# **Wstęp biznesowy**

W przeciągu ostatnich lat segment urządzeń mobilnych przeżył rewolucję, zarówno pod kątem technologicznym jak i użytkowym. Urządzenia mobilne w dzisiejszych czasach stały się nieodzownym narzędziem dla człowieka. Niemal każdy wykorzystuje je w życiu codziennym w praktycznie każdej dziedzinie życia, są wykorzystywane również w motoryzacji.

Zadaniem aplikacji jest pomoc w obliczaniu kosztów związanych z eksploatacją samochodu a także ukazywanie terminów ważności przeglądów i ubezpieczeń. Jej głównym zadaniem jest możliwość rejestrowania kosztów w celu późniejszej możliwości sprawdzenia statystyk z tym związanych i wyciągania wniosków dotyczących rentowności użytkowania pojazdu.

Duży nacisk położono na intuicyjność oraz łatwość obsługi programu. Dzięki czemu osiągnięto przystępność w zrozumieniu przez użytkownika, co spowoduje, że będzie chętnie z niej korzystał oraz najważniejsza czynność czyli rejestracja kosztów będzie przebiegała w sposób jak najszybszy i najbardziej efektywny.

Ponadto program przechowuje dane w chmurze, dzięki czemu użytkownik nie jest związany z konkretnym urządzeniem mobilnym, a posiada dostęp do swoich danych z dowolnego narzędzia opartego o system Android.

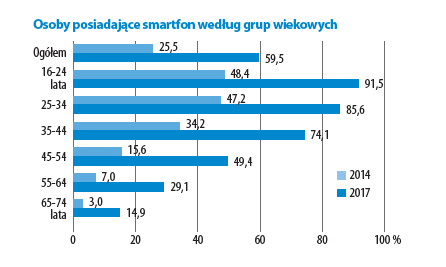
Rejestracja kosztów w telefonie ułatwia rzeczywiste określenie kosztów eksploatacji samochodu, wliczając w to zarówno koszty paliwa jak i koszty serwisu, co w może wpływać na sposoby wykorzystania pojazdu jak i perspektywie czasu może również wpłynąć na decyzję o zmianie bądź nie samochodu ze względu na koszty jego utrzymania.

# **Profil użytkownika**

Aplikacja kierowana jest do użytkowników pojazdów mechanicznych z napędem spalinowym, których interesują dane dotyczące kosztów jakie ponoszą w związku z ich użytkowaniem oraz którzy równocześnie korzystają z urządzeń mobilnych działających na systemie Android.

Ze względu na łatwą dostępność do sieci Internet, przy pomocy której program będzie dystrybuowany oraz przez to, że aplikacja w pierwszej wersji posiada interfejs wyłącznie w języku polski użytkownikami będą osoby, posługujące się językiem polskim, bez względu na aktualne miejsce zamieszkania.

Program ze względu na platformę mobilną na której działa kierowany jest głównie do grup wiekowych, które mają największy dostęp do urządzeń tego typu, np. smartfon.



Według badań przeprowadzonych przez Główny Urząd Statystyczny są to 3 grupy wiekowe:

* 16-24 lata
* 25-34 lata
* 35-44 lata.

Należy wziąć również pod uwagę, iż osoby poniżej 18 roku życia będą stanowiły niewielki odsetek użytkowników ze względu na ograniczenia dotyczące uprawnień korzystania z pojazdów mechanicznych oraz należy wziąć pod uwagę grupę wiekową 45-54 lata w której ponad 49% posiada urządzenia mobilne oraz porównując ten udział z latami poprzednimi posiada tendencję wzrostową.

# **Wymagania techniczne**

Aplikacja stworzona została na platformy mobilne oparte o system android i na takich działa. Nie ma ograniczeń związanych z rodzajem urządzenia mobilnego. Oprogramowanie działa zarówno na telefonach jak i tabletach, jak i również innych urządzeniach opartych o system android, pod warunkiem spełnienia poniższych wymagań minimalnych.

Wymagania minimalne:

## Hardware

* Urządzenie mobilne – urządzenie o ekranie min. 5.0 cal

## Software

* System Android – API level min. 15 czyli odpowiadająca temu wersja Android min. Ice Cream Sandwich lub wyższe

## Network

* Dostęp do sieci Internet

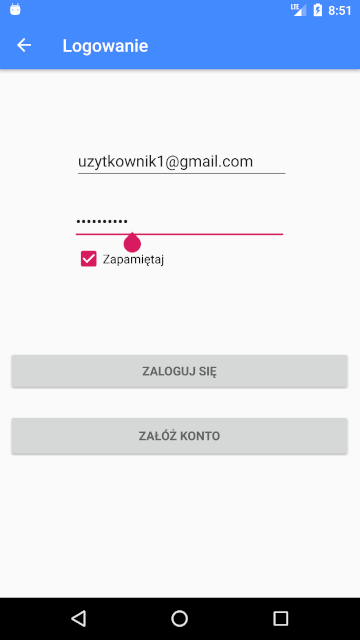
# **Opis funkcjonalności**

1. Wprowadzenie do aplikacji



* Po pierwszym uruchomieniu aplikacji użytkownik widzi ekran powitalny, z logiem aplikacji, które pozwala wprowadzić użytkownika do aplikacji. Użytkownik może pominąć ekran poprzez kliknięcie w przycisk “Rozpocznij”.
* Użytkownik jest pytany o zezwolenie na dostęp do Internetu poprzez aplikację. W przypadku wyrażenia zgody, wyświetlane są obiekty.

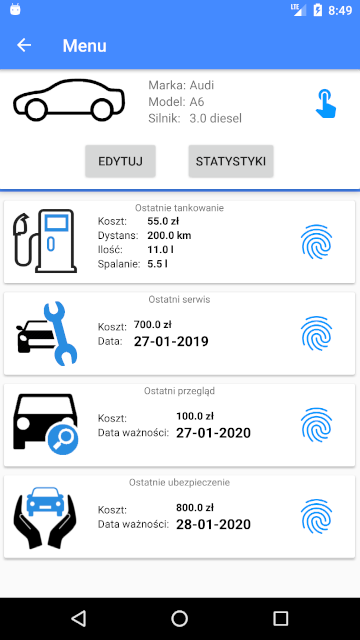
## 2. Logowanie i rejestracja



* Posiadanie konta użytkownika jest niezbędne do korzystania z prgoramu
* Logowanie do aplikacji oparte jest o adres email
* Dane niezbędne do rejestracji
  + Adres e-mail
  + Hasło do poczty e-mail
* Użytkownik posiada możliwość zapamiętania wprowadzanego hasła w celu ułatwienia ponownego logowania

## 3. Menu

Ekran Menu podzielony jest na sekcje tematyczne, dotyczące konkretnych rodzajów obiektów biznesowych aplikacji. Są to:



### Informacje o pojeździe.

* + Marka
  + Model
  + Silnik

Sekcja posiada rozwijane przyciski:

* Edytuj – przycisk otwierający okno z parametrami pojazdu, w którym można edytować aktualny profil
* Statystki – przycisk dzięki, któremu uruchomione zostanie odrębne okno – Statystki. Okno to jest nieedytowalne i zawiera aktualnie, na bieżąco wyliczane statystyki dotyczące kosztów użytkowania oraz utrzymania pojazdu oraz podsumowanie o tym jakie koszty użytkownik poniósł do chwili obecnej.

### Informacje o ostatnich zarejestrowanych kosztach:

* + Ostatnie tankowanie
    - Koszt
    - Dystans
    - Ilość
    - Spalanie
  + Ostatni serwis
    - Koszt
    - Data
  + Ostatni przegląd
    - Koszt
    - Data ważności
  + Ostatnie ubezpieczenie
    - Koszt
    - Data ważności

Sekcje posiadają rozwijane przyciski:

* Dodaj – przycisk pozwalający na zarejestrowanie nowego kosztu
* Historia – przycisk pozwalający na przejście do listy z historią obecnie zarejestrowanych kosztów

## 4. Statystyki



Okno Statystyki eksploatacji zawiera wyliczone informacje na podstawie zarejestrowanych wcześniej kosztów. Przedstawiane wyniki nigdzie nie są zapisywane, ich wartości pobierane i przeliczane są na bieżąco z chmury.

Okno podzielone jest na sekcje:

#### Całość

Sekcja prezentuje podsumowanie wszystkich zarejestrowanych kosztów:

* 1. Przebieg początkowy – przebieg „startowy” traktowany jest jako przebieg od którego liczone są wszelkie statystyki, pierwsze tankowanie nie jest z tego względu uwzględniane w algorytmach
  2. Przebieg aktualny - odczytanie informacji z ostatniego tankowania z bazy danych
  3. Całkowity koszt eksploatacji – zsumowane wszystkie koszty zarejestrowane w bazie danych, bez względu na rodzaj kosztu
  4. Całkowity przejechany dystans – różnica między dystansem początkowym a obecny, odczyty oparte na przebiegach rejestrowanych przy dodawanych tankowniach paliwa
  5. Całkowity koszt kilometra – wynik pokazuje koszt kilometra z uwzględnieniem wszystkich wartości, czyli koszt paliwa, serwisowania, przeglądów oraz ubezpieczenia

#### Paliwo

* 1. Całkowita ilość – odczytana całkowita ilość tankowań zarejestrowana w bazie danych
  2. Całkowity koszt – odczytany koszt wszystkich tankowań zarejestrowanych w bazie danych i zsumowana ta wartość
  3. Średnie spalanie – obliczone spalanie z wszystkich zarejstrowań w systemie, z pominięciem pierwszego tankowania, które jest traktowane jako startowe, rozruchowe

#### Serwisy

* 1. Całkowity koszt - odczytane i zsumowane wartości wszystkich zarejestrowanych serwisów w bazie danych
  2. Liczba serwisów - odczytana i zsumowana ilość wszystkich zarejestrowanych serwisów w bazie danych

#### Przeglądy

* 1. Całkowity koszt - odczytane i zsumowane wartości wszystkich zarejestrowanych przeglądów w bazie danych
  2. Liczba przeglądów - odczytana i zsumowana ilość wszystkich zarejestrowanych przeglądów w bazie danych

#### Ubezpieczenia

* 1. Całkowity koszt - odczytane i zsumowane wartości wszystkich zarejestrowanych ubezpieczeń w bazie danych
  2. Liczba serwisów - odczytana i zsumowana ilość wszystkich zarejestrowanych ubezpieczeń w bazie danych

# **5. Opis implementacji**

### Kompilator

Aplikacja została stworzona na platformę android w kompilatorze Android studio i zawiera elementy charakterystyczne dla tego środowiska, np. obiekty Activity – czyli aktywności, poszczególne okna i widoki aplikacji.

### Język

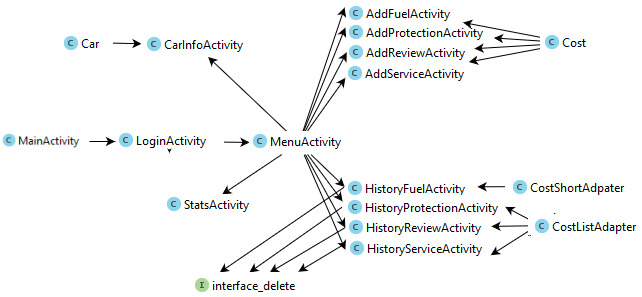
Język w jakim aplikacja jest napisana to Java.

W aplikacji wykorzystano bibliotekę klas i metod Firebase – ze względu na łączność z chmurą Firebase, gdzie tworzona jest baza danych zawierająca zapisy aplikacji jak i wykorzystywany jest moduł FirebaseAuth obsługujący autentykację użytkownika oraz tworzenie i przechowywanie jego profilu.

### Opis obiektów aplikacji:

1. Widoki ogólne:
   1. MainActivity - aktywność obsługująca pierwsze okno programu - okno powitalne, informujące hasłowo do czego służy uruchamiane oprogramowanie
   2. LoginActivity – aktywność obsługująca logowanie i rejestracje
   3. MenuActivity – aktywność wyświetlająca informacje o samochodzie oraz informacje o ostatnich zarejestrowanych kosztach
   4. StatsActivity – aktywność wyświetlająca wyliczone informacje na podstawie zarejestrowanych wcześniej kosztów. Przedstawiane wyniki nigdzie nie są zapisywane, ich wartości pobierane i przeliczane są na bieżąco z chmury
2. Widoki dodawania nowych kosztów – widoki wyświetlające okna pozwalające na dodanie nowych kosztów:
   1. AddFuelActivity
   2. AddProtexctionActivity
   3. AddReviewActivity
   4. AddServiceActivity
3. Widoki wyświetlające listy zarejestrowanych kosztów:
   1. HistoryFuelActivity
   2. HistoryServiceActivity
   3. HistoryReviewActivity
   4. HistoryProtectionActivity
4. Klasy Java:
   1. Car - Klasa zawierająca informacje na temat pojazdu którego koszty użytkownik będzie rejestrował. Informacje o pojeździe są zapisane w bazie i są przechowywane w chmurze a w aplikacji widoczne są w Menu programu
   2. Cost - Klasa zawierająca informacje na temat kosztu zarejestrowanego w programie. Informacje te są przechowywane w chmurze w bazie danych a w aplikacji widoczne są w Menu programu.
5. Adaptery:
   1. CostList - adpater wykorzystywany przy wyświetlaniu długich list pobranych danych z bazy danych, takich jak na przykład listę wszystkich tankowań
   2. CostShortAdapter - adpater wykorzystywany przy wyświetlaniu krótkich list pobranych danych z bazy danych, takich jak na przykład informacje o serwisach, przeglądach i ubezpieczeniach
6. Interfejsy:
   1. Interface\_delete - Interfejs implementujący metody niezbędne do usuwania dodanych kosztów z bazy danych

### Opis działania klas i metod



Uproszczony diagram klas aplikacji

Pierwszą aktywnością jaką można zauważyć na diagramie jest aktywność MainAcivity. Jej zadaniem jest obsługa okna powitalnego programu i poinformowanie użytkownika z jaką aplikacją ma do czynienia. Okno to udostępnia przycisk pozwalający uruchomić kolejną aktywność, za którą odpowiada klasa LoginActivity. Jak sama nazwa sugeruje kod tej klasy zapewnia funkcjonalność logowania i rejestracji użytkownika. Obsługa autentykacji została oparta o platformę Firebase, która udostępnia moduł Authentication. Aby móc wykorzystać jego możliwość należało najpierw połączyć projekt aplikacji z platformą. W tym celu należało stworzyć konto w chmurze Firebase i stworzyć nowy projekt a dalej w samym Android Studio zmodyfikować plik google-service.json oraz pliki build.gradle na poziomie projektu oraz na poziomie aplikacji według instrukcji jaką dostarcza platforma. Poza tym należy pamiętać, że aplikacja komunikuje się z bazą poprzez sieć Internet i aby było to możliwe należy umożliwić korzystanie z niej i pozwolenie na sprawdzanie jej stanu. Konfigurację praw wykonuje się w pliku AndroidManifest.xml.

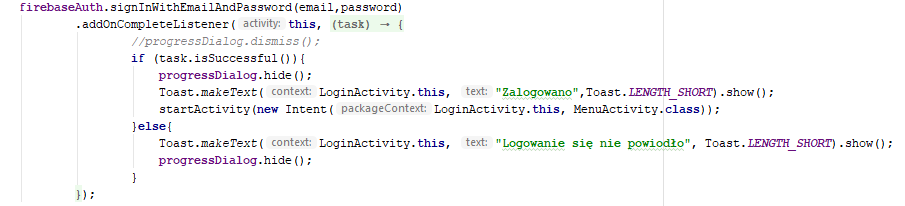


Rysunek 1. Dodanie pozwolenia w AndoidManifest.xml



Rysunek 2. Fragment pliku AndroidManifest.xml

Plik ten w projektach android odpowiada za wewnętrzne dane konfiguracyjne i definicje jej składników. Wczytanie manifestu aplikacji jest pierwszą czynnością jaką wykonuje system operacyjny gdy użytkownik uruchamia aplikację. Plik ten wywodzi się wprost z języka Java, skąd został zaczerpnięty do Androida i wskazuje, która klasa jest aktywnością wejścia do aplikacji.[[1]](#footnote-1) Po zmapowaniu projektu aplikacji z projektem w platformie Firebase można przystąpić do implementacji metod autentykacji dostarczanych przez bibliotekę Firebase.



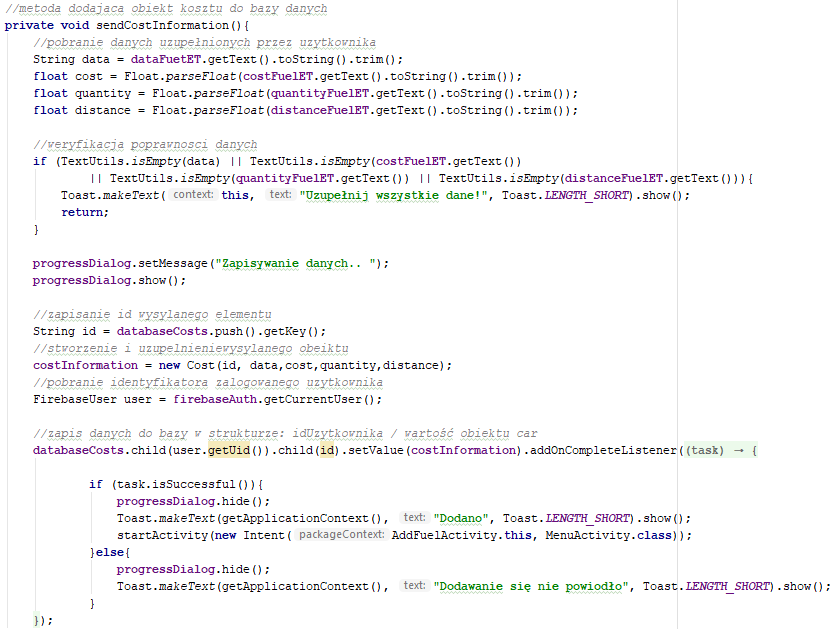
Rysunek 3. Przykład implementacji metody logującej użytkownika do chmury Firebase

Należy tutaj też wspomnieć że została wykorzystana również klasa SharedPreferences. Klasa ta pozwala na zapisanie w pamięci telefonu informacji po wyjściu z aplikacji. W omawianej aplikacji umożliwia to zapamiętanie danych logowania użytkownika, jeżeli taką opcję włączy aktywując checboxa „Zapamiętaj”. Po spełnieniu wszystkich warunków poprawności wprowadzonych danych i połączenia z siecią Internet zostaniemy zalogowani i uruchomione zostanie okno obsługiwane przez klasę MenuActivity. Jest to jedna z ważniejszych klas z tego powodu, że to właśnie z jej poziomu mamy dostęp do pozostałych funkcjonalności systemu oraz prezentuje dane dotyczące ostatnio dodawanych kosztów.



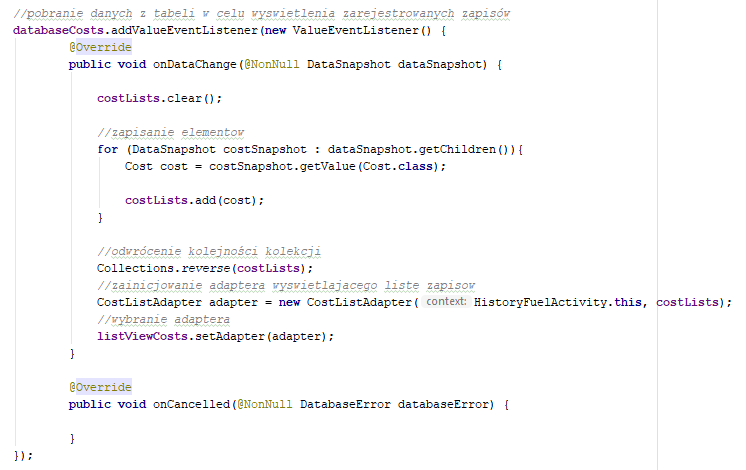
Struktura klasy MenuActivity

Na przykładnie wizualizacji klasy MenuActivity widzimy, że zbudowana jest z pól typu TextView czyli pól interfejsu graficznego. Komponenty te odpowiadają elementom w pliku struktury XML, z którym są mapowane metodą findViewById. Wykorzystując klasę R podawany jest identyfikator komponentu co powoduje, że obiekt zostaje powiązany z jego odpowiednikiem graficznym. Z poziomu klasy MenuActivity mamy możliwość dodania do bazy nowego kosztu eksploatacji poprzez uruchomienie jednej z aktywności: AddFuelActivity, AddProtectionAcitivity, AddReviewActivity lub AddServiceActivity. Klasy te wykorzystują funkcjonalność Firebase Realtime Database.



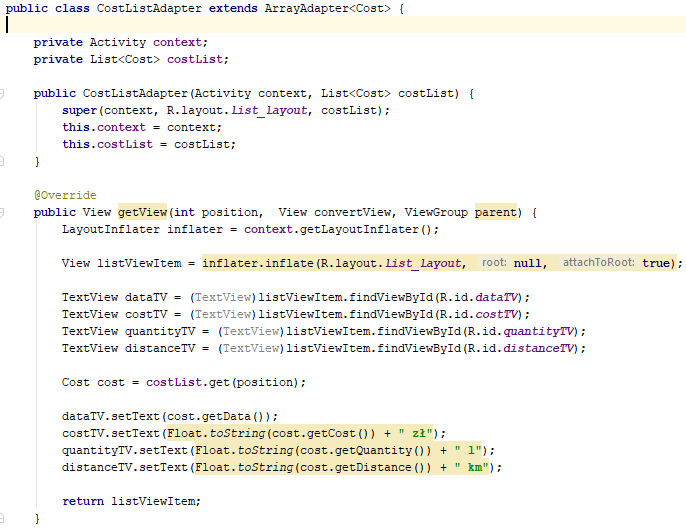
Metoda dodająca obiekt kosztu do bazy danych Firebase

Metoda przedstawiona na rysunku 9 obsługuje dodawanie danych do bazy danych Firebase. Jak widać wykorzystywana jest tutaj klasa Cost, której utworzony obiekt costInformation przechowuje dane dotyczące tworzonego kosztu, gdzie dalej jest wysyłany jako wartość w implementacji metody Firebase addOnCompleteListener. Analogicznie wygląda sytuacja w aktywności CarInfoActivity, w której inicjujemy obiekt typu Car a następnie wykorzystując metodę dodawania danych do Firebase obiekt przesyłamy. Odrębną grupą natomiast są aktywności HistoryFuelActivity, HistoryProtectionActivity, HistoryReviewActivity oraz HistoryServiceActivity, których zadaniem jest pobieranie zapisanych danych z bazy danych i wyświetlanie ich w formie listy. Do odczytu danych z bazy wykorzystywane są jak w przypadkach wcześniejszych metody z biblioteki Firebase.



Przykład implementacji metody pobierającej dane z bazy danych Firebase

Jak widać na rysunku 8 podczas metody pobierającej wykorzystywany jest obiekt klasy CostListAdpater. W programie stworzone są 2 klasy tego typu: CostListAdpater oraz CostShortAdapter. Ich działanie polega na przekazywaniu danych ze źródła danych do widoku. Do wyświetlenia ich na ekranie wykorzystywany jest obiekt ListView, który jest wliczany w typ ArrayAdapter. Platforma Android udostępnia kilka wbudowanych rodzajów adapterów ale programista może stworzyć własny.



Przykład własnej klasy adaptera – CostListAdapter

Na potrzeby omawianej aplikacji został stworzony m.in. adapter CostListAdapter. Jak widać na rysunku 9 klasa ta dziedziczy po standardowym adapterze ArrayAdapter. W pierwszej części widzimy definicję konstruktora, który przekazuje dane do klasy. W dalszej części implementacji przesłonięta zostaje metoda getView. Będzie ona uruchamiana przy każdej potrzebie wyświetlenia elementów w liście, na przykład podczas przewijania widoku listy. Zdefiniowany obiekt convertView służy optymalizacji, gdyż może wystąpić sytuacja gdy lista elementów będzie zbyt duża by je wyświetlić na raz, np. 1000 elementów. Wtedy istniejący obiekt convertView zapewnia stronicowanie danych i elementów interfejsu oraz zmienianie ich zawartości co w połączeniu z możliwościami ListView pozwala na przewijanie zapisów. Kolejno inflater ma za zadanie przekształcenie omawiany wcześniej układ na klasę a widoczne dalej metody findViewById klasycznie pobierają uchwyty do elementów klasy View. W następnym kroku do obiektu cost zostaje zapisany pobrany element danych. Po tym możemy odpowiednim polom widoku ustawić wartości z pobranych danych.[[2]](#footnote-2)

Będąc w temacie aktywności prezentujących pobrane dane należy wspomnieć, że wszystkie implementują interfejs interface\_delete. Interfejs ten definiuje metodę abstrakcyjną showUpdateDialog, która odpowiadając za wyświetlenie okna AlertDialog z przyciskiem Delete oraz metodę abstrakcyjną deleteCost obsługującą usuwanie zaznaczonego zapisu z ListView.

Ostatnią omawianą klasą jest aktywność StatsActivity. Jej zadanie jest uzyskanie dostępu do wszystkich tabel i wszystkich danych w bazie danych po to by zaimplementowane w niej metody mogły zwrócić informacje do użytkownika na temat m.in.: całkowitego kosztu eksploatacji pojazdu, całkowitego kosztu kilometra, całościowego kosztu paliwa, przeglądów itp. Klasa ta uruchamia listenery pobierające dane z 4 tabel bazy danych. W trakcie implementacji pojawić się problem z wychwyceniem momentu zakończenia ściągania danych, gdyż proces ten odbywa się asynchronicznie i przez to nie wiadomo, w którym momencie działania kodu dane będą pobranie i dostępne by można było je poddać obliczeniom i wyświetlić wyniki w polach widoku.

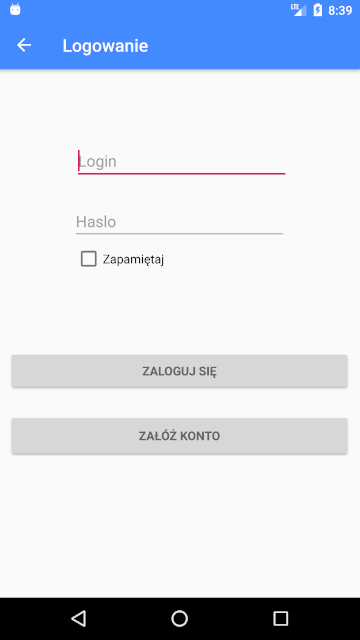


Metoda onDatabaseResult

W celu rozwiązania problemu stworzono metodę z rysunku 12 onDatabaseResult, która porównuje wartości 2 zmiennych. Właściwość currentNumberOfDatabaseRequest jest inkrementowana z rozpoczęciem każdego z procesów pobierania danych i jest porównywana ze zmienną numberOfDatabaseRequest, która ma zadeklarowaną całkowitą liczbę wystąpień sekcji, które muszą być pobrane zanim będą ustawiane wartości pól okna aktywności. Gdy warunek zostanie spełniony zostaje uruchomiony kod metody, który w wyniku zwraca wartości statystyczne dotyczące kosztów eksploatacji pojazdu i ustawia pola interfejsu użytkownika.

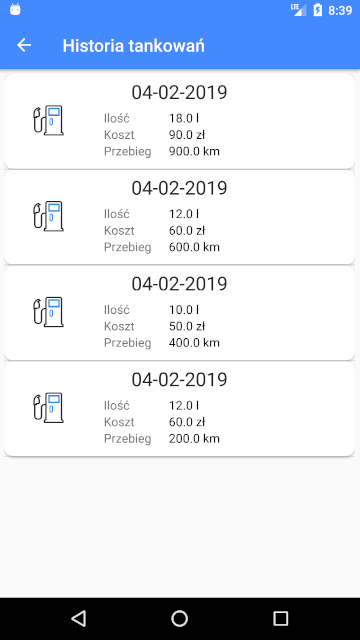
Implementację interfejsu graficznego aplikacji oparto na Material Design. Jest to język projektowania stworzony przez Google, który na pierwszym miejscu stawia czytelność oraz użyteczność kompozycji. Material to dopracowany i uporządkowany system , który jest standardem stosowanym w projektach interfejsów aplikacji na platformę Android. Aby spełnić założenia prostoty i czytelności interfejsu i ograniczyć natłok informacji ukryto na przykład w aktywności MenuActivity część przycisków uruchamiających kolejne okna aplikacji – pojawiają się dopiero po rozwinięciu sekcji kosztu.

Rezultaty implementacji graficznej prezentują rysunki 23.



Widok aktywności menu aplikacji

Widok aktywności logowania



Widok aktywności z listą dodanych kosztów

Widok aktywności wyliczającej koszty eksploatacji pojazdu

Widzimy, że aplikacja spełnia określone funkcjonalności oraz zwraca oczekiwane dane, przy czym jest prosta i intuicyjna w użyciu.

### Testowanie

W obecnych czasach poziom zaawansowania urządzeń mobilnych jak i złożoność oprogramowania na nie tworzonego wymusza konieczność tworzenia i przeprowadzania dokładnych testów kodu programu. Przed udostępnieniem aplikacji do odbiorcy docelowego należy poddać ją dokładnym i wszechstronnym testom.

Najczęściej myśląc o testowaniu większość ludzi ma skojarzenia związane z testowaniem ręcznym przez użytkownika, natomiast istnieje wiele technik testowania i kategoryzowania testów oprogramowania. Wyróżniamy:

* Testy manualne

Testy manualne należy rozumieć dokładnie tak jak nazwa wskazuje czyli są to sekwencje testowe wykonywane manualnie przez osobę fizyczną – testera w celu znalezienia błędu w oprogramowaniu. Potrzeba bierze się stąd, że testy automatyczne bardzo często nie są w stanie przewidzieć i wychwycić wszystkich błędów posiadanych przez tworzoną aplikację. Testowanie manualne również posiada ograniczenia, takie jak ilość możliwych urządzeń na których będzie pracować, stąd często opracowuje się strategie testowe. Pozwalają one na ograniczenie listy i zweryfikowanie najistotniejszych możliwych testów, na przykład testowanie aplikacji na skrajnych urządzeniach, posiadających najwyższą i najniższą możliwą konfigurację (np. wersję Androida, wielkość ekranu, rozdzielczość).

* Testy automatyczne

Testy automatyczne są to testy pisane i przeprowadzane przez programistów. Zaletą testów automatycznych jest to, że poza weryfikacją poprawności funkcjonalności aplikacji w aktualnym stanie zapewniają zapobieganie powstawania przypadkowych błędów przy zmianach w przyszłości. Testy powinny sprawdzać jak najwięcej możliwych przypadków. Warto pamiętać o sprawdzeniu przypadków brzegowych, gdyż jest to obszar w którym często dochodzi do popełnienia przez programistę błędu. Gdy wszystkie zaimplementowane testy nie zgłaszają błędów można założyć, że dana funkcjonalność spełnia warunki poprawności w określonym obszarze. Natomiast jeżeli w aplikacji pojawi się jednak błąd, na podstawie przeprowadzonych testów można wykluczyć obszar jego wystąpienia. Platforma Android zapewnia obsługę dwóch rodzajów takich testów:

* + Jednostkowe – testy służące do testowania wyłącznie na danej funkcjonalności, skupiające się na pracy tylko sprawdzanej jednostki, a nie powiązanego z nią kodu. Takie zachowanie zapewnia precyzję testów, które są odporne na efekty uboczne, które mogłyby być wywołane przez wpływ kodu, który nie jest sprawdzany. Częstym przypadkiem w którym stosowany jest ten typ testów są testy klas.
  + Integracyjne tzw. Funkcjonalne[[3]](#footnote-3) ponieważ taka nazwa jest używana w dokumentacji Android – są to testy, które w odróżnieniu od jednostkowych pozwalają na sprawdzenie działania kilku jednostek kodu z aplikacji. Ponadto sprawdzane jednostki kodu nie działają niezależnie od siebie ale wzajemnie na siebie wpływają, co pozwala na przeprowadzenie kompleksowych testów. Ten rodzaj w Androidzie wykorzystywany jest najczęściej do implementowania testów symulujących scenariusze zadań wykonywane przez użytkowników, sprawdzają jak działają cale funkcjonalności aplikacji.

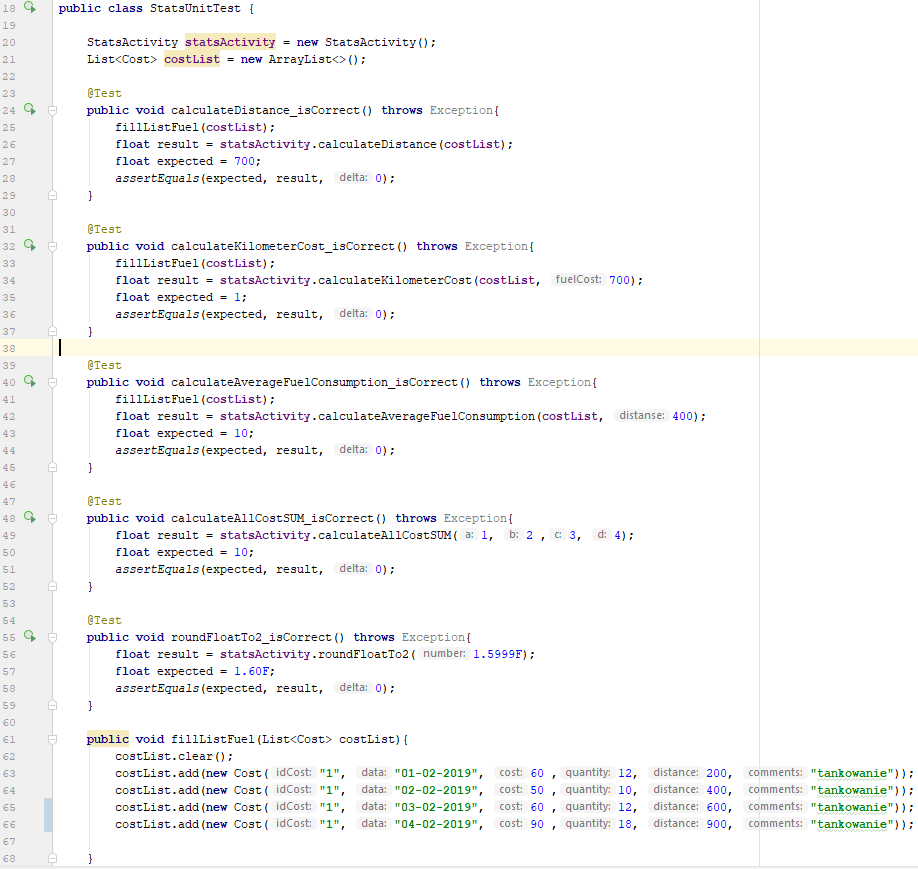
Poza podziałem testów na jednostkowe i funkcjonalne warto poznać dwie metody ich przeprowadzania:

* Testy oparte na Javie, uruchamiane na maszynie wirtualnej JVM (tak zwane testy lokalne). Przeprowadzanie testów w ten sposób polega na uruchamianiu ich na maszynie na której uruchamiane są zwykłe aplikacje w Java – JVM. Dalvik, czyli maszyna wirtualna wbudowana w Android przy tego typu testach nie jest wykorzystywana. Przeprowadzanie testów kodu na maszynie JVM jest w pełni dopuszczalną metodą pod warunkiem, że testowany kod nie korzysta z klas frameworku Androida. Największymi zaletami takiego podejścia jest:
  + Szybkość, ponieważ nie ma potrzeby przesyłania kodu testów do urządzenia lub do emulatora.
  + Elastyczność – tworząc testy nie jesteśmy ograniczeni do korzystania wyłącznie ze środowiska uruchomieniowego Androida, można swobodnie wykorzystywać inne frameworki, np. JUnit
* Testy oparte o framework Androida, tak zwane instrumentalne to testy, które polegają na uruchamianiu ich bezpośrednio na emulatorze lub urządzeniu z oprogramowaniem Android. I w kontekście Androida ten sposób jest bardziej preferowany, ponieważ testy, mają pełny dostęp do funkcji systemu urządzenia, a tym samym pozwala to na uruchomienie niemal każdego dowolnego testu. Trzeba mieć jednak świadomość, że rozwiązanie to posiada też wady, takie jak konieczność wykonania dodatkowych operacji z powodu potrzeby umieszczenia kodu testów na urządzeniu lub emulatorze oraz wolne działanie, ponieważ samo umieszczenie testów w urządzeniu to dłuższy proces. Stąd ten typ testów traci swoją skuteczność w sytuacji gdy nacisk kładziony jest na szybkość wykonania.

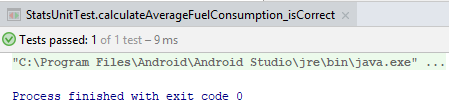
Również w przypadku omawianego projektu zostały zrealizowane działania testowe. Ich celem było wstępna weryfikacja działania aplikacji i ewentualne wykrycie błędów już na etapie implementacji i zapobieganie ich pojawieniu się w przyszłości. Przeprowadzanie testów odbyło się na 2 poziomach: testy manualne oraz testy automatyczne.

Testy automatyczne zostały podzielone na 2 rodzaje testów:

* Testy jednostkowe, które obejmowały konkretne fragmenty aplikacji, skupione zostały głównie na kodzie klasy StatsActivity. Klasa ta odpowiada za przeliczanie wartości statystycznych wyliczanych z zarejestrowanych kosztów używania pojazdu i zwraca je w jednym z widoków do informacji użytkownika.



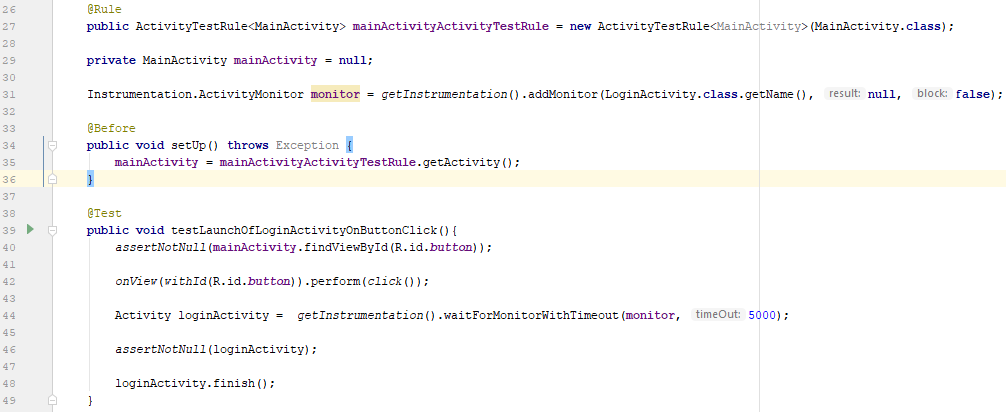
testy jednostkowe klasy StatsActivity



Wynik testu calculateAverageFuelConsumption\_isCorrect

Wykonane testy jednostkowe potwierdzały, że zwracane wyniki przez aplikację są poprawne co pozwalało na kontynuację implementacji i ewentualną zabezpieczoną przed błędami ingerencję w kod w przyszłości.

* Testy funkcjonalne obejmowały sprawdzenie scenariuszy obejmujących różne jednostki kodu, czego przykładem może być przechodzenie między ekranami wywoływane w reakcji na kliknięcie w przycisk.



Przykład testu funkcjonalnego – testLaunchOfLoginActivityOnButtonClick

W tym celi zaimplementowano i wykorzystano możliwości frameworku Espresso – jest to struktura dostarczana przez AndroidX, która zapewnia API do pisania testów interfejsu użytkownika. Testy Espresso mogą być uruchamiane na urządzeniach z poziomem API min. 10. Podstawową zaletą tej biblioteki jest zapewnienie automatycznej synchronizacji działań testowych z interfejsem użytkownika[[4]](#footnote-4).

Poza testami automatycznymi przeprowadzano testy manualne. W tym przypadku nie mamy do czynienia z weryfikacją wykonywaną przez program a testowanie odbywa się przez realizację konkretnych scenariuszy testowych. Testy odbywały się dwuetapowo: na emulatorze Android dostarczanym przez środowisko programistyczne Android Studio oraz na rzeczywistym urządzeniu mobilnym, którym był telefon komórkowy LG G3 z oprogramowaniem Android w wersji 6.0. Przykładowy scenariusz testowy prezentuje się tak:

***Warunek początkowy:***

* Uruchomione połączenie z Internetem

***Realizowane kroki:***

* Uruchomienie aplikacji
* Akceptacja kliknięciem w przycisk „Rozpocznij”
* Wprowadzenie Loginu oraz Hasła
* Zapamiętanie danych użytkownika poprzez kliknięcie w checkbox „Zapamiętaj”
* Akceptacja poprzez kliknięcie w przycisk „Zaloguj”
* Po pojawieniu się okna z menu programu kliknięcie w przycisk w sekcji „Ostatnie tankowanie”
* Kliknięcie w przycisk „Dodaj” co otworzy w nowe okno
* Uzupełnienie danych typu: Data, Ilość, Koszt, Przebieg
* Akceptacja przyciskiem „Zapisz” co wróci nas do okna Menu programu
* Ponowne kliknięcie w przycisk w sekcji „Ostatnie tankowanie”
* Kliknięcie w przycisk „Historia” co otworzy nowe okno z listą dodanych tankowań
* Przytrzymanie kursora na jednej z pozycji listy
* W oknie Akcje kliknięcie przycisku „DELETE”

***Oczekiwany rezultat***:

* Poprawne zalogowanie się użytkownika do aplikacji
* Sprawne przechodzenie pomiędzy oknami programu
* Dodanie do bazy danych zapisu z zarejestrowanym przez użytkownika kosztem i poprawne wyświetlenie go w oknie historii zarejestrowanych kosztów
* Możliwość usunięcia dodanego zapisu z poziomu okna historii zarejestrowanych kosztów

1. M.Collins, M. Galpin, M. Kaeppler, Android w praktyce, wydawnictwo HELION, Poland 2012, s. 74 [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)
3. M.Collins, M. Galpin, M. Kaeppler, Android w praktyce, wydawnictwo HELION, Poland 2012, s. 526. [↑](#footnote-ref-3)
4. https://developer.android.com/training/testing/espresso/basics [↑](#footnote-ref-4)