 POLITECHNIKA WARSZAWSKA

WYDZIAŁ MATEMATYKI

I NAUK INFORMACYJNYCH

PRACA DYPLOMOWA INŻYNIERSKA

INFORMATYKA

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

**Goggles for virtual and augmented reality based on   
smartphone’s screen and camera**

Autor:

Anna Zawadzka

Współautorzy:

Sylwia Nowak

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.

Praca składa się z X części….

SŁOWA KLUCZOWE

Rzeczywistość rozszerzona | Android | Google Cardboard

**ABSTRACT**

The purpose of present thesis vas to create application for mobile devices…

KEY WORDS

Augmented reality | Android | Google Cardboard

Spis treści

[1 Wstęp 6](#_Toc441851474)

[1.1 Rzeczywistość wirtualna i rozszerzona 6](#_Toc441851475)

[1.2 Google Cardboard 7](#_Toc441851476)

[2 Aplikacja FindMyMeal 10](#_Toc441851477)

[2.1 Opis aplikacji 10](#_Toc441851478)

[2.2 Diagram przypadków użycia 12](#_Toc441851479)

[2.3 Opis działania aplikacji 13](#_Toc441851480)

[2.3.1 Uruchomienie aplikacji 13](#_Toc441851481)

[2.3.2 Menu główne 13](#_Toc441851482)

[*2.3.3* Opcja *Find* 15](#_Toc441851483)

[2.3.4 Opcja Favourites 16](#_Toc441851484)

[*2.3.5* Opcja *Help* 19](#_Toc441851485)

[2.3.6 Powrót 19](#_Toc441851486)

[2.3.7 Wyjście z aplikacji 20](#_Toc441851487)

[3 Wymagane środowisko sprzętowe i systemowe 21](#_Toc441851488)

[3.1 Wymagania systemowe 21](#_Toc441851489)

[3.2 Wymagania sprzętowe 21](#_Toc441851490)

[3.3 Wymagane moduły 21](#_Toc441851491)

[4 Architektura systemu 22](#_Toc441851492)

[4.1 Opis architektury 22](#_Toc441851493)

[4.2 Opis klas 25](#_Toc441851494)

[4.2.1 Klasa BaseActivity 25](#_Toc441851495)

[4.2.2 Klasa ApplicationUtils 26](#_Toc441851496)

[4.2.3 Klasa MainActivity 27](#_Toc441851497)

[4.2.4 Klasa MenuActivity 28](#_Toc441851498)

[4.2.5 Klasa FavouritesActivity 29](#_Toc441851499)

[4.2.6 Klasa NavigationActivity 31](#_Toc441851500)

[4.2.7 Klasa HelpActivity 34](#_Toc441851501)

[4.2.8 Klasa ItemViewHolder 34](#_Toc441851502)

[4.2.9 Klasa ScrollingLinearLayoutManager 36](#_Toc441851503)

[4.3 Biblioteki 37](#_Toc441851504)

[4.3.1 Cardboard SDK for Android v0.6.0 37](#_Toc441851505)

[4.3.2 ButterKnife v7.0.1 38](#_Toc441851506)

[4.3.3 AppCompat v7:23.1.0 38](#_Toc441851507)

[4.3.4 Google Play – Services v8.1.0 39](#_Toc441851508)

[4.4 Algorytmy 39](#_Toc441851509)

[4.4.1 Przekształcenia współrzędnych geograficznych 39](#_Toc441851510)

[4.4.2 Obliczanie wielkości ikon w menu głównym 43](#_Toc441851511)

[5 Proces wytwarzania oprogramowania 43](#_Toc441851512)

[5.1 Wersjonowanie 43](#_Toc441851513)

[6 Podsumowanie 44](#_Toc441851514)

[6.1 Problemy nierozwiązane 44](#_Toc441851515)

[6.1.1 Problem z zakłamywaniem odczytów magnetometru (sensora magnetycznego) 44](#_Toc441851516)

[6.2 Dalsze plany rozwojowe 46](#_Toc441851517)

[6.2.1 Rozwinięcia interfejsu 46](#_Toc441851518)

[6.2.2 Wyznaczanie dokładnej trasy nawigacji 48](#_Toc441851519)

[6.2.3 Konta użytkowników 48](#_Toc441851520)

[6.2.4 Sterowanie głosowe 49](#_Toc441851521)

[6.3 Wnioski 49](#_Toc441851522)

[7 Bibliografia 49](#_Toc441851523)

# Wstęp

## 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 z 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, tablety i PDA. 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.

## 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: https://www.google.com/intl/pl\_ALL/get/cardboard/get-cardboard/

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 ustawiać 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

# Aplikacja FindMyMeal

## 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 API Google Maps.

Interfejs użytkownika zbudowany jest przy użyciu biblioteki Google Cardboard. 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.

## Diagram przypadków użycia

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



Rysunek 5. Diagram przypadków użycia  
Źródło: opracowanie własne

## Opis działania aplikacji

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

### 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

### 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

### 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

### 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

### Opcja *Help*

Po wybraniu tej opcji następuje uruchomienie filmu instruktażowego.



Rysunek 16. Okno pomocy  
Źródło: opracowanie własne

### Powrót

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



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

### Wyjście z aplikacji

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



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

# Wymagane środowisko sprzętowe i systemowe

## Wymagania systemowe

Projekt realizowany jest w technologii *Android*.

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

## 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.

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

## 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.

# Architektura systemu

## 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 90 stopni w prawą stronę – następuje wyjście z aplikacji. Analogicznie dla przechylenia telefonu w lewą stronę o 90 stopni – 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

## Opis klas

### 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

### 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 lokalizacje (szerokość i długość geograficzną) na punkty płaszczyzny.



Rysunek 21. Klasa ApplicationUtils  
Źródło: opracowanie własne

### Klasa MainActivity

Dziedziczy po klasie *BaseActivity*. Klasa ta jest ustawiona jako *launcher* (uruchamiana jest jako pierwsza po starcie programu). Odpowiada za wyświetlenie loga 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

### 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

### 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 *ScollingFavouritesLinearLayoutManager* 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

### 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 tabelki 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

### Klasa HelpActivity

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



Rysunek 26. Klasa HelpActivity oraz HelpFragment  
Źródło: opracowanie własne

### 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 przejście do kolejnej. Obsługuje jedynie operacje zaznaczania i odznaczenia.

### 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* i *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).

## Biblioteki

Przy tworzeniu aplikacji FindMyMeal wykorzystane zostały zewnętrzne biblioteki.

### 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 4x4 zawierająca dane o wartościach z przedziału -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 danych dynamicznych 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.

### 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.

### 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*.

### 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.

## Algorytmy

### Przekształcenia współrzędnych geograficznych

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

#### Obliczanie kierunku nawigowania

##### Cel algorytmu

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

##### 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 z 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:

Wielkość *R* jest stałą skalowania mapy. Do naszych obliczeń przyjmujemy *R = 256* z *8-ktornym przybliżeniem* (tj. nasz obszar odwzorowywanej mapy zostaje podzielony na *28 kawałków*). Obliczenia te w swej postaci uwzględniają krzywiznę Ziemi.

1. 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 wielki jest kąt między wektorem różnicy a *osią* *X*.
2. Aby obliczony kąt był zrozumiały dla użytkownika musi on odnosić się do kierunku północnego - *osi* *Y*. Dlatego też do obliczonego kąta należy dodać miarę *π/2* co odpowiada obrotowi o *90º* przeciwnie do ruchu wskazówek zegara.
3. 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.

#### Obliczanie odległości między dwiema współrzędnymi geograficznymi

##### Cel algorytmu

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

##### 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)

}

### 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

# Proces wytwarzania oprogramowania

## Wersjonowanie

Do celów wersjonowania kodu naszej aplikacji wykorzystaliśmy programu *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 30. 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.

# Podsumowanie

## Problemy nierozwiązane

### Problem z zakłamywaniem 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 *smartphony* 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 . 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.

## 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 ...

### Rozwinięcia interfejsu

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

#### Kompas trójwymiarowy

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



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

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

#### 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 . 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 został implementacji został przedstawiony na rysunku 34.



Rysunek . Przykład możliwej implementacji poruszających sie strzałek  
Źródło: opracowanie własne

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



Rysunek . Przykład możliwej implementacji ścieżki kierunku  
Źródło: opracowanie własne

### Wyznaczanie dokładnej trasy nawigacji

### Konta użytkowników

Możliwość interakcji między użytkownikami aplikacji FindMyMeal zapewni wprowadzenie osobistych kont.

#### Logowanie

Po włączeniu aplikacji użytkownik będzie mógł zalogować się przy pomocy konta w serwisie Facebook lub Gmail (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.

Kuba SCREENYYYYYYYYYYYYYYYYYYYYYYYYYYYYY!!!!!!!111111oneoneone

#### 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.

#### 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.

### Sterowanie głosowe

## Wnioski

# 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), 34-47

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

[4]https://pl.wikipedia.org/wiki/Wsp%C3%B3%C5%82rz%C4%99dne\_geograficzne  
[5] <http://encyklopedia.pwn.pl/haslo/wspolrzedne-geograficzne;3998457.html>

[6]https://en.wikipedia.org/wiki/Mercator\_projection#Derivation\_of\_the\_Mercator\_projection  
[7] <https://pl.wikipedia.org/wiki/Odwzorowanie_walcowe_r%C3%B3wnok%C4%85tne>

[9] https://en.wikipedia.org/wiki/Magnetometer

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

[11] https://pl.wikipedia.org/wiki/Hallotron

[12] https://pl.wikipedia.org/wiki/Anizotropowy\_magnetoop%C3%B3r

[13] https://pl.wikipedia.org/wiki/Gigantyczny\_magnetoop%C3%B3r

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.

...........................................