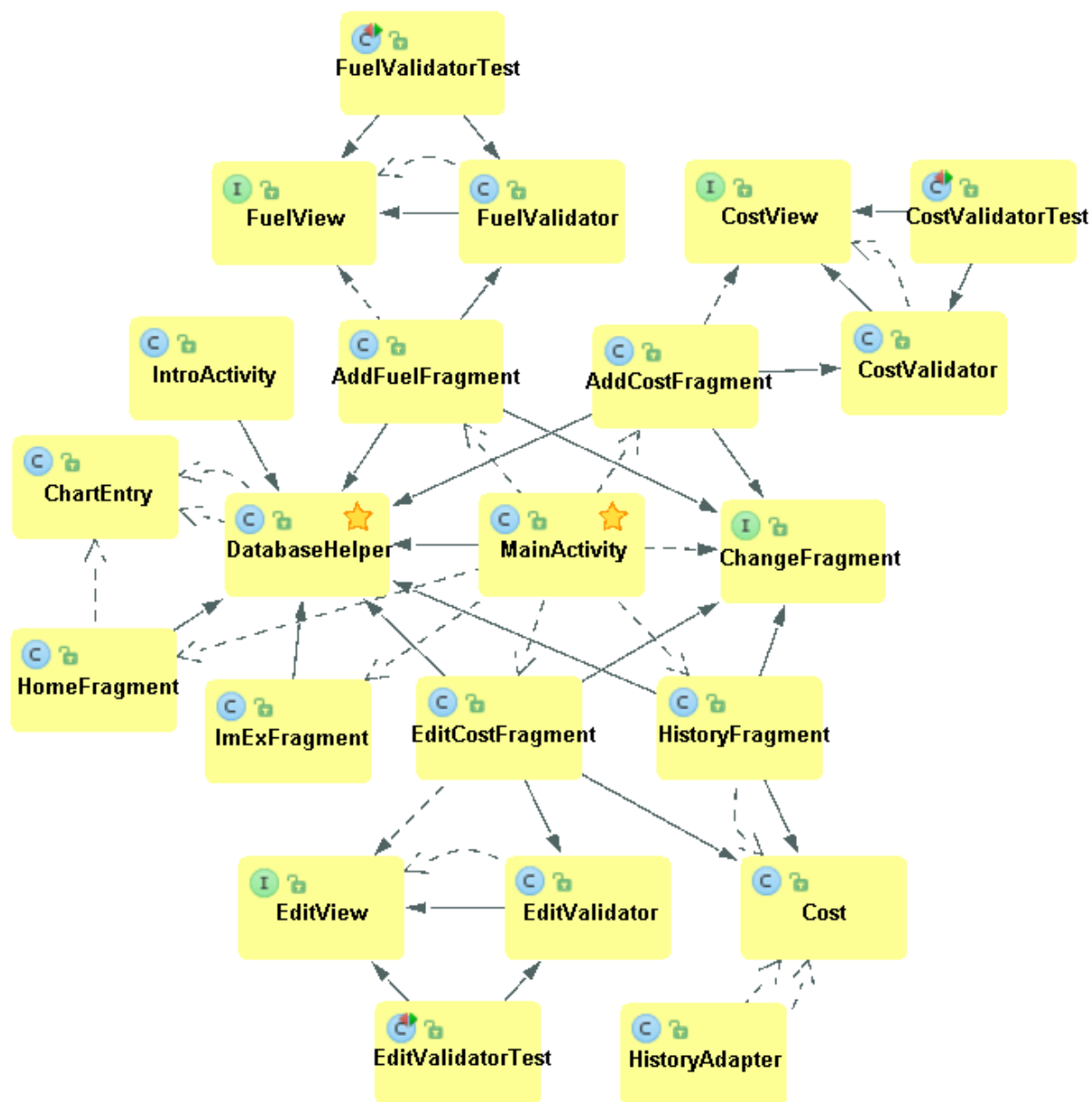
Rysunek 11 HistoryFragment – fragment historii

Rysunek 12 Wybór kategorii filtrowania

5.5. Diagram klas

Poza aktywnościami, fragmentami i widokami zaimplementowałem także kilka klas, które mają na celu reprezentować zbiór danych z bazy i udostępniać je w odpowiedniej formie.

Poniżej zamieszczam diagram klas zawierający także połączenia z aktywnościami i klasami fragmentów.



Rysunek 13 Diagram zależności klas

5.5.1. Interfejsy

Aplikacja zawiera następujące interfejsy:

- a) ChangeFragment – zaimplementowany przez klasę MainActivity służący do przełączania się pomiędzy fragmentami,
- b) CostView, FuelView, EditView – interfejsy zaimplementowane odpowiednio przez AddCostFragment, AddFuelFragment i EditCostFragment, służą do czytelniejszej komunikacji między wymienionymi fragmentami, a odpowiadającymi im klas Validatorów, które zostaną omówione w dalszej części pracy.

5.5.2. Klasy testujące

Klasy o nazwie kończącej się słowem Test służą do testowania klas Validatorów. Zawierają one instancje odpowiadających klas Validator i View. Same testy zostaną omówione w rozdziale 7.

5.5.3. Pozostałe klasy

Aplikacja składa się także z regularnych klas takich jak: Cost, ChartEntry i DatabaseHelper. Poza tym są klasy interfejsu, które zostały wymienione wcześniej w podrozdziale 5.4 oraz klasy aktywności omówione w podrozdziale 5.3.

Na diagramie nie zostały ujęte wszelkie klasy generowane automatyczne, przy których tworzeniu nie miałem udziału.

6. Implementacja

6.1. Opis kluczowych klas aplikacji

W tym rozdziale zostaną przedstawione listingi kodu wraz z omówieniem kluczowych klas programu. Nie zostały zaimplementowane klasy reprezentujące wszystkie tabele ze względu na fakt iż w tabelach user i vehicle istnieje tylko jeden wiersz i szybszym rozwiązaniem jest korzystanie z wbudowanych bibliotek do obsługi SQLite.

6.1.1. Cost

Klasa Cost reprezentuje tabelę cost, a więc posiada pola odpowiadające kolumnom tabeli. Do każdego pola klasy zostały utworzone gettery i setery, a także konstruktor inicjujący wszystkie pola klasy.

```
public Cost(int id, int vehicleId, double expense, String costDate, int mileage, int category, String description, double fuelUnitAmount, double fuelUnitPrice, int fuelFull, int fuelTankNum, String place, String insurer, int insurance, int tankMissed) {
    this.id = id;
    this.vehicleId = vehicleId;
    this.expense = expense;
    this.costDate = costDate;
    this.mileage = mileage;
    this.category = category;
    this.description = description;
    this.fuelUnitAmount = fuelUnitAmount;
    this.fuelUnitPrice = fuelUnitPrice;
    this.fuelFull = fuelFull;
    this.fuelTankNum = fuelTankNum;
    this.place = place;
    this.insurer = insurer;
    this.insurance = insurance;
    this.tankMissed = tankMissed;
}
```

Listing 1 Konstruktor klasy Cost

6.1.2. ChartEntry

Ta klasa przechowuje dane potrzebne do stworzenia wykresu słupkowego widocznego w fragmencie głównym – rysunek 9. Klasa ChartEntry nie zawiera pól odpowiadających pojedynczym elementom kolumny tabeli, ale posiada dwie listy, zawierające dane z całej kolumny o określonych własnościach wymaganych dla utworzenia wykresu. Pierwsza lista to ArrayList<String> zawierająca etykiety do poszczególnych słupków, a druga ArrayList<BarEntry>

zawiera obiekty klasy `BarEntry`, które przechowują indeks słupka i jego wartość, wbudowanej w bibliotekę do tworzenie wykresów, wymienionej w rozdziale o technologiach. Poniżej ciało klasy `ChartEntry`.

```
public class ChartEntry {
    private ArrayList<BarEntry> entries;
    private ArrayList<String> labels;

    public ChartEntry() {
        entries = new ArrayList<>();
        labels = new ArrayList<>();
    }

    public void addBarEntry(BarEntry barEntry) {
        entries.add(barEntry);
    }

    public void addLabel(String label) {
        labels.add(label);
    }

    public ArrayList<BarEntry> getEntries() {
        return entries;
    }

    public ArrayList<String> getLabels() {
        return labels;
    }
}
```

Listing 2 Klasa `ChartEntry`

6.1.3. DatabaseHelper

Klasa rozszerzająca klasę `SQLiteOpenHelper`. Jej pola zawierając etykiety wszystkich kolumn z trzech tabel bazy danych. Kolejne metody mają za zadanie wchodzić w interakcje z bazą `SQLite` i czytywać, umieszczać lub usuwać z niej dane. Klasa ta ma swoją instancję w niema każdej z pozostałych klas aplikacji, ponieważ wielokrotnie odwołują się w metodach do bazy danych. Fragmenty klasy zostaną omówione rozdziale dotyczącym komunikacji z bazą danych.

6.1.4. Klasy i interfejsy do testów

Do testów zostały stworzone klasy weryfikujące poprawność pobieranych danych kosztów przed ich umieszczeniem w bazie danych. Ich działanie zostanie omówione bardziej szczegółowo w rozdziale dotyczącym testów.

6.2. Algorytmy obliczeniowe

6.2.1. Obliczanie całkowitego kosztu kilometra

Obliczanie całkowitego kosztu przejechanego kilometra sprowadza się do zsumowania wszystkich kosztów, jakie zostały dodane od początku istnienia bazy oraz podzielenie tej sumy przez przebyty w tym czasie dystans obliczony na podstawie różnicy maksymalnej i minimalnej wartości przebiegu.

6.2.2. Średnie spalanie

Obliczanie dokładnego średniego spalania jest możliwe tylko wtedy, gdy użytkownik tankuje samochód do pełna przynajmniej raz na jakiś czas. Można także szacować średnie spalanie posiadając pojemność baku oraz przebyte kilometry, ale ja przyjąłem formę najbardziej precyzyjną nie zakładającą bez szacowania, a więc opierającą się na pełnych zatankowaniach samochodu. Aby obliczyć średnie spalanie należy obliczać dystanse pomiędzy pełnymi zatankowaniami i sumować ilość zatankowanego paliwa. Następnie sumę paliwa dzielimy przez dystans i mnożymy przez 100, aby uzyskać jednostkę spalania np. l/100km.

6.2.3. Obliczanie kosztu kilometra z ostatnich trzydziestu dni

Obliczanie całkowitego kosztu przejechanego kilometra sprowadza się do zsumowania wszystkich kosztów, jakie zostały dodane w ciągu ostatniego miesiąca oraz podzielenie tej sumy przez przebyty w tym czasie dystans obliczony na podstawie różnicy maksymalnej i minimalnej wartości przebiegu.

6.3. Komunikacja z bazą danych

W niniejszym podrozdziale zostanie omówiona komunikacja z bazą danych oparta o klasę DatabaseHelper dziedziczącą po klasie SQLiteOpenHelper.

6.3.1. Utworzenie bazy

Do utworzenia bazy danych została wykorzystana metoda onCreate() w klasie DatabaseHelper. Opiera się ona w trzykrotnym wywołaniu funkcji wykonującej zapytanie CREATE. W poszczególnych zapytaniach są tworzone kolejne tabele.

6.3.2. Nawiązanie połączenia z bazą danych

Połączenie z bazą nawiązuje się poprzez wywołanie konstruktora DatabaseHelper. Użytkany w ten sposób obiekt może korzystać z metod, z których każda zawiera linię tworzącą obiekt klasy SQLiteDatabase.

```
SQLiteDatabase db = this.getWritableDatabase();
```

Listing 3 Uzyskanie połączenia z bazą danych

6.3.3. Pobranie kosztów z ostatnich 30 dni

Pobranie kosztów odbywa się podanie zakresu dat korzystając z wyrażenia:

```
... WHERE cost_date BETWEEN date('now', '-1 month') AND date('now') ORDER BY  
cost_date
```

Listing 4 Wyznaczenie zakresu dat

Dużą rolę odgrywa tu funkcja SQLite date(), umożliwiającą uzyskiwanie dat o wymaganym w bazie formacie bez konieczności korzystania z formaterów łańcuchowych.

6.3.4. Dodanie nowego kosztu

Dodanie nowego kosztu jest oparte o wbudowaną funkcję insert(). Zadaniem programisty jest jedynie umieścić w obiekcie klasy ContentValues pary (nazwa_kolumny, wartość_wstawiana).

```
public boolean insertCostData(int vehicle_id, double expense,  
                               String cost_date, int mileage, int category,  
                               String description, double fuel_unit_amount,  
                               double fuel_unit_price, int fuel_full,  
                               int fuel_tank_num, String place,  
                               String insurer, int insurance,  
                               int tank_missed) {  
    SQLiteDatabase db = this.getWritableDatabase();  
    ContentValues contentValues = new ContentValues();  
    contentValues.put(COST_COL_2, vehicle_id);  
    contentValues.put(COST_COL_3, expense);  
    contentValues.put(COST_COL_4, cost_date);  
    contentValues.put(COST_COL_5, mileage);  
    contentValues.put(COST_COL_6, category);  
    contentValues.put(COST_COL_7, description);  
    contentValues.put(COST_COL_8, fuel_unit_amount);  
    contentValues.put(COST_COL_9, fuel_unit_price);  
    contentValues.put(COST_COL_10, fuel_full);  
    contentValues.put(COST_COL_11, fuel_tank_num);  
    contentValues.put(COST_COL_12, place);  
    contentValues.put(COST_COL_13, insurer);  
    contentValues.put(COST_COL_14, insurance);  
    contentValues.put(COST_COL_15, tank_missed);  
    return db.insert(COST_TABLE, null, contentValues) != -1;  
}
```

Listing 5 Dodawanie nowego kosztu

6.3.5. Aktualizacja wybranego kosztu

Aktualizacja kosztu odbywa się przy pomocy funkcji `update()`, która również jest dostępna w klasie `SQLiteDatabase`. Postępowanie jest analogiczne jak przy funkcji `insert()` z tą tylko różnicą, że w tym przypadku należy podać warunek identyfikujący wiersz w bazie danych.

6.3.6. Usunięcie wybranego kosztu

Również w przypadku usuwania jest dostępna wbudowana analogiczna funkcja `delete()`. Należy jedynie wskazać warunek identyfikujący wiersz w bazie danych.

```
public boolean deleteCostById(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    return db.delete(COST_TABLE, "id = " + id, null) != 0;
}
```

Listing 6 Usunięcie kosztu za pomocą id

6.3.7. Import i eksport bazy danych

Zarówno import jak i eksport polegają jedynie na skopiowaniu pliku bazy danych korzystając z uprawnień dostępu do pamięci zastrzeżonej dla użytkownika. Jest to bardzo prosta metoda tworzenia i odzyskiwania kopii zapasowej bazy danych, ale niesie także ze sobą pewne niebezpieczeństwa oraz jest mało wygodna.

```
public void importDatabase(Context context) throws IOException {
    File direct = new File(Environment.getExternalStorageDirectory() +
"/CarCost");
    String cDBPath = "//data//com.kuba.carcost//databases//carcost.db";
    String bDBPath = "/CarCost/carcost.db";
    if(!direct.exists()) {
        if(direct.mkdir()) {}
    }
    File sd = Environment.getExternalStorageDirectory();
    File data = Environment.getDataDirectory();
    try {
        if (sd.canWrite()) {
            File backupDB = new File(data, cDBPath);
            File currentDB = new File(sd, bDBPath);
            FileChannel src = new FileInputStream(currentDB).getChannel();
            FileChannel dst = new FileOutputStream(backupDB).getChannel();
            dst.transferFrom(src, 0, src.size());
            src.close();
            dst.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 7 Import pliku bazy danych

7. Testy aplikacji

7.1. Testy jednostkowe z wykorzystaniem Mockito

Testy jednostkowe zostały oparte na mockach, czyli imitacjach działającego obiektu. Zostały utworzone klasy, które posiadały metodę `validate()` mającą za zadanie pobierać dane z fragmentu, które zostały wpisane przez użytkownika w pola tekstowe, a następnie sprawdzana była ich poprawność:

- a) `FuelValidator` – klasa sprawdzająca dane tankowania,
- b) `CostValidator` – klasa sprawdzająca dane pozostałych kosztów,
- c) `EditValidator` – klasa sprawdzająca dane kosztu po edycji.

7.1.1. Opis wykonywanych testów

Pierwszym elementem było utworzenie w klasie fragmentu odpowiadającej instancji klasy sprawdzającej poprawność danych. Następnie przy akcji przycisku zatwierdzającego wprowadzone dane, była wywoływana na utworzonej instancji metoda `validate()`, która zwracała wartość logiczną odpowiadającą wynikowi wszystkich testów lub w przypadku niepowodzenia ustawiała komunikat błędu na polu tekstowym, z którego została pobrana dana wartość.

Testy przy wykorzystaniu Mockito testowały metodę `validate()` poprzez sprawdzanie jak reagują kolejne instrukcje warunkowe na podawane dane. Testy były uruchamiane z poziomu środowiska programistycznego. Mockami były instancje klasy `Validator` oraz interfejsu umożliwiającego komunikację między fragmentem, a klasą `Validator`.

7.1.2. Wnioski z testów

Dzięki przeprowadzaniu testów miałem możliwość naprawienia błędów w kodzie, które były przez testy wykrywane. Efektem jest zabezpieczenie kodu przed wprowadzeniem danych, które mogłyby spowodować błąd wykonania zapytania SQL lub być niemożliwe do późniejszej weryfikacji.

Testy programowalne są szybsze w wyszukiwaniu błędów niż ręczne sprawdzanie kombinacji i wygodniejsze w interpretacji.

7.2. Testy manualne na emulatorze

Do testów został wykorzystany emulator wbudowany w środowisko AndroidStudio. Emulowanym urządzeniem był Nexus 5X z systemem Android 7.1.1.

7.2.1. Opis wykonywanych testów

Testy na emulatorze polegały na wykonywaniu czynnościach związanych z wprowadzoną kolejną funkcjonalnością, które miały na celu działać wbrew zamyśle aplikacji np. podczas dodawania kosztów, sprawdzałem błędy, które były wyrzucane, gdy zostawiłem któreś pole puste lub wprowadzałem niepoprawną wartość. Podjąłem również próbę weryfikacji danych dodawanych do bazy emulatora. Nie powiodło się także odczytanie bazy danych utworzonej przez aplikację w emulatorze zarówno korzystając z Android Managera, a także poprzez ADB.

7.2.2. Wnioski z testów

Emulator działał szybciej i płynniej niż mój własny smartfon, ale poruszanie się po nim za pomocą kursora myszy znacznie spowalniało testowanie. Ze względu także na brak dostępu do bazy danych po pewnym czasie porzuciłem testowanie na emulatorze ze względu na jego niepraktyczność i brak możliwości weryfikacji jak dana wartość prezentuje się w bazie.

7.3. Testy manualne na urządzeniu mobilnym

Do testów manualnych został wykorzystany Samsung S2 (GT-i9100) z systemem CyanogenMod 13.0 oparty na systemie Android 6.0.1.

7.3.1. Opis wykonywanych testów

Na smartfonie była testowana każda wprowadzona funkcja, a także zmiany w wyglądzie. Korzystanie z aplikacji używając dotyku było wygodniejsze i szybsze mimo mniejszej mocy obliczeniowej oraz dostępnej pamięci operacyjnej. Była także możliwość dostępu do bazy danych, więc mogłem weryfikować wartości wstawiane w poszczególnych wierszach. Można wyszczególnić następujące testy:

- a) wprowadzanie niepoprawnych danych do pól tekstowych lub zostawianie pustych pól przy dodawaniu lub edycji kosztów,
- b) przerywanie działania aplikacji,
- c) wprowadzanie nieskorelowanych ze sobą danych,

- d) eksportowanie bazy danych i weryfikacja jej zawartości na przeglądarce SQLite na komputerze stacjonarnym,
- e) modyfikacja bazy na komputerze stacjonarnym oraz importowanie zmienionego pliku do aplikacji,
- f) sprawdzanie wyświetlania layoutu XML oraz porównanie z emulatorem.

7.3.2. Wnioski z testów

Ze względu na niższą rozdzielczość niż Nexus 5X miałem porównanie jak wygląda zaprojektowany interfejs na urządzeniach o różnej rozdzielczości. To pozwoliło zoptymalizować interfejs pod kątem uniwersalnego wyglądu na większości urządzeń. Weryfikacja danych w bazie pokazała, że wyniki dodawane poprzez fragmenty dodawania kosztu lub tankowania oraz te aktualizowane są poprawne. Także w poprawny sposób koszty były usuwane z bazy. Import i eksport również działał poprawnie, według przyjętej metody. Eksportowane pliki można było odczytać w przeglądarce SQLite na PC i przeglądać zawartość bazy, a także po modyfikacjach. Zmiany w zaimportowanych plikach były widoczne w aplikacji.

8. Podsumowanie

8.1. Wnioski

Celem pracy było napisanie aplikacji mobilnej oraz nauczenie się technologii programowania na system Android. Te cele zostały spełnione, bo powstała aplikacja, przy tworzeniu której nauczyłem się w jaki sposób są konstruowane aplikacje mobilne oraz jakie możliwości oferują. Wymagało to ode mnie dużo czasu i zaangażowania, ale efekt jest zadowalający.

Założenia wstępne odnośnie funkcjonalności aplikacji zostały okrojone i musiałem zrezygnować z zaawansowanych statystyk oraz wielu pojazdów. Nie zostały dodane również krótkie poradniki pomagające zrozumieć jak działa strona, ale za to interfejs został uproszczony, więc jest intuicyjny i czytelny.

8.2. Możliwości rozwoju

Możliwości rozwoju aplikacji są szerokie i w pierwszej kolejności można dodać funkcjonalności, które ze względu na brak czasu zostały przeze mnie porzucone, czyli posiadanie w aplikacji danych wielu pojazdów, poradniki po widokach lub konfigurowalne statystyki. Można także wprowadzić różne wersje językowe i prognozować zasięg pojazdu lub zawartość paliwa w baku na podstawie dystansu pokonanego od ostatniego tankowania.

9. Literatura i źródła

- [1] R. C. Martin. Czysty kod: Podręcznik Dobrego Programisty. Helion, Gliwice, 2014.
- [2] Google Inc., Android Developers. <https://developer.android.com> - dostęp 28.01.2017.
- [3] Damian Chodorek. Kurs Android. <http://damianchodorek.com/category/kursy/android> – dostęp 18.01.2017.
- [4] Marco Vitas. Android Testing Tutorial: Unit Testing like a True Green Droid. <https://www.toptal.com/android/testing-like-a-true-green-droid> – dostęp 26.01.2017.
- [5] Forum programistyczne. <http://stackoverflow.com> – dostęp 28.01.2017.
- [6] Michał Gellert. Kurs programowania na platformę Android. <https://www.youtube.com/playlist?list=PLTs20Q-BTEMNaj4UgOcQfSBsMvIH8bjuX> – dostęp 21.12.2016.

10. Załącznik – instrukcja

10.1. Zawartość płyty CD

W załączniku znajduje się płyta CD wraz z wersją instalacyjną aplikacji w formacie *.apk. Aby zainstalować aplikację na urządzeniu, należy skopiować plik instalacyjny do pamięci smartfonu, a następnie uruchomić go poprzez kliknięcie na niego i ewentualne wybranie aplikacji, która przeprowadzi proces instalacji (zależy od urządzenia) oraz zatwierdzenie wymagań aplikacji odnośnie uprawnień aplikacji (od wersji Androida 6.* i wyższej zatwierdzenie uprawnień występuje za pierwszym razem, gdy zostanie zgłoszona potrzeba skorzystania przez aplikację z danego elementu lub w momencie zaprojektowanym w tym celu przez programistę). Po upływie kilku do kilkunastu sekund instalacja będzie zakończona, więc będzie możliwość korzystania z aplikacji.

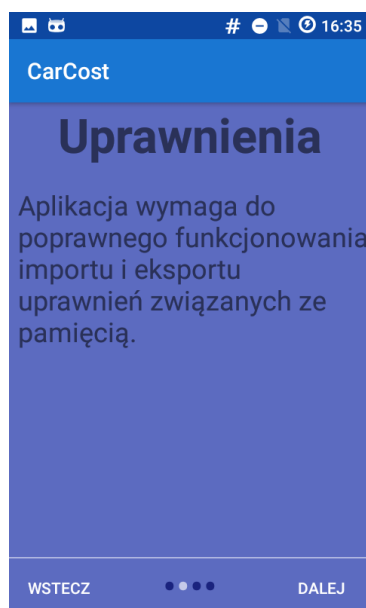
10.2. Instrukcja korzystania z aplikacji

10.2.1. Intro

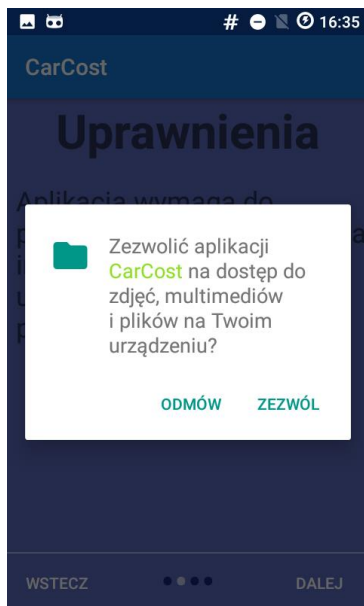
Przy pierwszym uruchomieniu aplikacji pojawi się intro z czterema widokami. Posługując się przyciskami Dalej, Wstecz i Zakończ przemieszczamy się pomiędzy ekranami.



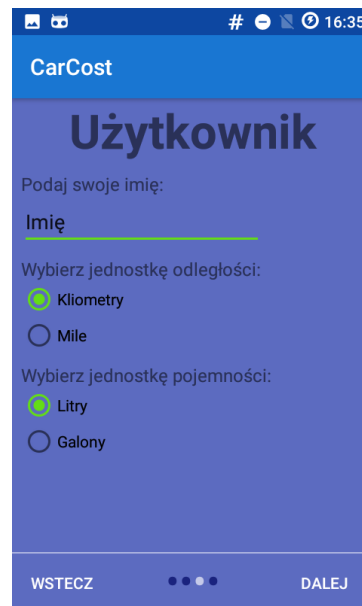
Rysunek 14 Intro - ekran pierwszy



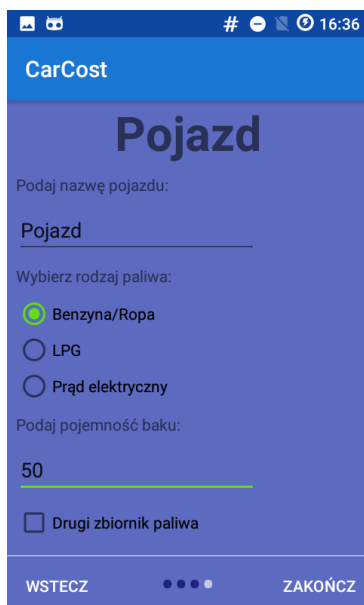
Rysunek 15 Intro - Uprawnienia



Rysunek 16 Intro - Zapytanie o uprawnienia



Rysunek 17 Intro - Wprowadzenie danych użytkownika



Rysunek 18 Intro - Wprowadzenie danych pojazdu

Widoczne powyżej rysunki przedstawiają kolejne widoki intra. Przy rysunku 16 należy zezwolić na dostęp, a w przypadku rysunku 17 i 18 wprowadzić dane i zaznaczyć wybrane pola. Przycisk Zakończ zapisze dane do bazy i zamknie intro przenosząc do ekranu głównego – rysunek 19.

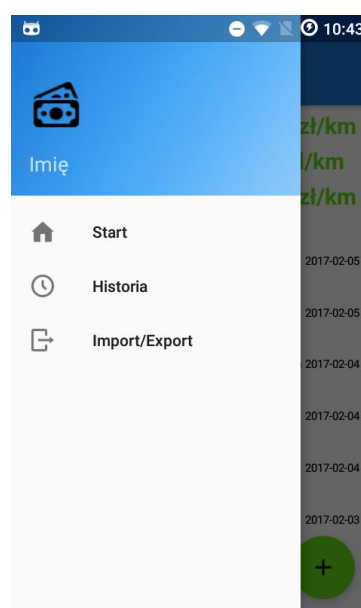
10.2.2. Ekran główny

Jest to pierwszy ekran widoczny przy regularnym uruchomieniu aplikacji. W lewym górnym rogu widoczne są trzy paski, których kliknięcie rozwija menu boczne – rysunek 20. Menu można także rozwinąć poprzez pociągnięcie palcem od lewej krawędzi. W menu dostępne są trzy zakładki. „Start” przenosi do ekranu głównego, „Historia” przenosi do widoku listy kosztów, a „Import/Eksport” otwiera widok z przyciskami do importu i eksportu bazy danych.

W prawym dolnym rogu znajduje się zielony przycisk pływający. Krótkie przyciśnięcie tego przycisku uruchamia dodawanie tankowania. Długie przyciśnięcie uruchamia dodawanie kosztów z pozostałych kategorii.



Rysunek 19 Ekran główny



Rysunek 20 Menu boczne

10.2.3. Dodawanie tankowania i kosztu

Widoki dodawania tankowania i kosztu przedstawiają odpowiednio rysunki 21 i 22. W widoku dodawania tankowania należy wypełnić wszystkie pola, wybrać datę oraz zaznaczyć odpowiednie pola. W przypadku dodawania pozostałych kosztów należy dodatkowo wybrać kategorię i wypełnić pola charakterystyczne dla danej kategorii. Pola o nazwach: opis, miejsce, ubezpieczyciel nie są obowiązkowe. Jeśli nie jest widoczny przycisk Dodaj, należy przewinąć ekran, bo przycisk znajduje się niżej, poza widoczną częścią. Jeśli wprowadzone dane będą niepoprawne, wyświetli się błąd.

Rysunek 21 Dodaj tankowanie

Rysunek 22 Dodaj koszt

10.2.4. Historia

W widoku historii – rysunek 23 – na górze widoku znajdują się pola do wyboru zakresu dat oraz przycisk wyboru kategorii wywołujący okno dialogowe z listą checkboxów kategorii – rysunek 24. Wprowadzone filtry należy zatwierdzić przyciskiem Szukaj. Poniżej znajduje się lista kosztów domyślnie ze wszystkich kategorii oraz dodanych w ciągu ostatnich 30 dni. Klikając na wybranym koszcie przechodzimy do ekranu edycji.

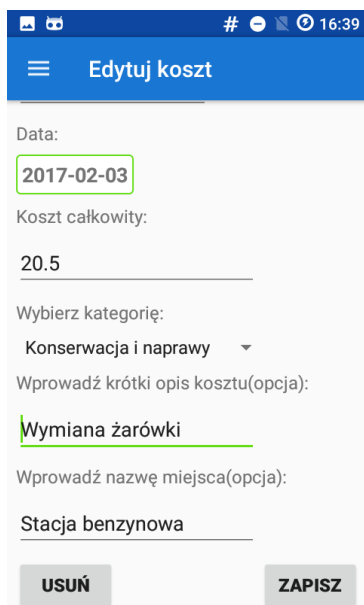
Historia		
Od: 2017-01-03 KATEGORIE		
Do: 2017-02-03 SZUKAJ		
Tankowanie		50.0
2017-02-03		50
Konservacja i naprawy		20.0
2017-02-03		300

Rysunek 23 Widok historii

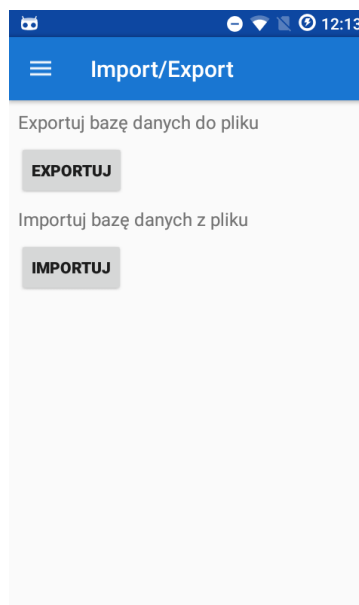
Rysunek 24 Okno dialogowe wyboru kategorii

10.2.5. Edycja kosztu

Widok edycji – rysunek 25 – umożliwia zmianę parametrów kosztu wybranego w historii. Struktura pól w tym widoku jest analogiczna do widoku dodawania kosztu lub tankowania. Jediną różnicą jest przycisk Usun, który usuwa koszt z bazy danych.



Rysunek 25 Edytuj koszt



Rysunek 26 Import i Eksport

10.2.6. Import i Eksport

Najprostszym widokiem jest Import/Eksport – rysunek 26. Składa się z dwóch przycisków, z których jeden z nich Importuje bazę danych, a drugi ją eksportuje.