POLITECHNIKA WARSZAWSKA

WYDZIAŁ MATEMATYKI

I NAUK INFORMACYJNYCH

PRACA DYPLOMOWA INŻYNIERSKA

INFORMATYKA

Gogle do rzeczywistości wirtualnej i rozszerzonej oparte o ekran i kamerę smartfona

Virtual and augmented reality goggles based on smartphone's screen and camera

Autor:

Sylwia Nowak

Współautorzy:

Anna Zawadzka

Jakub Cieślik

Promotor:

dr inż. Paweł Kotowski

Warszawa, luty 2016

....

podpis promotora podpis autora

STRESZCZENIE

Celem niniejszej pracy dyplomowej było stworzenie aplikacji przeznaczonej na urządzenia mobilne realizującej koncepcję gogli do rozszerzonej i wirtualnej rzeczywistości.

W naszej pracy skupiliśmy się na połączeniu wielu rozwiązań w celu zaoferowania użytkownikowi jak największej ilości doznań, które dostarcza rzeczywistość rozszerzona. Połączyliśmy funkcjonalności kamery smartfona, serwisu *Google Maps* oraz platformy *Google Cardboard*. W ten sposób udało nam się skonstruować aplikację ***FindMyMeal***, która pozwala na odnajdywanie najlepszych restauracji w pobliżu użytkownika. W aplikacji znalazły się również funkcjonalności pozwalające na tworzenie kolekcji ulubionych miejsc, możliwość wybrania jednego miejsca będącego celem nawigacji oraz film instruktażowy obrazujący właściwe korzystanie z aplikacji.

Projekt oparty jest o platformę *Android* (w wersji *KitKat* i nowszej). Ponadto urządzenie, na którym uruchamiana jest nasza aplikacja, musi być wyposażone w zestaw sensorów, takich jak: magnetometr i akcelerometr. Użytkownik musi zagwarantować jej stały dostęp do Internetu oraz zezwolić na odczytywanie danych o bieżącym położeniu geograficznym.

Elementem niezbędnym do działania aplikacji *FindMyMeal* jest moduł gogli *Cardboard*. Kartonowe pudełko wyposażone w dwie soczewki skupiające oraz podział ekranu na dwie części i niezależne generowanie obu obrazów zapewnia użytkownikowi przestrzenną wizję. Interakcję z interfejsem umożliwia przycisk magnetyczny, który zastępuje dotyk ekranu urządzenia. Poruszanie się w obrębie aplikacji (np. menu główne) możliwe jest dzięki ruchom głowy.

Aplikacja zbudowana jest w oparciu o model komponentowy. Dzięki temu zyskała niezależność swoich modułów oraz możliwość łatwego wprowadzania zmian. Ponadto napisane przez nas komponenty mogą być adaptowane do innych projektów, a dodawanie nowych rozszerzeń do naszej aplikacji nie powinno stwarzać jakichkolwiek problemów.

Kierunkiem dalszego rozwoju aplikacji może być jej dalsze dopracowywanie, tak aby była w jak największym stopniu kompatybilna z kolejnym produktem - *Google Glass*.

ABSTRACT

The aim of this thesis was to create an application designed for mobile devices implementing the concept of augmented reality goggles.

In our work we focused on combining multiple solutions in order to offer the user the greatest possible experience which augmented reality gives. We have combined the functionality of a smartphone camera, *Google Maps* and the *Google Cardboard* platform. Using this, we were able to develop ***FindMyMeal*** application that offers to user the ability to find the best restaurants in her/his vicinity. The application also includes functionality for creating a collection of user's favourite places, ability to choose one place for the target navigation and an instructional video that is showing the proper use of an application.

Whole project is based on the *Android* platform (version *KitKat* and grater). Furthermore to run the application the device must be equipped with a set of sensors such as magnetometer or accelerometer. User must guarantee the Internet connection and allow our application to access the data about the current geographical location.

FindMyMeal requires *Cardboard* goggles module. *Cardboard* box is equipped in two focusing lens. Splitting screen into two parts and generating two independent images provides user a spatial vision. Interface is interacting with the magnetic button - this replaces the touch on screen device. Navigating within the application (eg. at the main menu) is possible due to movements of the head.

The application is built in *Component Object Model*. Thanks to the independence of its' modules we gained the ability to make changes more easily. Components written by us can be adapted in other projects. Adding new extensions to our application should not create any problems.

We already have directions of further development of the application. It should be compatible with next product - *Google Glass*.

Spis treści

1 Wstęp

1.1 Rzeczywistość wirtualna i rozszerzona

Rzeczywistość rozszerzona (*poszerzona rzeczywistość*, ang. *augmented reality*) to metodologia pracy z systemami informatycznymi, polegająca na łączeniu świata rzeczywistego z obiektami pochodzącymi ze świata wirtualnego. Nakładanie informacji odbywa się za pośrednictwem *wirtualnej powłoki* i następuje w czasie rzeczywistym. Rzeczywistość rozszerzona zdobywa informacje o otoczeniu dzięki przetwarzaniu danych z kamery, czujników lokalizacji (takich jak *GPS* lub poprzez wykonanie tzw. *triangulacji*) oraz dzięki odczytom z sensorów położenia mówiących np. o jego fizycznym położeniu względem powierzchni Ziemi. Dzięki temu generowane w ramach *wirtualnej powłoki* rozszerzenia mogą przybierać wszelaką postać. Poczynając od prostych informacji nałożonych na świat rzeczywisty (np. nazwy ulic lub nawigacja) aż do skomplikowanych obiektów fotorealistycznych, które wtapiają się w świat realny i tworzą z nim jedną całość (np. rekonstrukcje zniszczonych historycznych budynków czy symulacje militarne).

Rzeczywistość wirtualną (ang. *virtual reality*) od rzeczywistości rozszerzonej odróżnia fakt niewchodzenia w interakcje z obiektami rzeczywistymi, poprzestając jedynie na wyświetlaniu generowanych komputerowo obiektów. W 1994 roku Paul Milgram i Fumio Kishino sformułowali definicję rzeczywistości mieszanej (ang. *mixed reality*) za pomocą koncepcji schematu ciągłości rzeczywistość – wirtualność (ang. *virtuality continuum*) przedstawiającego relacje między rzeczywistym a wirtualnym światem.

Rysunek 1. Schemat ciągłości rzeczywistość – wirtualność
Źródło: R. Azuma, Y.Baillot, R. Behringer, S.Feiner, S. Julier, B.MacIntyre, Recent Advances in Augmented Reality

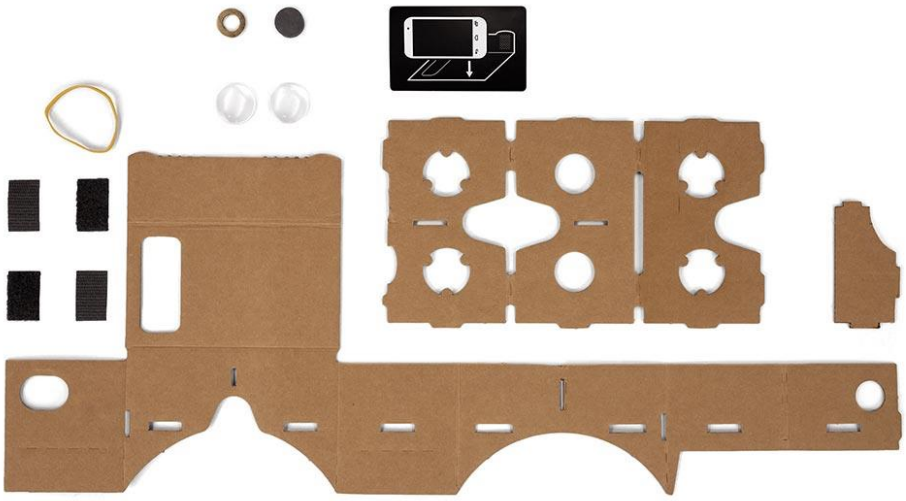
Za jeden z końców powyższego schematu przyjęto środowisko rzeczywiste (ang. *real environment*), obok którego umiejscowiono rzeczywistość rozszerzoną (ang. *augmented reality*). Im bliżej systemowi do wirtualnej rzeczywistości, tym bardziej zredukowana jest liczba elementów rzeczywistych. Rzeczywistość może być „rozszerzana" o wirtualne obiekty, analogicznie do tego wirtualny świat może być „rozszerzany" przez rzeczywiste obiekty. Takie środowisko nazywane jest rozszerzoną wirtualnością (ang. *augmented virtuality*). Na schemacie umiejscowiona jest tuż przy środowisku wirtualnej rzeczywistości (ang. *real environment*). Opanowanie idei całego schematu pomaga w klasyfikacji wszystkich systemów, w których mieszany jest świat rzeczywisty i wirtualny.

Aplikacje wykorzystujące rzeczywistość wirtualną i rozszerzoną najczęściej spotykane są na takich urządzeniach jak telefony i tablety. Proces miniaturyzacji sprzętu oraz chęć uczynienia go coraz bardziej funkcjonalnym powodują, iż rynek domaga się rozwiązań zintegrowanych, łączących maksimum funkcjonalności w jednym urządzeniu. Jedyną wadą wymienionych wcześniej urządzeń jest stosunkowo niewielki obszar roboczy wyświetlacza, ograniczający w pewien sposób możliwości pełnej interakcji użytkownika z aplikacją. Dlatego też pionierzy w dziedzinie rozwiązań mobilnych opartych o rozszerzoną rzeczywistość dążą do opracowania okularów przeziernikowych, pozwalających oglądać świat własnymi oczami. Firma *Google* jako jedna z pierwszych postanowiła poważnie zainwestować w projekt oparty o takie rozwiązanie. Jednak ich pomysł na *Google Glass* z szerzej nie znanych powodów nie został oddany do szerokiej sprzedaży. Amerykański producent postanowił wystartować z dużo bardziej dostępnym produktem *Google Cardboard*, który odróżnia się tym, iż obraz trafia do użytkownika za pośrednictwem ekranu i kamery telefonu przytwierdzonego do specjalnego, zewnętrznego modułu.

1.2 Google Cardboard

Google Cardboard to platforma wirtualnej rzeczywistości stworzona przez firmę *Google* do użycia z kartonowymi goglami oraz smartfonem.

Jednym z elementów *Google Carboard* są gogle, składające się z odpowiednio wyciętych kartonowych części, dwóch soczewek o średnicy 25 mm i ogniskowej około 40 mm, dwóch magnesów (jeden neodymowy oraz jeden ferrytowy lub ceramiczny), rzepów oraz gumki podtrzymującej smartfon.

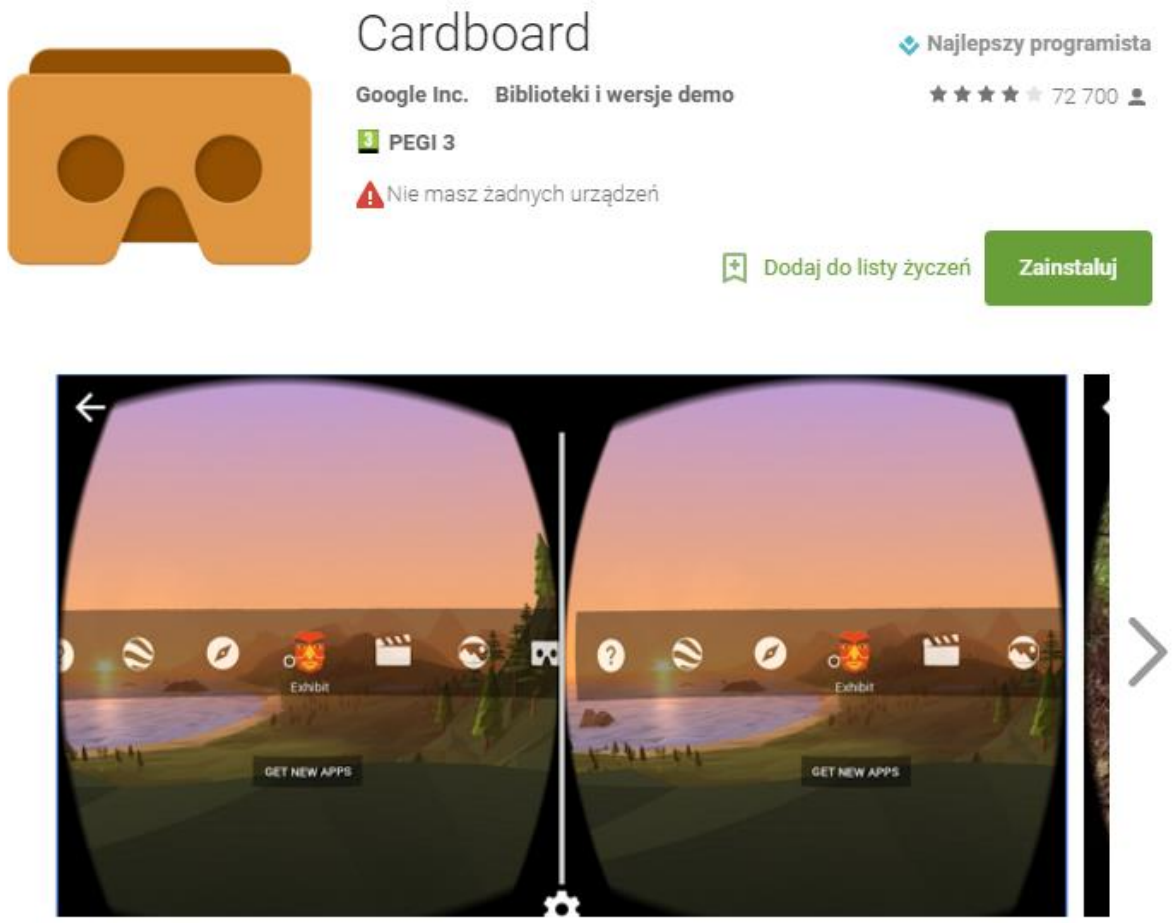


Rysunek 2. Elementy gogli Google Cardboard przed złożeniem
Źródło: https://www.google.com/intl/pl_ALL/get/cardboard/get-cardboard/

Rysunek 3. Gogle Google Cardboard po złożeniu
Źródło: http://www.in.techradar.com/news/Google-I/O-2015-Heres-how-the-magic-on-Google-cardboard-works/articleshow/47471574.cms

Aplikacje kompatybilne z *Google Cardboard* dostępne są do zainstalowania w Sklepie *Google Play* lub *Apple App Store*. Najważniejszym ich elementem jest podzielenie ekranu na dwie części i wyświetlenie jednocześnie dwóch identycznych obrazów - po jednym dla każdego oka. Dzięki niewielkiemu przesunięciu wyświetlanych części uzyskuje się efekt widzenia trójwymiarowego. Kolejnym niezbędnym aspektem jest śledzenie ruchów głowy (ang. *head tracking*). Dzięki akcelerometrom, które śledzą zarówno położenie jak i kąt nachylenia urządzenia, możliwe jest swobodne rozglądanie się po wirtualnym otoczeniu. W efekcie ruchy na ekranie odzwierciedlają ruchy głowy użytkownika w rzeczywistości.

Jedną z popularniejszych aplikacji dostępnych w *Google Play* jest *Cardboard*. Pozwala ona poznać możliwości jakie daje rozszerzona i wirtualna rzeczywistość, odkrywać nowe aplikacje oraz kalibrować gogle.



Rysunek 4. Aplikacja Cardboard w sklepie Google Play
Źródło: https://play.google.com/store/apps/details?id=com.google.samples.apps.cardboarddemo

Urządzenia kompatybilne z *Google Cardboard*:

- Google/LG Nexus 4, 5, 6
- HTC Eco 3D, One (Mini, S, X, X+), Sensation, Sensation XE, Velocity 4G
- Huawei Ascend G 615 + P1
- iOcean X7
- LG G2, G3, Optimus 3D Max (P720), Optimus 4X HD (P880), Optimus G (E975), Optimus G Pro + P940 Prada 3
- Samsung Aktiv S, Galaxy (Beam, S2, S3, S3 Mini), S4 (Active, Mini)
- Sony Xperia S, SP, T + Z1
- Wiko Highway

Źródło: http://shop.zaak.io/pages/compatible-smartphones

2 Aplikacja FindMyMeal

2.1 Opis aplikacji

Realizacją tematu pracy dyplomowej jest aplikacja mobilna wykorzystująca rzeczywistość rozszerzoną, która może być elementem pomocnym w codziennym życiu użytkownika.

Głównym założeniem aplikacji jest wskazywanie użytkownikowi najbliższych restauracji, stąd też nazwa projektu – *FindMyMeal*. Dodatkowo umożliwia ona użytkownikowi zapamiętywanie ulubionych miejsc oraz pozwala na nawigację do wybranego przez niego lokalu.

Aby umożliwić stały dostęp do usług świadczonych przez aplikację na bieżąco przetwarzane są dane o obecnym położeniu użytkownika pobierane z modułu *GPS*. Dane o lokalizacji restauracji pobierane są przez *Google Maps API*.

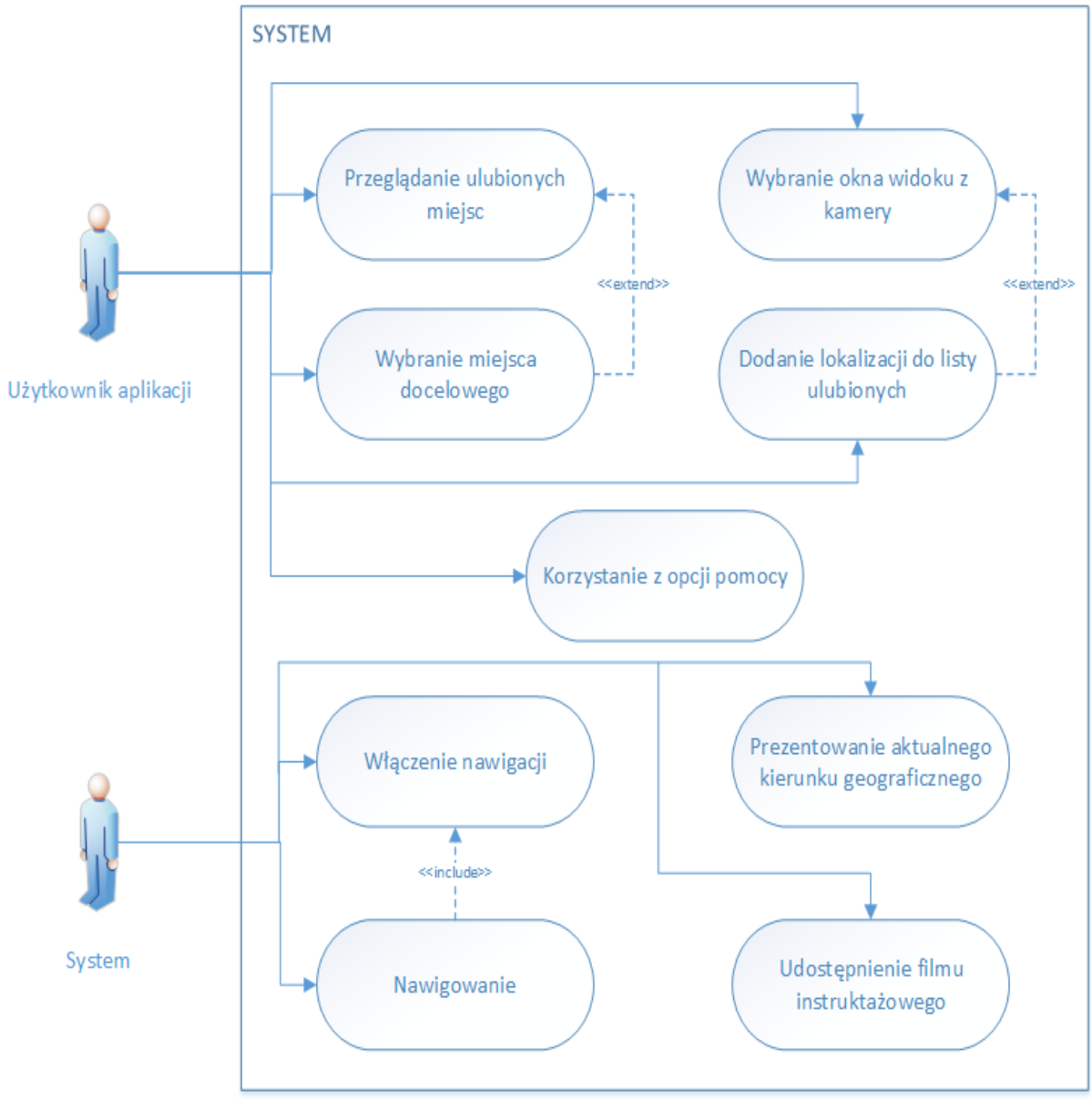
Interfejs użytkownika zbudowany jest przy użyciu biblioteki *Google Cardboard* oraz oparty jest o założenia koncepcji *Material Design*. Jego działanie opiera się na informacjach pochodzących z dwóch akcelerometrów: żyroskopu oraz akcelerometru orientacji. Dodatkowo, w ramach *API Google Cardboard* odczytywane są akcje związane z wciśnięciem przez użytkownika klawisza akceptującego dane działanie bądź akcję. Dane z interfejsu użytkownika używane są dalej do generowania odpowiednich zachowań aplikacji, przez intuicyjną obsługę menu głównego do wyświetlania przybliżonych informacji o kierunku włącznie.

Jak już wspomniano, dodatkowa funkcjonalność aplikacji pozwala użytkownikowi na zapisywanie ulubionych lokalizacji (restauracji, lokali, pubów). Zebrane w ten sposób dane są dostępne z menu głównego aplikacji po wybraniu opcji ulubione miejsca (ang. *Favourites*). Dane zaprezentowane są w postaci wybieralnej listy. Wybranie elementu powoduje przejście do okna z nawigacją do wskazanej przez użytkownika lokalizacji.

W celu zapewnienia jak najwyższej dostępności aplikacji dostępna jest opcja pomocy w postaci filmu instruktażowego.

2.2 Diagram przypadków użycia

Diagram przypadków użycia przedstawia pokrótce funkcjonalność systemu wraz z jego otoczeniem.



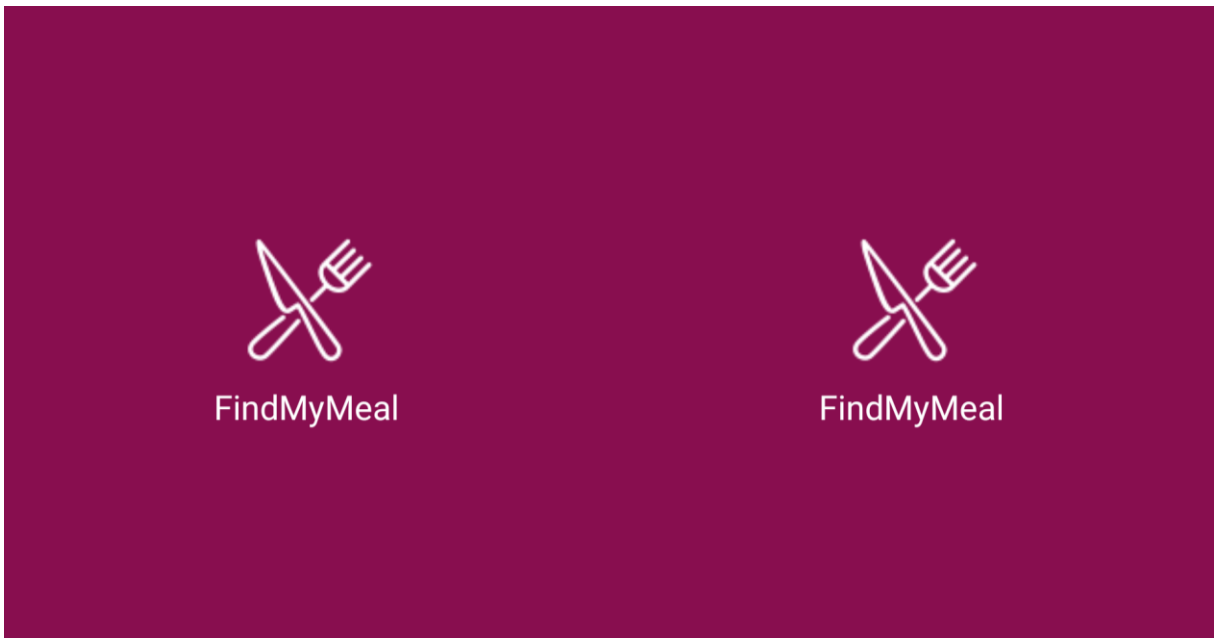
Rysunek 5. Diagram przypadków użycia
Źródło: opracowanie własne

2.3 Opis działania aplikacji

Przed uruchomieniem aplikacji należy upewnić się, że moduł *Google Cardboard* został prawidłowo założony na telefonie.

2.3.1 Uruchomienie aplikacji

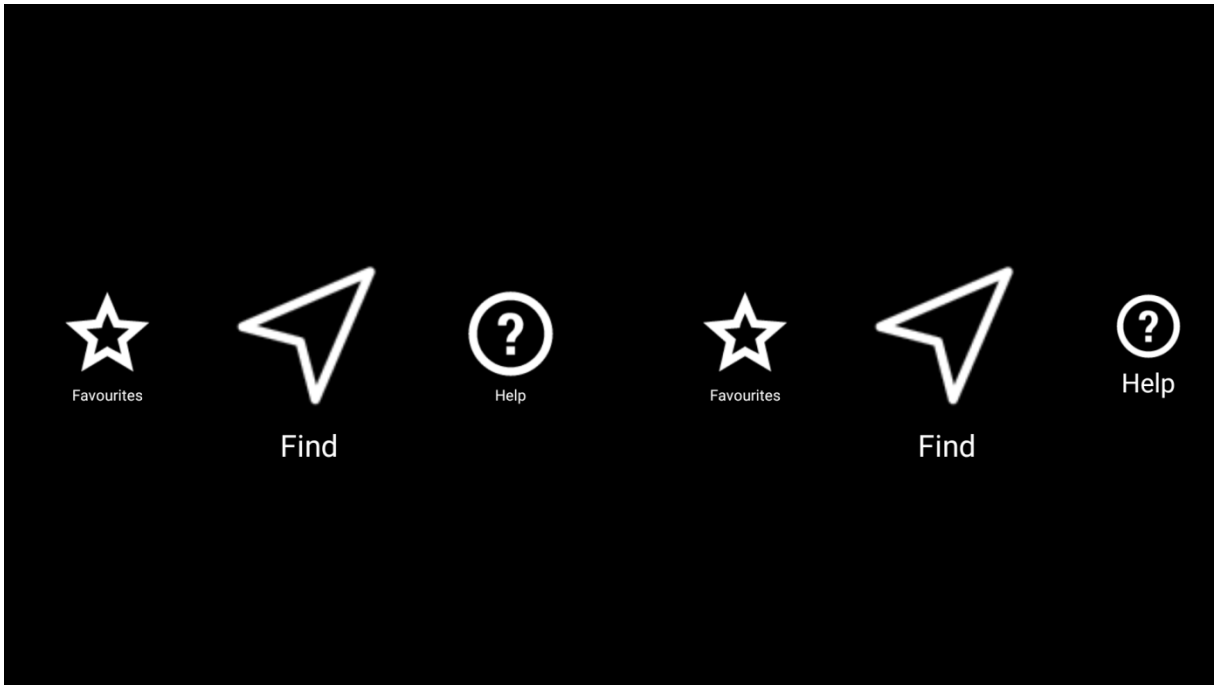
Po uruchomieniu aplikacji na ekranie wyświetlane jest jej logo. W tym momencie w tle aktywowane są wszystkie usługi umożliwiające poprawne działanie poszczególnych modułów aplikacji.



Rysunek 6. Okno ładowania aplikacji
Źródło: opracowanie własne

2.3.2 Menu główne

Po załadowaniu wszystkich komponentów pojawia się ekran zawierający menu główne aplikacji, które zawiera trzy opcje: *Favourites* (*Ulubione*), *Find* (*Znajdź*) oraz *Help* (*Pomoc*).



Rysunek 7. Menu główne
Źródło: opracowanie własne

Przechodzenie między różnymi elementami menu głównego następuje poprzez poruszanie modułem *Cardboard* przez użytkownika w prawą i lewą stronę.

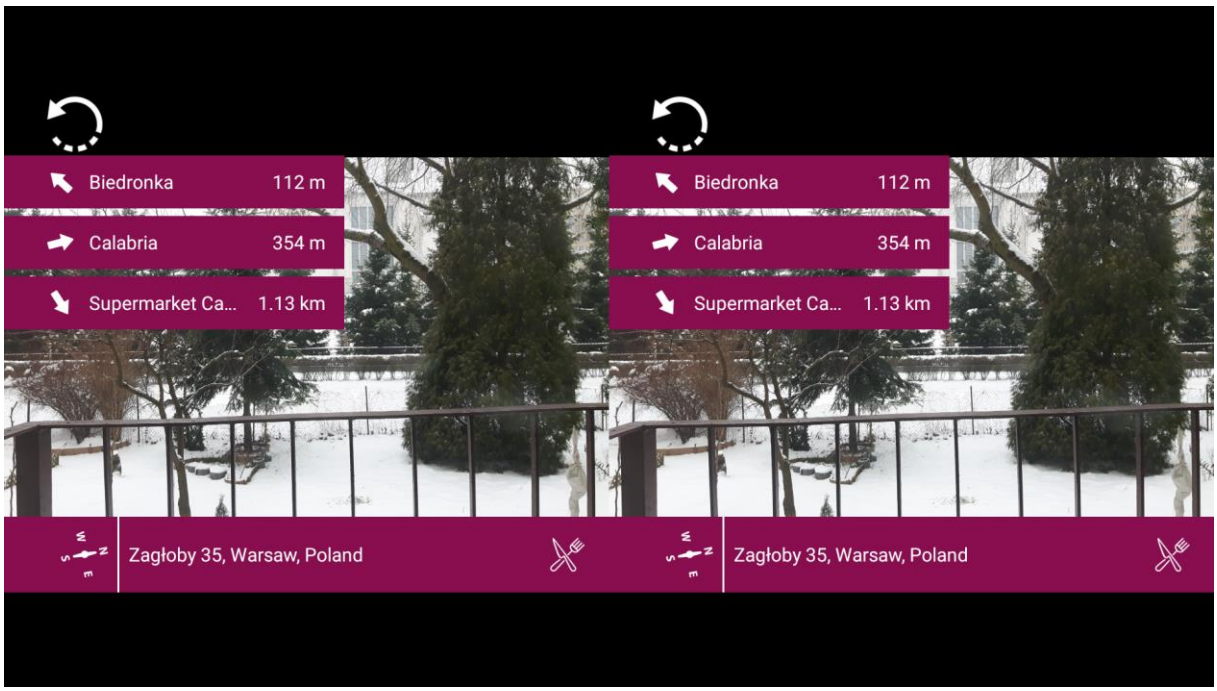
Rysunek 8. Kierunki poruszania modułem Cardboard
Źródło: opracowanie własne

Wybieranie zaznaczonej opcji wykonywane jest przy pomocy przycisku magnetycznego modułu *Google Cardboard*.

Rysunek 9. Wciśnięcie przycisku magnetycznego
Źródło: opracowanie własne

2.3.3 Opcja *Find*

Po wybraniu opcji *Find* następuje przejście do okna nawigacji.



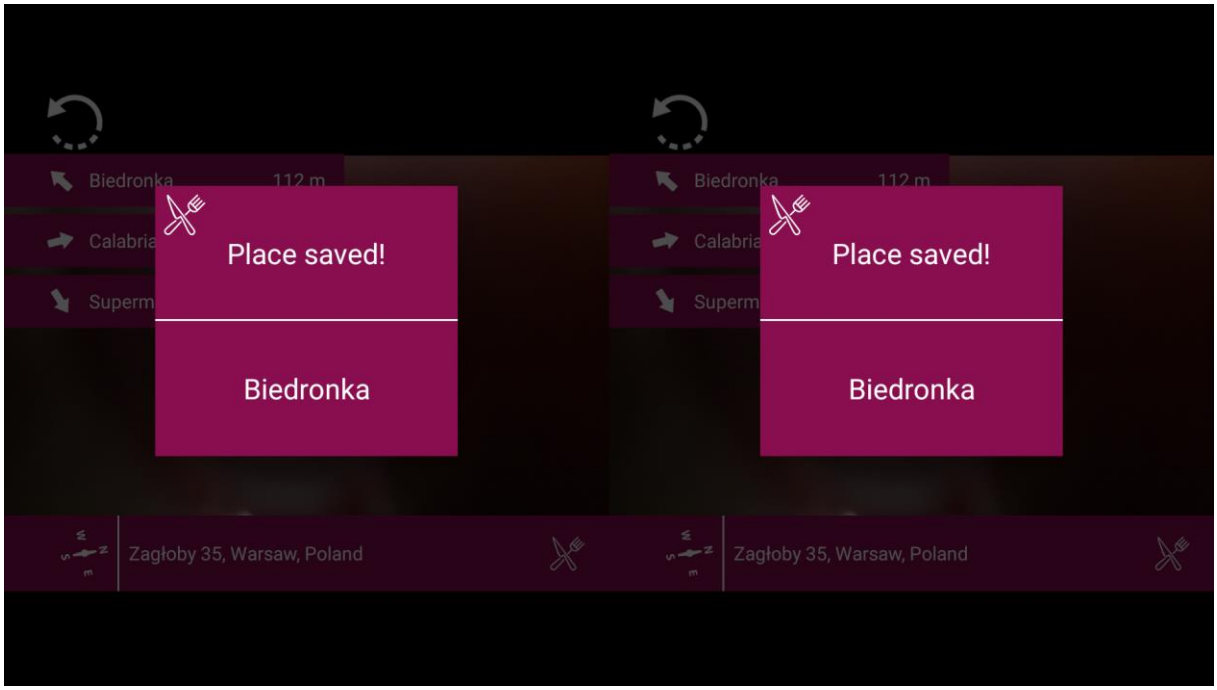
Rysunek 10. Okno nawigacji
Źródło: opracowanie własne

Na ekranie widoczny jest obraz z kamery wzbogacony o informacje o aktualnym położeniu (bieżący adres) oraz o najbliżej znajdujących się restauracjach.

Na pasku stanu (dolna belka) wyświetlany jest pełny adres oraz kompas informujący, w jakim kierunku zwrócony jest użytkownik.

Komunikaty o najbliższych lokalach (co najwyżej trzy komunikaty, zawierające nazwę restauracji, odległość do pokonania oraz strzałkę, wskazującą kierunek poruszania) wyświetlane są w lewym górnym rogu okna. Informacje te będą zmieniać się wraz ze zmianą położenia użytkownika.

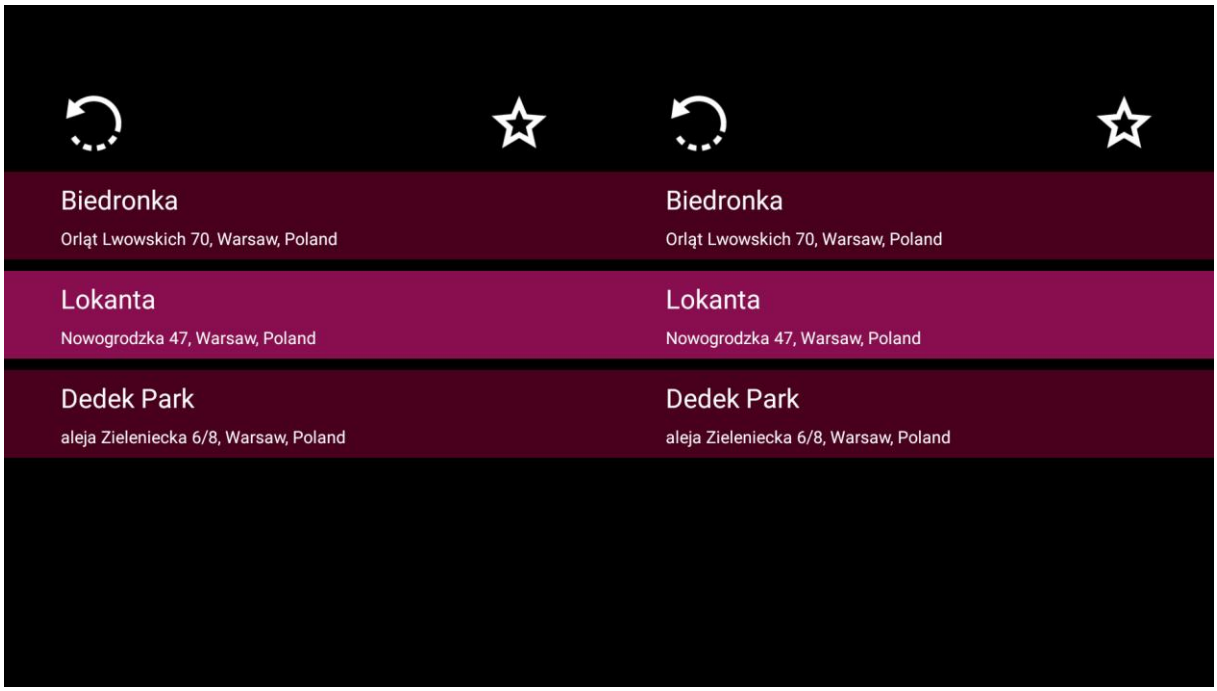
Z poziomu tej opcji możliwe jest dodatkowo dodanie nowego miejsca do listy ulubionych. Następuje to poprzez użycie przycisku magnetycznego – do listy dostępnej pod opcją *Favourites* dodawany jest rekord zawierający nazwę lokalu i adres pierwszego elementu z listy najbliższych restauracji, po czym pojawia się odpowiedni komunikat.



Rysunek 11. Komunikat o dodaniu miejsca do listy ulubionych
Źródło: opracowanie własne

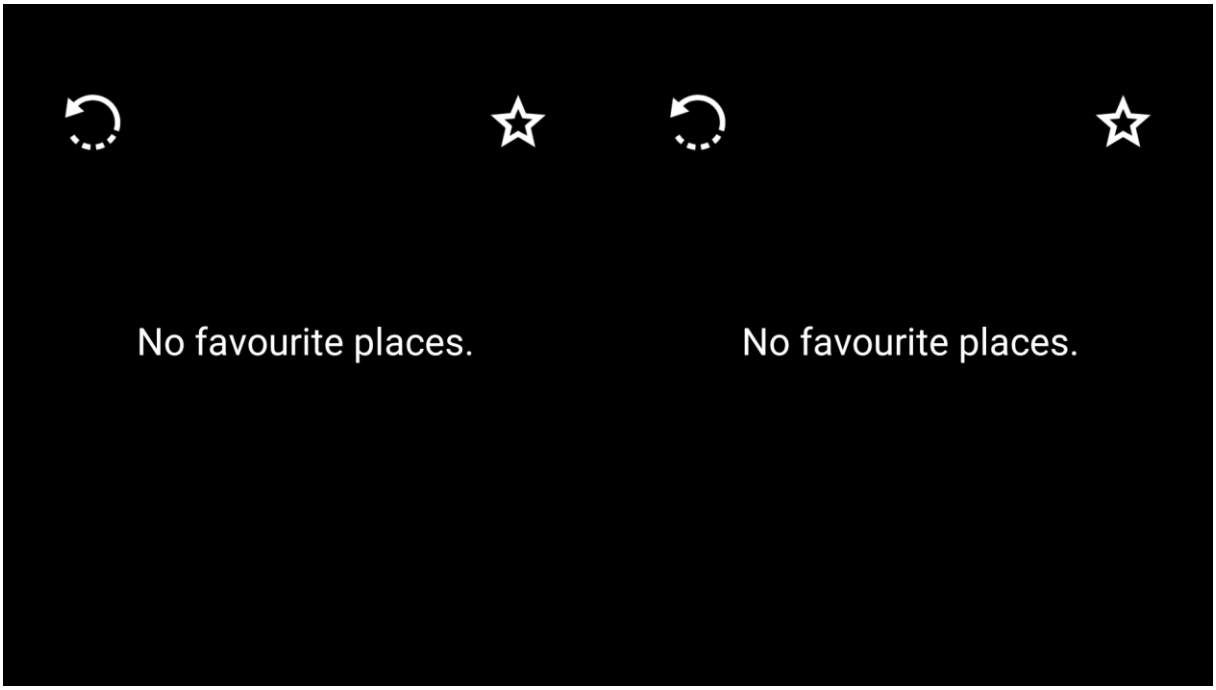
2.3.4 Opcja *Favourites*

Kolejną opcją, którą można wybrać z menu głównego jest *Favourites*. W tym miejscu na ekranie pojawia się lista zapisanych przez użytkownika ulubionych miejsc (nazwa lokalu oraz adres) posortowana malejąco po dacie dodania rekordu.



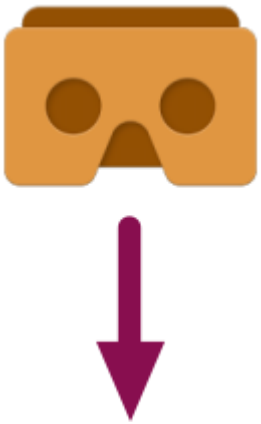
Rysunek 12. Lista ulubionych miejsc
Źródło: opracowanie własne

W przypadku braku zapisanych miejsc wyświetlany jest odpowiedni komunikat.



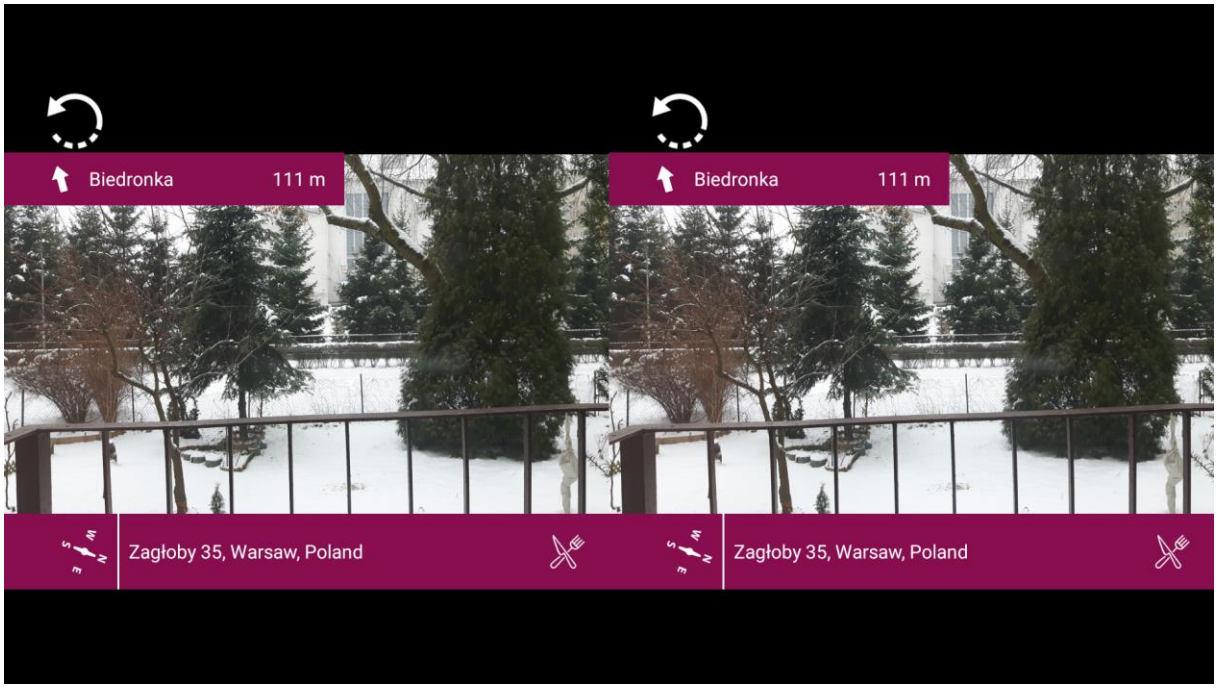
Rysunek 13. Komunikat o braku elementów na liście ulubionych miejsc
Źródło: opracowanie własne

Przechodzenie po liście odbywa się poprzez poruszanie przez użytkownika modułem *Cardboard* w dół. W momencie osiągnięcia końca listy kolejnym zaznaczonym elementem jest pierwsza restauracja na liście (dodana jako ostatnia).



Rysunek 14. Schemat poruszania modułem Cardboard w dół
Źródło: opracowanie własne

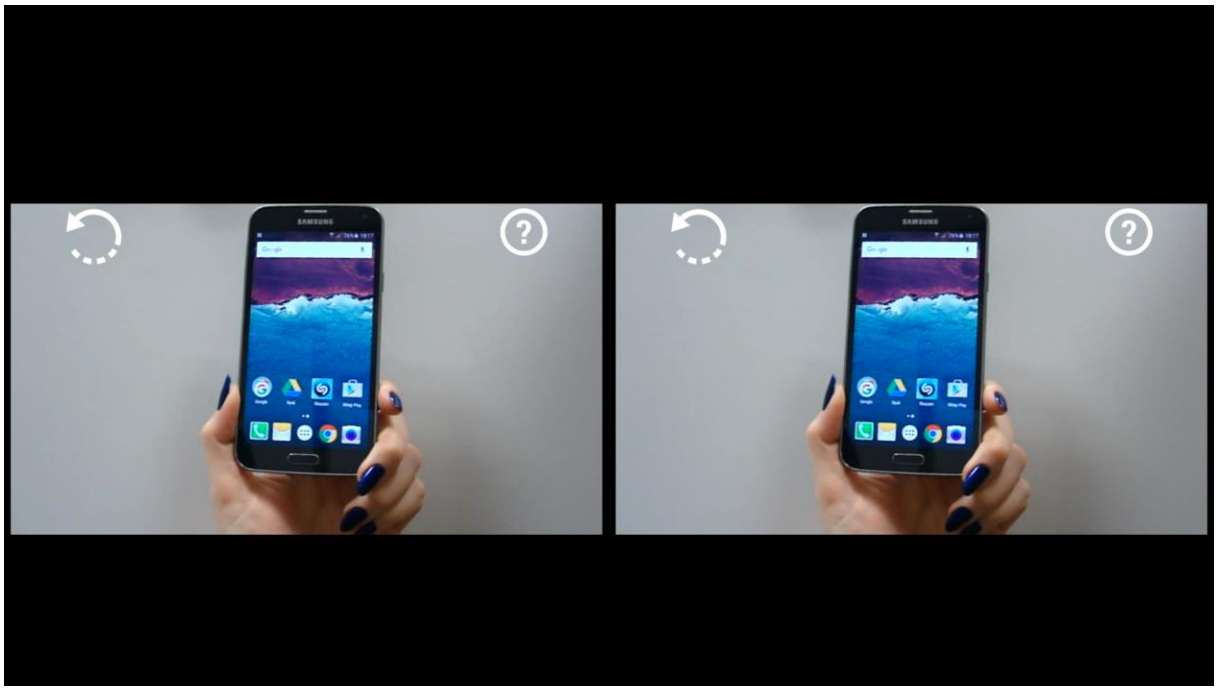
Wybranie elementu listy odbywa się poprzez użycie przycisku magnetycznego. Po wybraniu elementu następuje przejście do okna widoku z kamery uzupełnionego informacją o kierunku poruszania i pozostałym dystansie do wybranego miejsca.



Rysunek 15. Okno nawigacji wybranego ulubionego miejsca
Źródło: opracowanie własne

2.3.5 Opcja *Help*

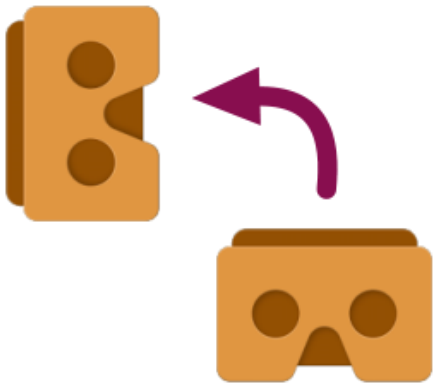
Po wybraniu tej opcji następuje uruchomienie filmu instruktażowego.



Rysunek 16. Okno pomocy
Źródło: opracowanie własne

2.3.6 Powrót

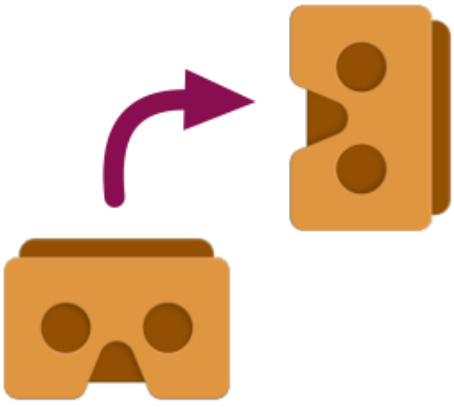
Aby powrócić do poprzedniego okna aplikacji należy przechylić moduł *Google Carboard* w lewo pod kątem 90º, następnie powrócić do normalnego położenia.



Rysunek 17. Ruch modułem Google Cardboard – powrót
Źródło: opracowanie własne

2.3.7 Wyjście z aplikacji

Zgodnie z oficjalnymi wytycznymi Google przechylenie modułu *Carboard* w prawo pod kątem 90º skutkuje wyjściem z aplikacji *FindMyMeal*.



Rysunek 18. Ruch modułem Google Cardboard – wyjście
Źródło: opracowanie własne

3 Wymagane środowisko sprzętowe i systemowe

3.1 Wymagania systemowe

Projekt zrealizowany jest w technologii *Android*.

Minimalne wymagania systemu to *API 19 – Android KitKat*.

3.2 Wymagania sprzętowe

Projekt przeznaczony jest na urządzenia mobilne o minimalnej przekątnej ekranu 4.4''. Przekątna ekranu telefonu nie powinna przekraczać 7'' .

Urządzenie mobilne musi posiadać kamerę oraz czujniki: żyroskop, akcelerometr orientacji oraz akcelerometr pola magnetycznego (tzw. magnetometr).

Urządzenie musi mieć stały dostęp do Internetu oraz wbudowany moduł *GPS*.

3.3 Wymagane moduły

Aby w pełni korzystać z funkcji udostępnianych przez aplikację *FindMyMeal*, użytkownik powinien posiadać dodatkowo zakupiony moduł *Google Cardboard*, który umożliwia obsługę interfejsu aplikacji. Moduł ten powinien być dopasowany do wielkości ekranu urządzenia.

4 Architektura systemu

4.1 Opis architektury

Projekt został zrealizowany w architekturze komponentowej. Aplikacja składa się z następujących modułów:

- *common,*
- *favourites,*
- *help,*
- *home,*
- *main_menu,*
- *navigation.*

Ze względu na architekturę systemu *Android* każdy ekran generowany jest przez odpowiadającą mu aktywność. Aby uwspólnić funkcjonalność ekranów, wprowadzono aktywność *BaseActivity*, po której dziedziczą wszystkie inne aktywności. Dzięki *BaseActivity,* która jest kompatybilna z aktywnością *CardboardActivity* dostępną z *API Google Cardboard,* możliwe jest przechwytywanie akcji wciśnięcia przycisku magnetycznego oraz zmiany położenia telefonu względem ziemi i stron świata. Każda aktywność zachowuje się w sposób identyczny w momencie wykrycia przechylenia telefonu o 90º w prawą stronę – następuje wyjście z aplikacji. Analogicznie dla przechylenia telefonu w lewą stronę o 90º – następuje powrót do poprzedniego ekranu.

Aby przyspieszyć działanie aplikacji, każdy widok przechowywany jest w odpowiadającym mu *Fragmencie*. Dzięki tak zastosowanemu rozwiązaniu aktywności odpowiadają za logikę aplikacji, a *fragmenty* za widoki. Przykładowo aktywność *NavigationActivity* odpowiada za łączenie się z *API Google Maps* w celu uzyskania listy najbliżej położonych, najlepiej ocenianych miejsc i przekazuje dane do *NavigationFragment*, który dla każdej restauracji buduje widok w postaci prostokąta w lewym górnym rogu ekranu.

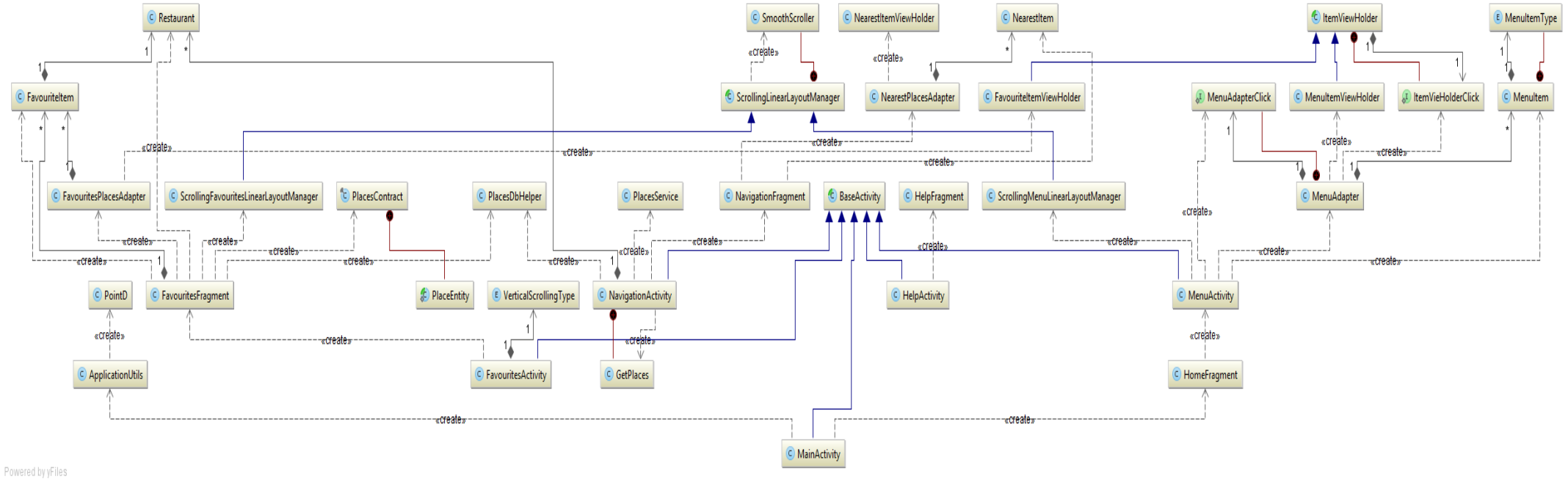
Każda lista prezentowana w aplikacji jest w architekturze *MVVM*, to znaczy każdy element listy posiada dane informacyjne, np. *NearestItem* przechowuje informacje jedynie o nazwie

restauracji i odległości do niej. Widokiem przetrzymującym pojedynczy *NearestItem* jest *NearestItemViewHolder* posiadający dwa *TextView* oraz *ImageView*. Komponentem łączącym dane z modelu i widoku jest zawsze *Adapter*, w przypadku listy najbliżej położonych restauracji jest to *NearestPlacesAdapter*. Dla każdej listy elementów *adaptery* są własnością *RecycleView*, który rozumiany jest jako lista. Wszystkie listy posiadają własnych *LayoutManager'ów*, dzięki którym przechodzenie z jednego elementu do drugiego odbywa się przy pomocy ruchów głowy użytkownika.

Połączenie z *API Google Maps* wykonywane jest jedynie z modułu *navigation*. Tworzony jest własny *klient http* w celu połączenia się z adresem *https://maps.googleapis.com/maps/api/place/search/json* z parametrami: unikalny klucz aplikacji, obecna lokalizacja, typ wyszukiwanych miejsc oraz maksymalny promień wyszukiwania. Dane otrzymane w postaci *JSON* rzutowane są na komponent typu *Restaurant*. Informacje przechowywane w tym komponencie to identyfikator restauracji, nazwa restauracji oraz jej lokalizacja geograficzna.

Dane dotyczące listy ulubionych restauracji przechowywane są w lokalnej bazie danych na urządzeniu. Dostęp do bazy – wczytywanie oraz zapisywanie danych – wykonywane jest w *NavigationFragment* oraz *FavouritesFragment*. System zabezpiecza się przed sytuacją dodania dwóch identycznych miejsc do listy ulubionych. W takim przypadku aktualizowana jest data dodania rekordu. Każdy rekord przechowuje następujące informacje: identyfikator miejsca pobrany z *API Google Maps*, nazwa lokalu, data ostatniego dodania do listy oraz współrzędne geograficzne. Wszystkie komponenty potrzebne do łączenia się z bazą danych znajdują się w module *common* w pakiecie *database*.

Komponentem dostępnym z każdego poziomu aplikacji odpowiadającym za wykonywanie statycznych obliczeń jest *ApplicationUtils*. Wylicza on odległości między współrzędnymi geograficznymi, zwracając wynik w sposób przyjazny dla użytkownika. Dodatkowo każda lokalizacja geograficzna prezentowana jest w postaci adresu zrozumiałego dla użytkownika.



Rysunek 19. Schemat modelu dziedziny
Źródło: opracowanie własne

4.2 Opis klas

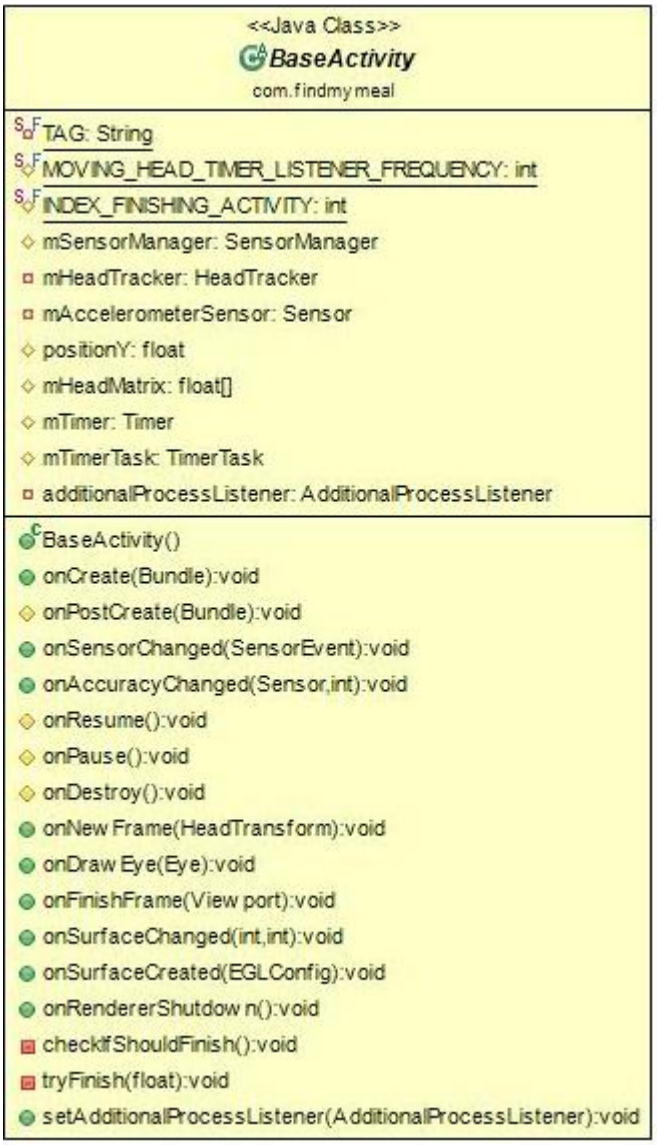
4.2.1 Klasa *BaseActivity*

Dziedziczy po klasie *CardboardActivity*, implementuje interfejsy *SensorEventListener* oraz *Cardboard.StereoRender*. Tę klasę dziedziczą główne aktywności aplikacji – klasy *FavouritesActivity*, *NavigationActivity*, *MenuActivity* oraz *HelpActivity*.

Interfejs *SensorEventListener* wykorzystywany jest do otrzymywania informacji o bieżącym położeniu telefonu w przestrzeni.

Interfejs *Cardboard.StereoRender* służy do wykonywania operacji na macierzach kamery. Dzięki niemu można określić, w którą stronę skierowana jest kamera urządzenia.

Zaimplementowane w ramach tej klasy funkcje pozwalają użytkownikom na nawigację wewnątrz aplikacji zgodną ze standardami innych aplikacji kompatybilnych z *Google Cardboard*. Klasa ta odpowiada za rozpoznawanie ruchu dedykowanego wyłączaniu aplikacji lub ruchu przypisanego do cofania do poprzedniego widoku. Do tego celu wykorzystywana jest instancja klasy *SensorManager* pobierana z serwisu udostępnianego przez system *Android*.

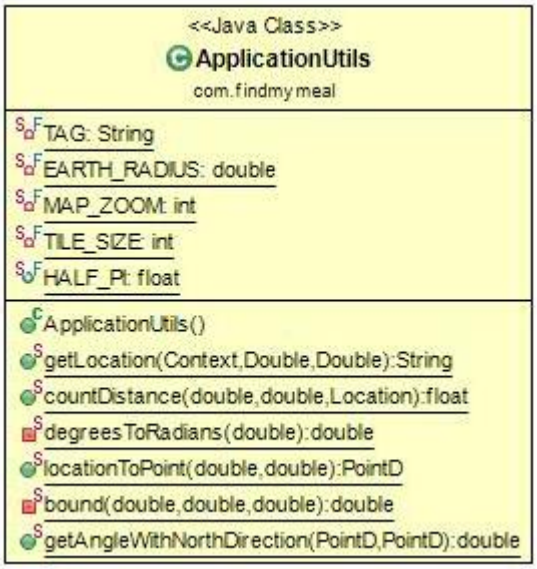


Rysunek 20. Klasa BaseActivity
Źródło: opracowanie własne

4.2.2 Klasa *ApplicationUtils*

Klasa ta służy do translacji lokalizacji geograficznych (domyślnych formatów *Google* do przetrzymywania informacji o położeniu geograficznym danego miejsca) na adres zawierający ulicę, numer budynku, miasto i kraj, w którym znajduje się dana lokalizacja.

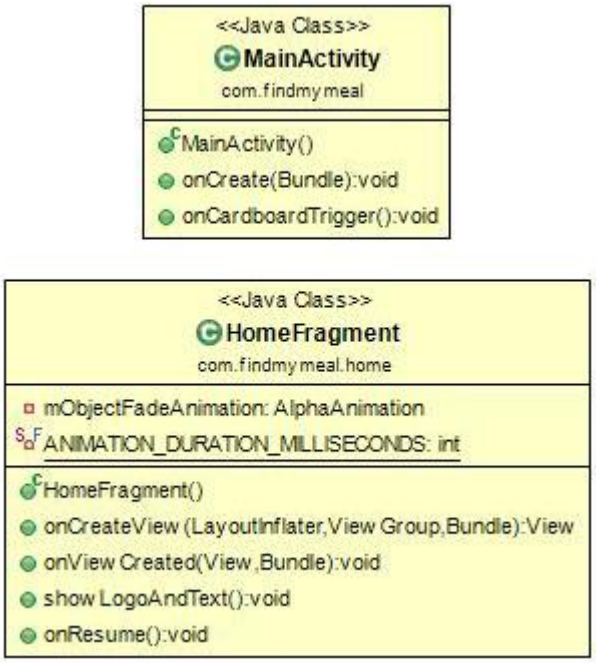
Dodatkowo oblicza odległości między dwoma lokalizacjami geograficznymi i prezentuje wynik w sposób zrozumiały dla użytkownika. Zmienia lokalizację (szerokość i długość geograficzną) na punkt płaszczyzny.



Rysunek 21. Klasa ApplicationUtils
Źródło: opracowanie własne

4.2.3 Klasa *MainActivity*

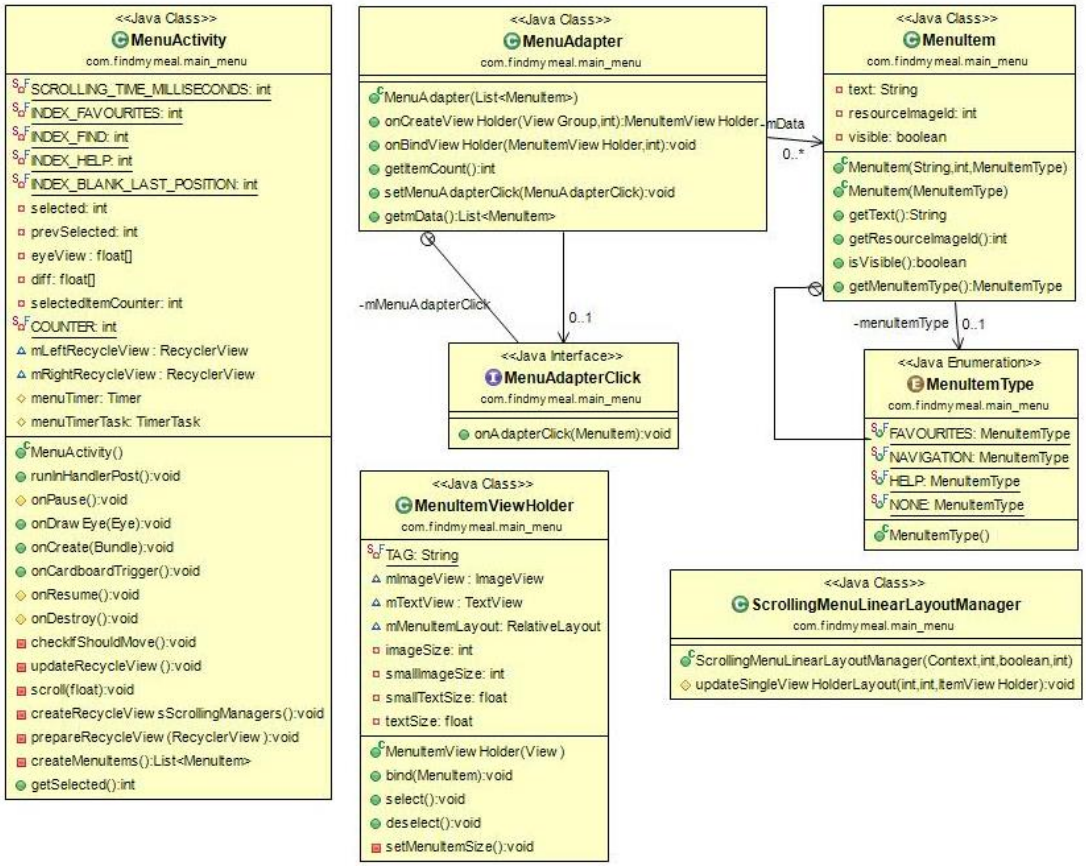
Dziedziczy po klasie *BaseActivity*. Klasa ta jest ustawiona jako *launcher* (uruchamiana jest jako pierwsza po starcie programu). Odpowiada za wyświetlenie logo aplikacji i jego animacji oraz za przejście do menu głównego – klasy *MenuActivity*. *Fragmentem* przetrzymującym widok *MainActivity* jest *HomeFragment*.



Rysunek 22. Klasa MainActivity oraz klasa HomeFragment
Źródło: opracowanie własne

4.2.4 Klasa *MenuActivity*

Głównym zadaniem klasy *MenuActivity* jest utworzenie i wyświetlenie menu głównego aplikacji. Pozycje menu przetrzymywane są wewnątrz obiektu klasy *MenuAdapter* w postaci listy. Dane każdego elementu menu przechowywane są w postaci instancji klas *MenuItem*. Widok każdego elementu listy przechowywany jest w instancji klasy *MenuItemViewHolder*. Elementy menu przenoszą kolejno do *FavouritesActivity*, *NavigationActivity* oraz *HelpActivity*. Wybór pozycji z menu odbywa się za pomocą użycia przycisku magnetycznego. Dzięki dziedziczeniu po klasie *BaseActivity* klasa *MenuActivity* nasłuchuje zdarzenia wciśnięcia tego przycisku. Zmiana pozycji z menu na inną odbywa się poprzez obsłużenie zdarzenia zmiany położenia urządzenia w przestrzeni. Do przełączania między pozycjami menu wykorzystywana jest klasa *ScrollingMenuLinearLayoutManager*.



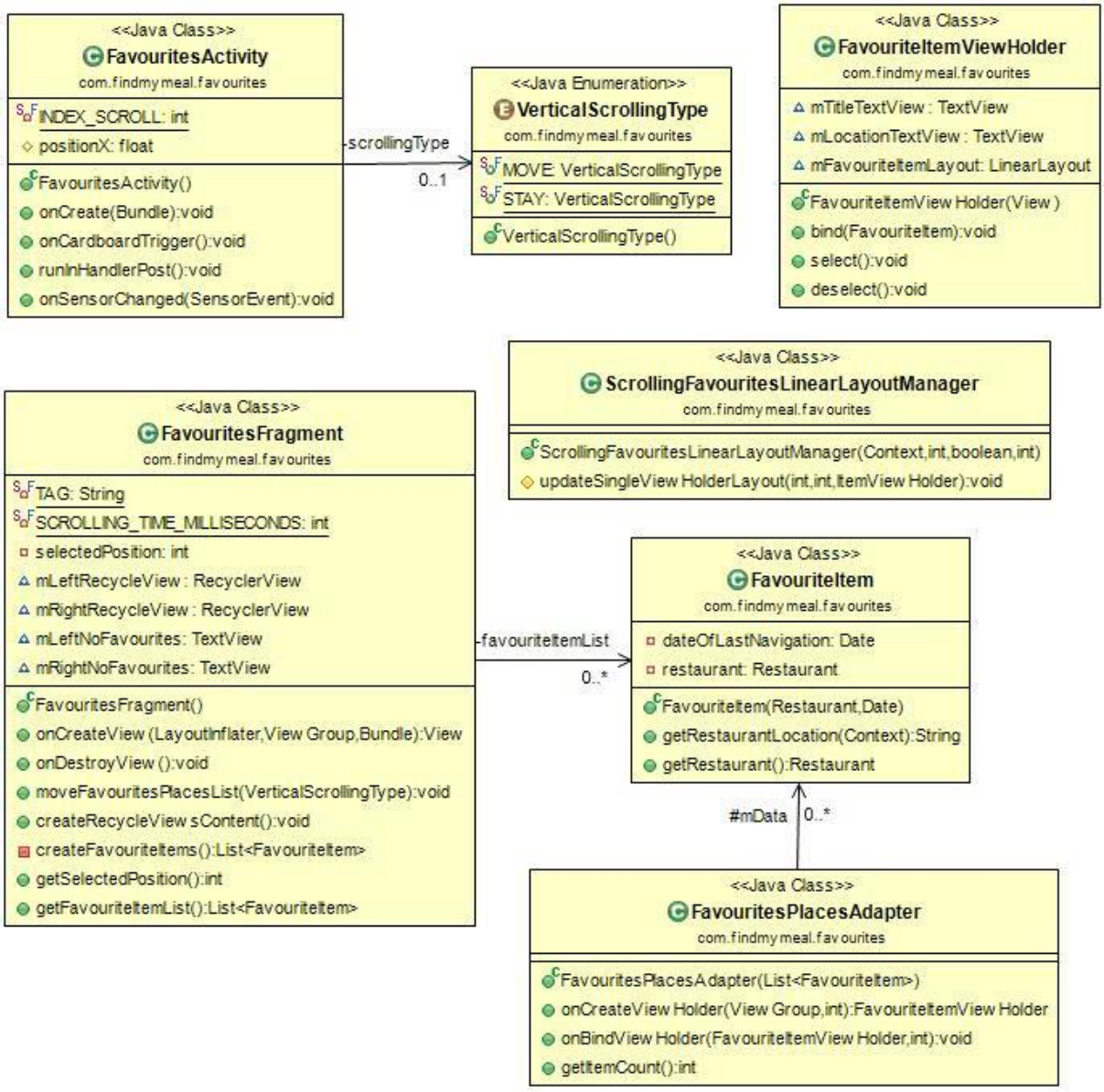
Rysunek 23. Klasa MenuActivity wraz z komponentami
Źródło: opracowanie własne

4.2.5 Klasa *FavouritesActivity*

Dziedziczy po klasie *BaseActivity*. Podczas wywołania konstruktora obiektu dokonywana jest inicjalizacja połączenia z lokalną bazą danych. Jeżeli połącznie zostanie nawiązane pomyślnie, z bazy tej pobierane są wszystkie istniejące rekordy zawierające informacje o ulubionych miejscach. Każdy rekord rzutowany jest na obiekt klasy *FavouriteItem*. Widokiem danych instancji klasy *FavouriteItem* jest instancja klasy *FavouriteItemViewHolder*. Model ulubionego miejsca *bind'owany* jest z odpowiadającym mu widokiem w instancji klasy *FavouritesPlacesAdapter*. Widok listy ulubionych restauracji przechowywany jest we fragmencie *FavouritesFragment*.

Poruszanie się między elementami listy obsługiwane jest poprzez instancję klasy *ScrollingFavouritesLinearLayoutManager* przyjmującym parametr typu *VerticalScrollingType* informujący o podświetleniu następnego elementu na liście, bądź pozostaniu na obecnie zaznaczonym.

W momencie, w którym użytkownik wykona operację naciśnięcia przycisku magnetycznego rozpoczęta zostaje nawigacja do wybranego miejsca przy wykorzystaniu *NavigationActivity*. Element, który był podświetlony na liście ulubionych miejsc zostaje obrany za docelowy punkt nawigacji.



Rysunek 24. Klasa FavouritesActivity wraz z komponentami
Źródło: opracowanie własne

4.2.6 Klasa *NavigationActivity*

Dziedziczy po *BaseActivity*, implementuje *GoogleApiClient.ConnectionCallbacks*, *GoogleApiClient.OnConnectionFailedListener*
oraz *com.google.android.gms.location.LocationListener*. Klasą odpowiedzialną za przetrzymywanie widoku jest *NavigationFragment*. Dodatkowo klasa *NavigationActivity* nasłuchuje zmianę pola magnetycznego i pobiera dane otrzymane z sensora, wyświetlając obraz kompasu w odpowiednim kierunku, jak również wyznacza kąt obrotu strzałek wskazujących kierunki najbliższych lokali.

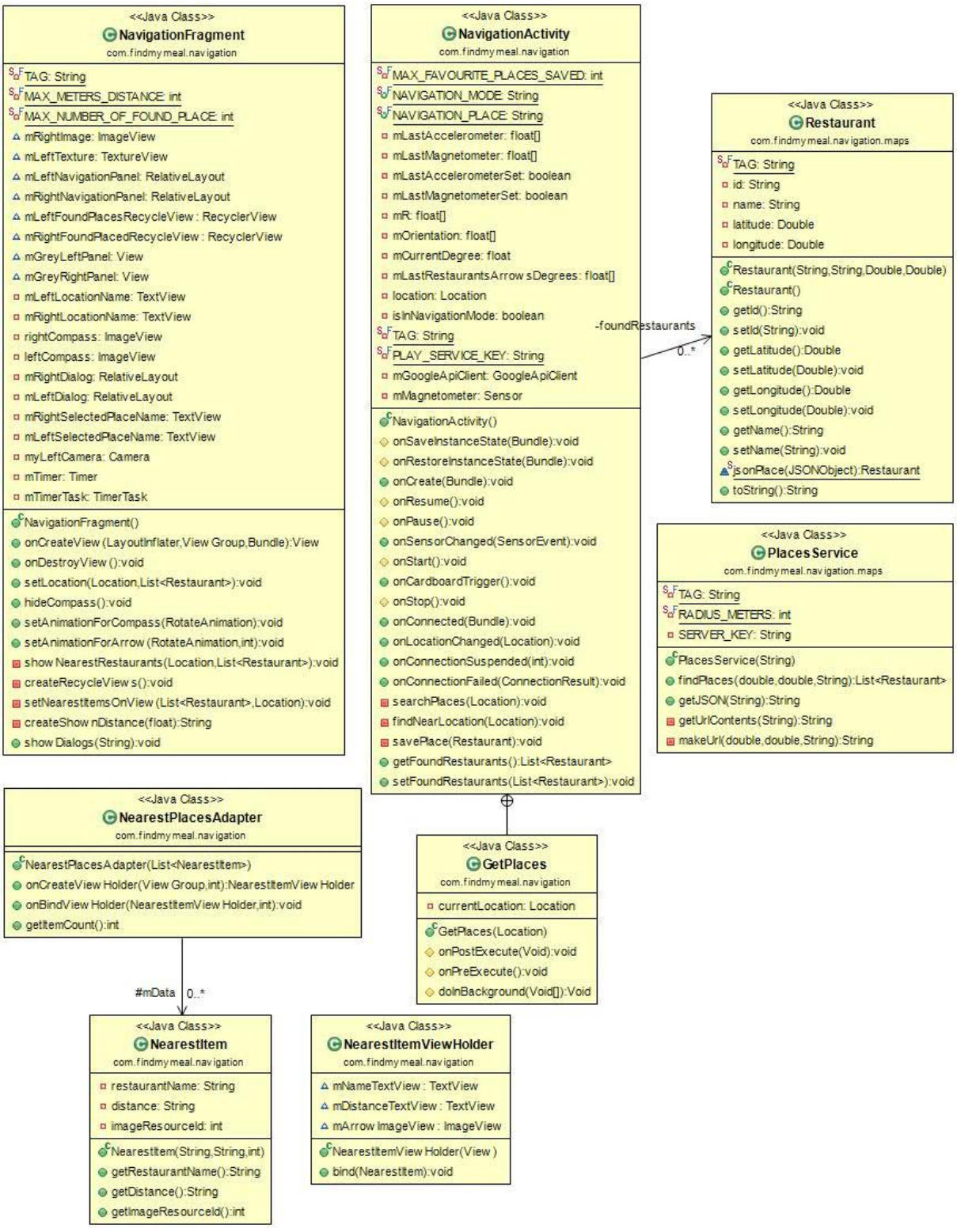
Dzięki implementacji *GoogleApiClient.ConnectionCallbacks* możliwe jest wystawienie klienta *Google API* oraz wykonywanie zapytań do *Google API Web Services*.

Interfejs *com.google.android.gms.location.LocationListener* umożliwia przechwytywanie zamian lokalizacji w momencie przemieszczania się użytkownika, bądź przy pozytywnym połączeniu z *Google API Web Services*.

W momencie zmiany lokalizacji (z dokładnością do około 10 metrów) wykonywane jest nowe zapytanie o najbliższe restauracje do *Google API Web Services*. Zawsze zwracana jest lista zawierająca maksymalnie 20 lokali w odległości do 3 kilometrów od obecnej lokalizacji. Zwracana lista jest już posortowana ze względu na trafność wyszukiwania. Każdy element listy przechowuje swoje dane w instancji klasy *NearestItem*, a jego widok przechowywany jest w instancji klasy *NearestItemViewHolder*. Obie listy najbliższych lokali wyświetlane na prawej i lewej stronie ekranu urządzenia posiadają instancje klasy *NearestPlacesAdapter* łączące maksymalnie 3 dane pobrane z *Google API Web Services* z prostokątnymi kafelkami pojawiającymi się w lewym górnym rogu ekranu.

Naciśnięcie przycisku magnetycznego powoduje zapisanie lokalu znajdującego się na pierwszej pozycji na liście najbliższych restauracji do listy ulubionych miejsc. Wykonywane jest połączenie z lokalną bazą danych w celu wpisania elementu do tabeli o nazwie *place*.

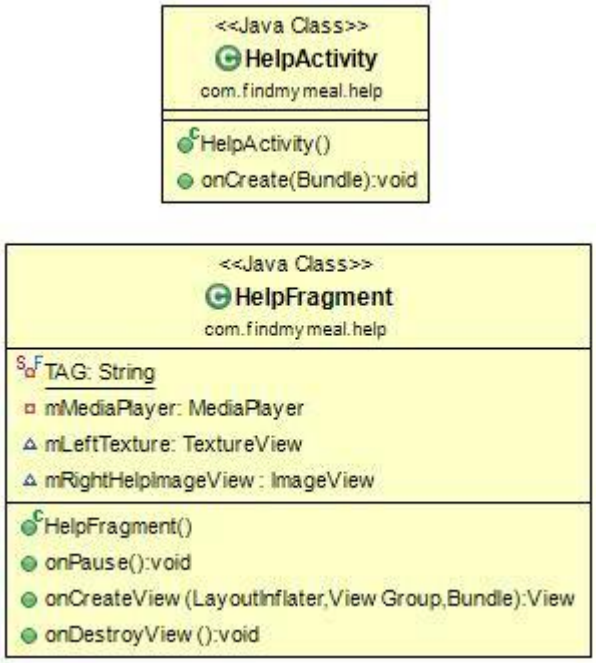
Wszystkie zapytania do *Google API Web Services* wykonywane są asynchronicznie przy pomocy instancji klasy *GetPlaces* będącej *AsyncTask'iem*. W tle tworzony jest serwis *PlacesService* łączący się bezpośrednio przy użyciu protokołu *http* z *Google API Web Services*. Otrzymane dane w postaci *JSON* rzutuje na instancje klas *Restaurant* i przekazuje z powrotem do *NavigationActivity*. Dzięki takiemu rozwiązaniu urządzenie nie zawiesza się podczas wielokrotnych zapytań, a jakiekolwiek niepowodzenie nie skutkuje utratą danych.



Rysunek 25. Klasa NavigationActivity wraz z komponentami
Źródło: opracowanie własne

4.2.7 Klasa *HelpActivity*

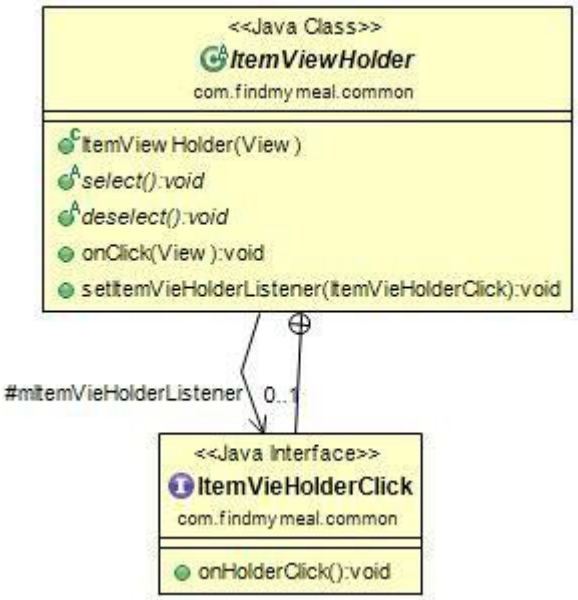
Dziedziczy po *BaseActivity*, jej zadaniem jest poprawne wyświetlenie filmu instruktażowego w sposób jednoczesny na lewej i prawej stronie ekranu urządzenia. Klasą odpowiedzialną za widok *HelpActivity* jest *HelpFragment*. Aby uzyskać efekt jednoczesnego odtwarzania wideo na dwóch widokach w tym samym czasie wykorzystano instancję klasy *TextureView*, która umożliwia przechwytywanie obrazu z filmu w momencie aktualizacji widoku wideo.



Rysunek 26. Klasa HelpActivity oraz HelpFragment
Źródło: opracowanie własne

4.2.8 Klasa *ItemViewHolder*

Pozwala ona na wybieranie lub odznaczanie danego elementu na widoku listy przy pomocy interfejsu *ItemViewHolderClick*. Jest to bazowa klasa wszystkich widoków list wykorzystanych w aplikacji.



Rysunek 27. Klasa ItemViewHolder oraz interface ItemViewHolderClick
Źródło: opracowanie własne

Spełnienie założeń zasad SOLID:

S (single responsibility principle) - zadaniem tej klasy jest udostępnienie klasom ją dziedziczącym zaplecza pozwalającego na dokonywanie operacji zaznaczania bądź odznaczania wybranego elementu listy.

O (open/closed principle) - klasa ta jest zamknięta, ale otwarta na dalsze rozszerzenia. Dokonują ich między innymi klasy *FavouriteItemViewHolder* i *MenuItemViewHolder*.

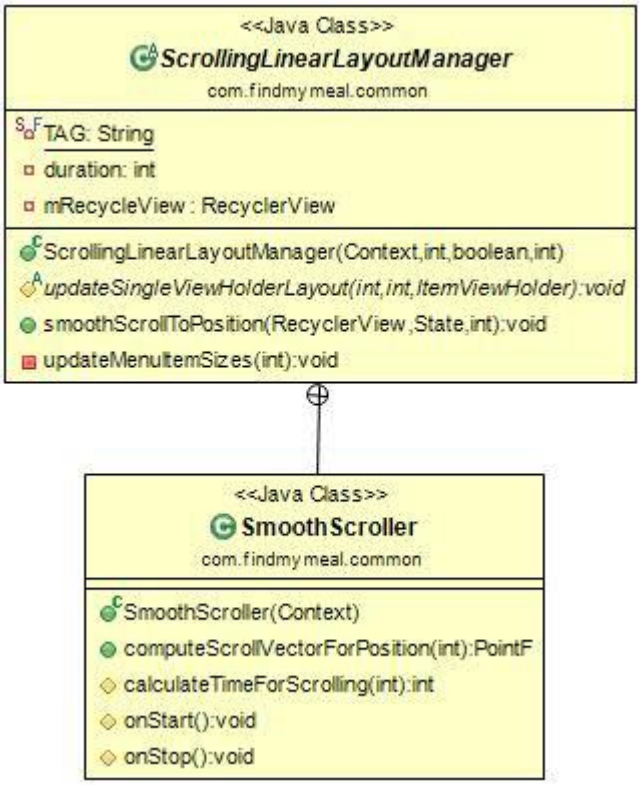
L (Liskov subsitution principle) - obiekty klas dziedziczących po tej klasie można z dużą łatwością między sobą wymieniać w przypadku gdy dochodzi do konieczności wymiany np. sposobu zaznaczania pozycji w menu.

I (Interface segregation principle) - pomimo tego, że klasa nie implementuje żadnego interfejsu, rozszerza klasę *ViewHolder* z *package'u RacycleView*. Nie dochodzi tu więc do dziedziczenia np. całej, ogromnej klasy ogólnej widoku.

D (Dependency inverios principle) - klasa ta nie uściśla w jaki sposób dochodzi do wybrania danej pozycji w menu lub przejścia do kolejnej. Obsługuje jedynie operacje zaznaczania i odznaczania.

4.2.9 Klasa *ScrollingLinearLayoutManager*

Jej zadaniem jest umożliwienie poruszania się po liście w sposób płynny, z określoną prędkością do wcześniej określonej pozycji. Po zakończeniu etapu przesuwania wywoływana jest metoda aktualizacji widoku elementów z poziomu instancji klasy *SmoothScroler*. Skutkiem jest zaznaczenie bądź odznaczenie elementów listy.



Rysunek 28. Klasa ScrollingLinearLayoutManager oraz klasa SmoothScroller
Źródło: opracowanie własne

Spełnienie założeń zasad SOLID:

S (single responsibility principle) - klasa ta odpowiada za wykonywanie jednej czynności - umożliwienie przemieszczania się po liście w sposób zgodny z *User Experience*.

O (open/closed principle) - klasa ta otwarta jest na rozszerzenia. Dziedziczą po niej i rozszerzają ją klasy *ScrollingFavouritesLinearLayoutManager* oraz *ScrollingMenuLinearLayoutManager* dodając odpowiednie im względem zastosowania funkcjonalności.

L (Liskov subsitution principle) - obiekty klas, które dziedziczą po tej klasie mogą być łatwo między sobą wymieniane w momencie jeżeli dojdzie np. do potrzeby zmiany zachowania obsługi menu z wertykalnego na horyzontalne.

I (Interface segregation principle) - klasa ta co prawda nie implementuje żadnego interfejsu, ale dziedziczy natomiast po klasie *LinearLayoutManager*. Istnieje wiele klas typu *LayoutManager* dedykowanych odpowiednim widokom.

D (Dependency inverios principle) - klasa ta nie bazuje na żadnym uściśleniu (odpowiada tylko i wyłącznie za obsługę menu wertykalnego).

4.3 Biblioteki

Przy tworzeniu aplikacji *FindMyMeal* wykorzystane zostały zewnętrzne biblioteki.

4.3.1 Cardboard SDK for Android v0.6.0

Biblioteka *Cardboard* została dostarczona przez firmę *Google* w postaci skompresowanej biblioteki typu *jar* kompatybilnej z systemem *Android API 19* i wyższe. Cała biblioteka jest nieodłączną częścią projektu. Stosowana jest do integracji z akcelerometrami. Dzięki zaproponowanemu rozwiązaniu przez architekturę biblioteki obsługa przycisku magnetycznego i implementacja zmiany położenia urządzenia w przestrzeni jest możliwa w sposób różnorodny dla każdego widoku. Dane otrzymywane podczas zmiany położenia urządzenia w przestrzeni są aktualizowane w sposób ciągły w odstępach nie większych niż 10 milisekund. Informacją przetwarzaną przez aplikację *FindMyMeal* otrzymywaną przy

pomocy biblioteki *Carboard* jest macierz o wymiarach 4 na 4 zawierająca dane o wartościach z przedziału od -10 do 10, przy czym ostatnia kolumna macierzy jest zawsze w postaci wektora [1,0,0,0]. Dzięki obserwacji zmiany wartości macierzy z dokładnością do 0.1 możliwe jest wykonywane odpowiednich akcji będących reakcją na zmianę położenia telefonu, bądź użycia przycisku magnetycznego. Przy tworzeniu widoków zrezygnowano z zaproponowanego rozwiązania biblioteki *Cardboard*, proponującej zastosowanie biblioteki *OpenGL* i dynamiczne tworzenie widoków, na rzecz statycznych widoków zapisanych w plikach typu *XML* w *Resource'ach* aplikacji. Zaletą takiego rozwiązania jest generacja szablonu widoków na poziomie budowania aplikacji. Dostęp do utworzonego widoku jest możliwy jedynie z poziomu posiadającej go aktywności. Dane dynamiczne *bind'owane* są do zapamiętanych widoków. Aktualizacja tych danych nie powoduje utraty utworzonych widoków. Podział ekranu na dwie części w celu uzyskania przestrzennego widoku wykonywany jest przy pomocy statycznych widoków, zamiast zaproponowanego przez bibliotekę *CarboardView*. Takie rozwiązanie pozwala na całkowite stosowanie języka *Java* w projekcie, wszystkie komponenty wyświetlane na ekranie urządzenia są standardowymi obiektami widoku systemu *Android*.

4.3.2 ButterKnife v7.0.1

Jest to biblioteka udostępniana przez użytkownika JakeWharton w serwisie *GitHub*. Biblioteka wykorzystywana jest w celu *bind'owania* statycznych widoków w zawierających je aktywnościach, bądź fragmentach. Każdy widok *bindowany* jest podczas metody *onCreateView/onCreate* odpowiednio we *fragmencie/aktywności* generując w ten sposób widok ekranu na nowo. Podczas wywoływania metod *onDestroyView/onDestroy* z poziomu odpowiednio *fragmentu/aktywności* widoki usuwane są z tymczasowej pamięci programu dzięki bibliotece *ButterKnife* i metodzie *unbind*. Zaletą stosowania tej biblioteki jest uporządkowanie wykorzystywanych widoków pochodzących bezpośrednio z biblioteki *Android* oraz jednokrotne *bindowanie* widoków podczas cyklu życia *aktywności*, bądź *fragmentu*. Dodatkowo biblioteka wprowadza przejrzystość kodu, skutkując utratą zakładanej prywatności *zbindowanych* widoków na rzecz dostępu do nich typu *Package*.

4.3.3 AppCompat v7:23.1.0

Jest to biblioteka dostępna do zainstalowania z poziomu *Android SDK Manager* i jest składnikiem rozszerzającym podstawową wersję *Android SDK*. Umożliwia stosowanie rozbudowanych widoków dla systemu *Android* z *API 19* i wyższym. Pomimo możliwości stosowania zoptymalizowanych klas aktywności *AppCompatActivity* oraz klasy *Fragment* pochodzącej również z biblioteki *AppCompat* architektura aplikacji *FindMyMeal* uniemożliwiła stosowanie tych obiektów ze względu na dziedziczenie po aktywności pochodzącej z biblioteki *Cardboard*. Zastosowano jednak bibliotekę w celu reprezentacji i obsługi prezentowanych list. Wykorzystano *RecycleView* do całkowitej reprezentacji menu, listy ulubionych miejsc oraz listy najbliżej położonych lokali. Obsługa przesyłania danych z serwisów, bądź baz danych do list odbywa się przy pomocy klas *Adapter* oraz *ViewHolder*.

4.3.4 Google Play – Services v8.1.0

Biblioteka ta wykorzystywana jest do integracji z serwisami dostarczanymi przez firmę *Google*. Główną funkcjonalnością wykorzystywaną w aplikacji *FindMyMeal* jest pobieranie informacji na temat obecnej lokalizacji w postaci współrzędnych geograficznych oraz bieżącego adresu. W celu umożliwienia pobierania danych udostępnianych przez *Google* należało zarejestrować aplikację na stronie *console.developers.google.com*. Połączenie z serwisami *Google* odbywa się poprzez utworzenie klienta *GoogleApiClient* umożliwiającego nasłuchiwanie na pozytywne połączenie się z serwisami, bądź też nagłe zerwania połączenia. Dzięki wbudowanej bibliotece *GMS* możliwe jest wychwytywanie zmiany pozycji telefonu w odstępach co najmniej 10 metrów. Tak uzyskiwane dane są niezbędne do prawidłowego działania nawigacji w aplikacji. Połączenie z *Google Places API Web Service* wykonywane jest niezależnie od biblioteki *Google Play Services* w celu przyspieszenia pobierania danych z usługi. Wykorzystywany jest w tym celu własny serwis łączący się przy pomocy protokołu *http* bezpośrednio z usługą. Zarejestrowana aplikacja jest przeznaczona dla niewielkiej grupy użytkowników ze względu na ograniczenia narzucone z góry przez firmę *Google* w postaci maksymalnie 1000 odpowiedzi z serwisów do wszystkich zainstalowanych aplikacji na urządzeniach.

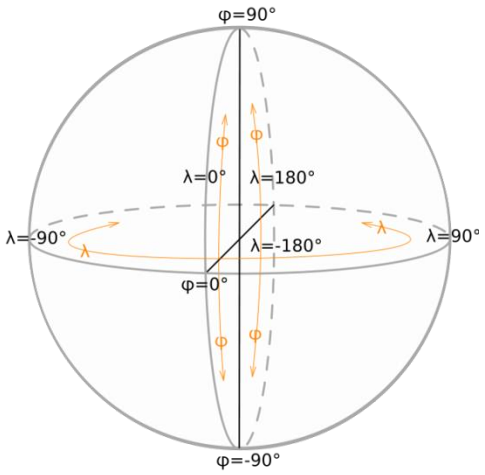
4.4 Algorytmy

4.4.1 Przekształcenia współrzędnych geograficznych

W aplikacji zastosowano dwa algorytmy służące do przekształcania współrzędnych geograficznych - szerokości geograficznej (ang. *latitude*) oznaczanej przez φ i długości geograficznej (ang. *longitude*) oznaczanej przez λ. Obie te wartości mierzone są w stopniach, minutach i sekundach kątowych. Początkiem układu współrzędnych geograficznych jest przecięcie południka zerowego z równikiem.

Kąt λ przyjmuje wartości od -180º do 180º i jest on zawarty między półpłaszczyzną południka przechodzącego przez dany punkt i półpłaszczyzną południka zerowego. Wartości ujemne liczone są w kierunku zachodnim natomiast dodatnie w kierunku wschodnim.

Kąt φ przyjmuje wartości od -90º do 90º i zawarty jest między kierunkiem normalnej do powierzchni Ziemi (od jej jądra) a płaszczyzną równika ziemskiego. Wartości dodatnie otrzymywane są w kierunku północnym, ujemne natomiast w kierunku południowym.



Rysunek 29. Sfera przedstawiająca wartości współrzędnych geograficznych
Źródło: https://pl.wikipedia.org/wiki/Plik:Geographic_coordinates_sphere.svg

4.4.1.1 Obliczanie kierunku nawigowania

4.4.1.1.1 Cel algorytmu

Celem algorytmu jest obliczenie kierunku, w którym ma podążać użytkownik aby osiągnąć miejsce docelowe.

4.4.1.1.2 Opis ogólny algorytmu

W celu ułatwienia zrozumienia działania algorytmu został on przedstawiony w sposób opisowy. Podczas działania naszej aplikacji algorytm ten wykonywany jest wielokrotnie dla wielu restauracji w pętli. Zastosowane w programie z tego tytułu optymalizacje zostały pominięte.

Argumenty wejściowe:

- *bieżąca lokalizacja użytkownika* - wielkości kątów współrzędnych geograficznych,
- *lokalizacja docelowej restauracji* - wielkości kątów współrzędnych geograficznych,
- *miara kąta odchylenia urządzenia od kierunku północnego* - w radianach, pochodząca z sensora magnetycznego.

Zwracana wartość:

- *miara kąta między punktem docelowym a kierunkiem północnym* - w radianach.

Kolejne kroki algorytmu:

1. Dla wejściowych obiektów lokalizacji wykonywane jest przekształcenie ze współrzędnych geograficznych do współrzędnych płaszczyzny i zapisanie otrzymanych wartości x i y w postaci punktu. Wykorzystywane do tego celu jest rozwiązanie problemu *odwzorowania walcowego równokątnego*. Wielkości x i y obliczane są według wzorów:

$$x = R(\lambda - \lambda^o)$$

$$y = R \ln\left[\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right]$$

   Wielkość R jest stałą skalowania mapy. Do naszych obliczeń przyjmujemy $R = 256$ z *8-ktornym przybliżeniem* (tj. nasz obszar odwzorowywanej mapy zostaje podzielony na 2^8 *kawałków*). Obliczenia te w swej postaci uwzględniają krzywiznę Ziemi.

2. W drugim kroku dokonywane jest obliczenie kąta miedzy dwoma obliczonymi wcześniej wartościami punktów (tj. dla bieżącej lokalizacji i lokalizacji docelowej). Punkty traktowane są jako dwa wektory. W celu ułatwienia i znormalizowania obliczeń używana jest funkcja *atan2*, której jako parametr zostaje podany wektor różnicy dwóch wcześniej wymienionych wektorów. Otrzymany wynik w radianach mówi nam jak duży jest kąt między wektorem różnicy a *osią X*.
3. Aby obliczony kąt był zrozumiały dla użytkownika musi on odnosić się do kierunku północnego - *osi Y*. Dlatego też do obliczonego kąta należy dodać miarę $\pi/2$, co odpowiada obrotowi o 90^o przeciwnie do ruchu wskazówek zegara.
4. Ostatnim krokiem algorytmu jest dodanie obliczonej wartości w radianach do informacji o kierunku położenia telefonu otrzymanego z sensora magnetycznego.

Do implementacji wykorzystywane są następujące klasy pomocnicze:

- klasa *Point*,
- klasa *Location*.

4.4.1.2 Obliczanie odległości między dwiema współrzędnymi geograficznymi

4.4.1.2.1 Cel algorytmu

Celem algorytmu jest obliczenie odległości między dwiema współrzędnymi geograficznymi uwzględniając krzywiznę powierzchni Ziemi.

4.4.1.2.2 Opis algorytmu

Dane wejściowe:

- *bieżąca lokalizacja* - wielkości kątów współrzędnych geograficznych,
- *lokalizacja docelowa* - wielkości kątów współrzędnych geograficznych,
- *długość promienia Ziemi* - zależny od bieżącego położenia, wyrażony w jednostce metrycznej *układu SI* (np. metry lub kilometry).

Zwracana wartość:

- *odległość między dwiema lokalizacjami* - wyrażona w określonej wcześniej jednostce metrycznej *układu SI* dla promienia Ziemi.

Pseudokod algorytmu:

```
obliczOdleglosc(biezacaLokalizacja, docelowaLokalizacja, promienZiemi) {
  roznicaLatitude = docelowaLokalizacja.Latitude - biezacaLokalizacja.Latitude
  roznicaLongitude =  biezacaLokalizacja.Longitude - biezacaLokalizacja.Longitude

  radianyLatitude = stopnieDoRadianow(roznicaLatitude);
  radianyLongitude = stopniedoRadianow(roznicaLongitude);

  a = cos(stopnieDoRadianow(docelowaLokalizacja.Latitude))
  a *= cos(stopnieDoRadianow(biezacaLokalizacja.Latitude))
  a *= sin(radianyLongitude /2)^2
  a += sin(radianyLatitude/2)^2

  c = 2 * atan2(√a, √(1-a));
  zwróć promienZiemi * c;
}

stopnieDoRadianow(stopnie) {
  zwróć stopnie * (π/180)
}
```

4.4.2 Obliczanie wielkości ikon w menu głównym

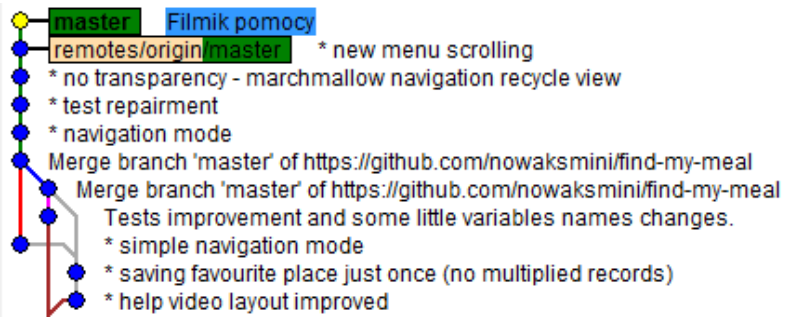
Przedstawiony poniżej pseudokod w sposób ogólny opisuje sekwencję kodu, który wykorzystywany jest w momencie dodawania kolejnych elementów menu głównego do listy. Lista ta następnie jest wykorzystywana przez widok menu. Algorytm ten wyznacza w sposób dynamiczny szerokość i wysokość elementów menu w celu uzyskania najlepszej jakości widoku prezentowanego na ekranie urządzenia.

```
width:=szerokość urządzenia w dp/6
height:=wysokość urządzenia w dp
imgSize:=min(width,height)
smallImgSize:=50%*imgSize
pxImgSize:=imgSize w pikselach
pxSmallImgSize:=smallImgSize w pikselach
szerokość kontenera ikony:=imgSize
wysokość kontenera ikony:=imgSize
textSize:=pxImgSize/20
smallTextSize:=pxSmallImgSize/20
jeżeli width < szerokość kontenera tekstu
     szerokość kontenera tekstu:=width
jeżeli height < wysokość kontenera tekstu
     wysokość kontenera tekstu:=height
ustaw szerokość i wysokość obiektu menu na width i height
```

5 Podsumowanie

5.1 Wytwarzanie oprogramowania – wersjonowanie

Do celów wersjonowania kodu naszej aplikacji wykorzystaliśmy program *Git*. Umożliwia on współpracę wielu użytkowników jednocześnie, bez centralnego punktu referencyjnego (czyli np. serwera, z którym łączyli by się wszyscy użytkownicy). Dzięki temu znacząco zredukowaliśmy szansę na utracenie całości historii zmian w projekcie. Dodatkowo zyskaliśmy łatwy w obsłudze sposób zarządzania różnymi *branchami* (równoległymi pracami nad różnymi, często nie związanymi ze sobą funkcjonalnościami).



Rysunek 10. Przykład pracy na kilku, równolegle rozwijanych branchach
Źródło: opracowanie własne

Nasze *repozytorium* postanowiliśmy założyć na popularnym serwisie *GitHub*. Każdy z użytkowników dodatkowo sklonował je do pamięci swojego komputera. Serwis *GitHub* udostępnia łatwy w obsłudze zestaw narzędzi umożliwiający kontrolowanie zmian zachodzących w trakcie rozwijania aplikacji.

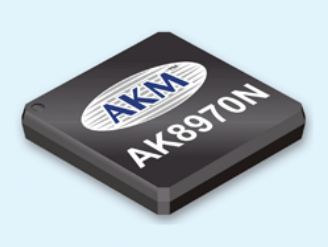
Dodatkowym atutem programu *Git* jest jego synchronizacja z *IDE AndroidStudio* co pozwoliło nam na śledzenie w czasie rzeczywistym zmian w kodzie dokonywanych przez innych użytkowników.

5.2 Problemy nierozwiązane

5.2.1 Zakłamywanie odczytów magnetometru (sensora magnetycznego)

Magnetometr jest przyrządem do pomiaru wielkości, kierunku oraz zmian pola magnetycznego. W naszej aplikacji wykorzystywany jest on do wskazywania kierunku północnego kompasu, a ten jako kierunek referencyjny w module nawigacji.

Istnieją 3 typy magnetometrów wykorzystywanych przez nowoczesne smartfony i są to *Hallotron*, *Anizotropowy magnetoopór* (ang. *Anisotropic Magnetoresistive*, w skrócie *AMR*) oraz *Gigantyczny magnetoopór* (ang. *Giant Magnetoresistive*, w skrócie *GMR*). Dane odczytywane przez te urządzenia mogą być zapisywane w postaci punktów przestrzeni dwu (współrzędnych x, y) lub trzy wymiarowej (współrzędnych x, y, z).



Rysunek 11. Przykład modułu kompasu AK8970N odczytującego dane w trzech wymiarach firmy AKM Semiconductor
Źródło: http://www.electronicproducts.com/Analog_Mixed_Signal_ICs/Electronic_compass_IC_brings_three-axis_measurement_to_handhelds.aspx

Problem pojawia się w momencie, gdy użytkownik przed założeniem modułu *Google Cardboard* źle założył magnesy - tj. neodymowy w środku a ferrytowy na zewnątrz - lub posiada tańszą wersję modułu, gdzie oba magnesy są ferrytowe. Dochodzi wtedy do zaburzenia pola magnetycznego w otoczeniu telefonu. Zjawisko to odbija się negatywnie na odczytach, które nasza aplikacja odbiera z magnetometru. Nie jest to oczywiście niczym zadziwiającym, ponieważ magnes przyłożony do kompasu powoduje, że jego igła zaczyna wskazywać błędny kierunek bądź wręcz obraca się w kółko, nie mogąc przejść do stanu ustalonego.

W tym momencie wszystkie moduły korzystające z odczytów magnetometru - czyli kompas i nawigacja wskazują na ekranie błędne kierunki, co uniemożliwia poprawne ich wykorzystywanie.

Niestety, ponieważ problem ten jest natury czysto fizycznej, nie ma dla niego łatwego programistycznego rozwiązania.

5.2.2 Ograniczona liczba zapytań do *Google Places API Web Service*

Pozyskiwanie listy najbliższych lokali odbywa się przy wykorzystaniu *Google Places API Web Service*, poprzez wysyłanie zapytania do serwisu przy przemieszczeniu się urządzenia o co najmniej 10 metrów. Możliwe jest to dzięki rejestracji aplikacji w serwisie *Google* przeznaczonym dla deweloperów na stronie *https://console.developers.google.com*, przy czym w ciągu 24 godzin liczonych od godziny 0:00 do 23:59 tego samego dnia według czasu *Pacific Time (PT)* możliwe jest pobieranie danych maksymalnie 1000 razy. Zapytania ze wszystkich urządzeń korzystających z aplikacji *FindMyMeal* są sumowane, co może prowadzić do szybkiego wykorzystania limitu. Aby uniknąć tego ograniczenia, należy wykupić odpowiednią ilość zapytań na stronie konsoli deweloperskiej *Google*. Aplikacja używana jednocześnie przez co najwyżej 5 użytkowników powinna mieć wykupione minimum 100 tysięcy zapytań. Aby z aplikacji mogła korzystać nieograniczona liczba użytkowników, należy wykupić możliwość nielimitowanego korzystania z *Google Places API Web Service*.

5.3 Dalsze plany rozwojowe

Po zakończeniu projektu w ramach działań związanych z pracą inżynierską planujemy dalej rozwijać naszą aplikację. Naszym celem jest jej dalsze dopracowywanie tak, a by była w jak największym stopniu kompatybilna z kolejnym produktem - *Google Glass*.

W tym celu zamierzamy dalej rozwijać interfejsy w kierunku rzeczywistości rozszerzonej, planujemy rozszerzyć naszą aplikację o rozwiązania integracyjne z platformami społecznościowymi oraz wzbogacić ją o funkcje sterowania głosowego.

5.3.1 Rozwinięcia interfejsu

Mamy kilka pomysłów, których wprowadzenie w ramach interfejsu naszej aplikacji mogłoby uczynić ją jeszcze bardziej użyteczną i atrakcyjną.

5.3.1.1 Kompas trójwymiarowy

Obecnie w naszej aplikacji kompas generowany jest jako obrazek dwuwymiarowy na płaskiej powierzchni paska bieżącej lokalizacji - patrz rysunek 32.

Rysunek 12. Przykład układu kompasu na pasku bieżącej lokalizacji
Źródło: opracowanie własne

Naszym pomysłem jest jego zamiana na kompas generowany w trzech wymiarach.

5.3.1.2 Interaktywne strzałki nawigacji

W obecnym momencie nasza aplikacja wyświetla informacje o nawigacji w postaci dymków w lewym górnym rogu każdego widoku - patrz rysunek 33.

Rysunek 13. Przykład 3 dymków nawigacji
Źródło: opracowanie własne

Pierwszym z pomysłów jest zamienienie ich na strzałki, które poruszają się po obszarze widoku w kierunku, w którym powinien przemieszczać się użytkownik. Przykład możliwej implementacji został przedstawiony na rysunku 34.

Rysunek 14. Przykład możliwej implementacji poruszających się strzałek
Źródło: opracowanie własne

Drugim pomysłem jest zastosowanie ścieżek, które mają aproksymować kierunek, w którym ma przemieszać się użytkownik - patrz rysunek 35.

Rysunek 15. Przykład możliwej implementacji ścieżki kierunku
Źródło: opracowanie własne

5.3.2 Wyznaczanie dokładnej trasy nawigacji

W chwili obecnej nasza aplikacja obsługuje uproszczony model nawigacji polegający na obliczaniu kierunku między miejscem docelowym i kierunkiem, w który zwrócony jest użytkownik (*rozdział 4.4.1.1 Obliczanie kierunku nawigowania*). Z biegiem czasu mamy zamiar ulepszyć moduł nawigowania do wybranego ulubionego miejsca. Aby zapewnić użytkownikowi dużo większą skuteczność w poruszaniu się chcemy zaimplementować wyznaczanie dokładnej trasy, takie jak przywykliśmy wszyscy używać w popularnych modułach nawigacji *GPS* czy aplikacjach nawigujących. Już teraz powstały dwie konwencje wykonania tego zadania:

1. Pobieranie trasy z *Google Maps API*

   Ten wariant pozwoliłby by nam na zbudowanie nawigacji z interfejsem użytkownika zgodnym z resztą aplikacji. Jest to jednak zadanie bardzo skomplikowane. Naszym wstępnym pomysłem jest podzielenie otrzymanej jako odpowiedź od *Google* trasy na kawałki i nawigowanie do kolejnych wygenerowanych w ten sposób punktów przy pomocy zaimplementowanego w obecnej aplikacji wersji algorytmu.
2. Uruchamianie klienta nawigacji

   Dużo prostszym sposobem jest uruchomienie domyślnego dla urządzenia z systemem *Android* modułu nawigacji. Używany jest w takiej sytuacji klient *Google Maps Directions API*. Jako parametry uruchomieniowe wystarczy podać punkt startowy i końcowy. Minusem jest niezgodność z interfejsem naszej aplikacji.

5.3.3 Konta użytkowników

Możliwość interakcji między użytkownikami aplikacji *FindMyMeal* zapewni wprowadzenie osobistych kont.

5.3.3.1 Logowanie

Po włączeniu aplikacji użytkownik będzie mógł zalogować się przy pomocy konta w serwisie *Facebook* lub *Google* (wymagane jest posiadanie jednej z tych aplikacji na urządzeniu). Możliwe będzie również korzystanie z aplikacji bez logowania się, w tym przypadku nie będą dostępne funkcje dodatkowe. Komunikacja będzie możliwa między użytkownikami, którzy mają siebie w gronie znajomych.

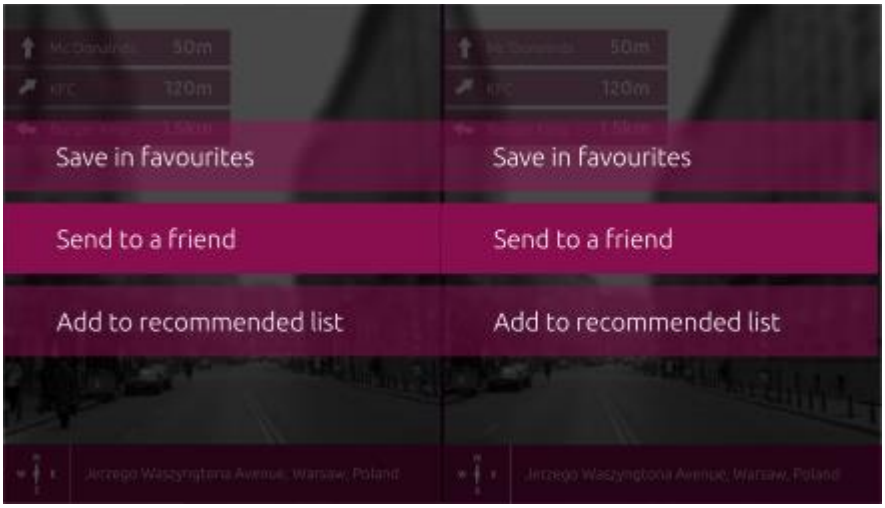
Rysunek 36. Przykład przycisków logowania za pośrednictwem kont aplikacji społecznościowych
Źródło: http://www.makeuseof.com/tag/never-touch-code-filament-io-non-developers-everywhere/

5.3.3.2 Udostępnianie lokalizacji innym użytkownikom

Funkcją dostępną po zalogowaniu będzie udostępnianie wybranych miejsc innym użytkownikom. Możliwe to będzie z poziomu opcji *Find*. Użycie przycisku magnetycznego spowoduje wyświetlenie listy dostępnych opcji – istniejące już dodanie miejsca do ulubionych oraz nowa możliwość – udostępnienie go wybranemu użytkownikowi, który otrzymuje odpowiedni komunikat, po czym może nawigować do otrzymanej lokalizacji.

5.3.3.3 Publiczna lista polecanych miejsc

Dodatkową opcją dostępną z poziomu nawigacji dla zalogowanych użytkowników może być dodanie lokalizacji do listy polecanych, która widoczna jest dla innych użytkowników. Aby w pełni korzystać z tej funkcjonalności dostępna będzie opcja z menu głównego umożliwiająca przeglądanie udostępnianych miejsc przez innych użytkowników i nawigowanie do nich.



Rysunek 37. Przykład listy dostępnych opcji z poziomu nawigacji
Źródło: opracowanie własne

5.3.4 Sterowanie głosowe

Obecnie coraz większe uznanie wśród użytkowników urządzeń mobilnych zyskują rozwiązania oparte o komendy głosowe i rozpoznawanie mowy. Istnieje ku temu wiele powodów - jednymi z nich są np. wygoda użytkowania urządzenia w trakcie ruchu lub możliwość szybkiego pisania długich wiadomości bez potrzeby używania niewielkich rozmiarów klawiatury.

Ponieważ telefon w trakcie działania naszej aplikacji umieszczony jest w module *Cardboard*, użytkownik nie ma dostępu do jego ekranu. W przyszłości, rozwiązania oparte o *Google Glass* będą całkowicie pozbawione wyświetlacza dotykowego. Dlatego też rozważamy możliwość wykorzystania komend głosowych pozwalających na sterowanie naszą aplikacją.

Istnieje wiele możliwości realizacji tej idei. Najbardziej obiecującym okazuje się być wciąż jeszcze rozwijany *Web Speech API* od *Google*.

5.4 Zakończenie

Mamy nadzieję że niniejsza praca przybliżyła zagadnienia, z którymi musieliśmy się zmierzyć podczas pisania naszej aplikacji. Staraliśmy się w sposób jak najbardziej szczegółowy opisać wszystkie nasze działania, napotkane komplikacje, rozwiązane problemy.

Dzięki wytworzonemu produktowi udało nam się poszerzyć wiedzę o systemie *Android* oraz poznać najważniejsze biblioteki umożliwiające przyjazną prezentację danych. Stworzona aplikacja jest kompatybilna z założeniami *Google Cardboard*. Osiągnięcie takiego stanu rzeczy było naszym największym wyzwaniem i ostatecznie przerodziło się w nasz ogromny sukces.

Do zaprojektowania architektury i zaimplementowania algorytmów podczas wytwarzania aplikacji w pełni wykorzystaliśmy zasób wiedzy zdobyty podczas studiów.

6 Słownik pojęć

Adapter

Jest to obiekt będący mostem pomiędzy prezentacją danych a ich wartością. Zezwala na dostęp do listy przechowywanych danych i jest odpowiedzialny za tworzenie widoku dla każdego elementu z tej listy.

Akcelerometr

Nazywany inaczej również jako przyspieszeniomierz. Jest to przyrząd do pomiaru przyspieszeń liniowych lub kątowych. Mierzy on własny ruch. Stosowany jest do badania ruchu części maszyn i przeciążeń (np. samolotów), a także w nawigacji bezwładnościowej.

Aktywność (ang. *Activity*)

Jeden z podstawowych komponentów systemu *Android*. Dostarczona implementacja klasy jest odpowiedzialna za interakcję z użytkownikiem oraz współpracę z wbudowanymi serwisami urządzenia. W aplikacji *FindMyMeal* każda aktywność odpowiada za tworzenie nowego okna i uruchamianie podstawowych komponentów systemowych. Jedynie zarejestrowane aktywności w pliku konfiguracyjnym aplikacji mają dostęp do komponentów systemowych urządzenia takich jak np. Internet czy kamera.

Android KitKat – API 19

Android – system operacyjny z rodziny *Linux* dla urządzeń mobilnych. Obecnie najpopularniejszy system mobilny na świecie.

Android KitKat – API 19 jest jedną z wersji systemu operacyjnego *Android*. Wydana została 31.10.2013. Obecnie najpopularniejsza dystrybucja tego systemu. Jej następcą jest *Android 5.x* zwany *Lollipop*.

Aplikacja

Inaczej *oprogramowanie użytkowe*. Jest to program, który nie wchodzi w pakiet aplikacji dostarczanych wraz z systemem operacyjnym lub jako składnik któregoś z elementów urządzenia.

Aplikacja dzięki swojemu interfejsowi pozwala na bezpośredni kontakt z człowiekiem. Jej zadaniem jest przetwarzanie informacji oraz rozwiązywanie problemów zadanych przez użytkownika.

Binding

Łączenie ze sobą co najmniej dwóch komponentów. W aplikacji *FindMyMeal* wykorzystywane w celu łączenia widoków z danymi niezależnymi np. łączenie nazwy elementu z menu z obrazem go reprezentującym.

Fragment

Odpowiedzialny jest za część interfejsu przypisanej mu aktywności. Możliwe jest tworzenie różnorodnych fragmentów pojedynczej aktywności w celu stworzenia wielowidokowego ekranu np. dla widoku z zakładkami. Każdy fragment posiada własny cykl życia, ale nie może istnieć bez aktywności. Wszystkie fragmenty są odnawialne.

Google Maps

Mapy Google – serwis internetowy umożliwiający wyszukiwanie obiektów, oglądanie map i zdjęć lotniczych powierzchni Ziemi oraz udostępniający pokrewne im funkcje. Został stworzony w 2005 roku przez firmę *Google*.

GPS

Global Positioning System, jeden z systemów nawigacji satelitarnej, stworzony przez Departament Obrony Stanów Zjednoczonych, obejmujący swoim zasięgiem całą kulę ziemską. Zadaniem systemu jest dostarczenie użytkownikowi informacji o jego położeniu oraz ułatwienie nawigacji po terenie.

ImageView

Komponent wyświetlający wybrany obraz, na przykład ikonę czy pobraną z sieci grafikę.

Interfejs użytkownika

Część urządzenia odpowiedzialna za interakcję z użytkownikiem. Człowiek nie jest zdolny do bezpośredniej komunikacji z maszynami. Aby stało się to możliwe urządzenia są wyposażone w odpowiednie urządzenia wejścia-wyjścia tworzące razem interfejs użytkownika, w tym przypadku interfejs graficzny (GUI).

JSON

JavaScript Object Notation jest prostym formatem wymiany danych. Zapis i odczyt danych w tym formacie jest łatwy do opanowania przez ludzi. Jednocześnie, z łatwością odczytują go i generują komputery. Jego definicja opiera się o podzbiór języka programowania JavaScript.

Klient http

Klient HTTP otwiera połączenie i wysyła komunikat żądania do serwera *http*.

LayoutManager

Klasa odpowiedzialna za wyliczanie rozmiarów i pozycjonowanie widoków elementów znajdujących się w klasie *RecycleView*. Dzięki implementacji *LayoutManager'a* możliwe jest tworzenie własnej animacji w momencie przesuwania listy elementów znajdujących się w *RecycleView*.

Material design

Zbiór wytycznych zaprezentowanych przez firmę Google dotyczących prawidłowego prezentowania danych na urządzeniach z systemem *Android 21* lub wyżej.

MVVM

Wzorzec projektowy *Model-View-ViewModel*. Zawiera 3 elementy: *view* (widok), którego zadaniem jest wyświetlanie danych, *viewmodel* – eksponuje model dla widoku oraz *model*, który zarządza danymi.

RecycleView

Jest to elastyczny widok służący do prezentacji danych, najczęściej w postaci listy elementów, przy czym każdy element z listy posiada własny widok, który można ustandaryzować. Klasa ta pozwala na pozyskiwanie konkretnego elementu z listy, prezentację danych w postaci rozbudowanej animacji. W przypadku bardzo długiej listy elementów, nie generuje całego widoku na początku życia, jedynie w momencie wyświetlenia wybranej części.

Resources

Zasoby aplikacji dostępne z poziomu istniejącego kontekstu, na przykład w aktywności bądź fragmencie. Wykorzystywane do przechowywania informacji statycznych np. obrazki, napisy, wymiary.

Smartfon

Telefon komórkowy z większymi możliwościami, zwykle z dużym ekranem dotykowym.

TextureView

Komponent w którym można umieszczać tekstury (napis, wideo, obraz), w aplikacji *FindMyMeal* umieszczany jest tam obraz z kamery.

TextView

Komponent służący do wyświetlania tekstu.

Triangulacja

Metoda wyznaczania w terenie współrzędnych punktów, wykorzystująca taką własność trójkąta, że znajomość jednego boku i dwóch kątków wystarczy do pomiaru całej figury.

Widok

Zbiór komponentów prezentowanych użytkownikowi na urządzeniu. Zawiera informacje statyczne i dynamiczne. Odpowiada słowie *Layout* w technologii *Android*.

7 Bibliografia

[1] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, Blair MacIntyre, *Recent Advances in Augmented Reality,* IEEE Computer Graphics and Applications 21, 6 (Nov/Dec 2001)

[2] *http://www.e-edukacja.net/piata/referaty/sesja_IIIb/26_e-edukacja.pdf*

[3] *https://pl.wikipedia.org/wiki/Wsp%C3%B3%C5%82rz%C4%99dne_geograficzne*

[4] *http://encyklopedia.pwn.pl/haslo/wspolrzedne-geograficzne;3998457.html*

[5] *https://en.wikipedia.org/wiki/Mercator_projection#Derivation_of_the_Mercator_projection*

[6] *https://pl.wikipedia.org/wiki/Odwzorowanie_walcowe_r%C3%B3wnok%C4%85tne*

[7] *https://en.wikipedia.org/wiki/Magnetometer*

[8] *https://www.quora.com/How-does-a-compass-work-in-smart-phones-What-sensors-are-used-and-how-do-they-show-the-correct-directions*

[9] *https://pl.wikipedia.org/wiki/Hallotron*

[10] *https://pl.wikipedia.org/wiki/Anizotropowy_magnetoop%C3%B3r*

[11] *https://pl.wikipedia.org/wiki/Gigantyczny_magnetoop%C3%B3r*

[12] *http://encyklopedia.pwn.pl/encyklopedia/akcelerometr.html*

[13] Frank Ableson, Robi Sen, Chris King, *Android in action 2nd edition*, Manning, 2011

[14] *https://pl.wikipedia.org/wiki/Oprogramowanie_u%C5%BCytkowe*

[15] *https://pl.wikipedia.org/wiki/Mapy_Google*

[16] *https://pl.wikipedia.org/wiki/Global_Positioning_System*

[17] *https://pl.wikipedia.org/wiki/Interfejs_u%C5%BCytkownika*

[18] *http://sjp.pl/triangulacja*

[19] *http://sjp.pl/smartfon*

[20] *http://www.json.org/*

Warszawa, dnia .....

Oświadczenie

Oświadczam, że moją część pracy inżynierskiej pod tytułem: „Gogle do rzeczywistości wirtualnej i rozszerzonej oparte o ekran i kamerę smartfona”, której promotorem jest dr inż. Paweł Kotowski, wykonałem/am samodzielnie, co poświadczam własnoręcznym podpisem.

.....