Politechnika Warszawska

Wydział Mechatroniki

Praca dyplomowa inżynierska

Aplikacja do lokalizacji w budynkach na podstawie sieci Wifi

Mateusz Wojewódka

Numer albumu 269979

Promotor

mgr inż. Bartłomiej Fajdek

**Streszczenie**

**Aplikacja mobilna do lokalizacji w budynkach na podstawie sieci Wifi, we współpracy ze zdalnym serwerem.**

**Niniejsza praca przedstawia opis użytych technologii oraz implementację aplikacji mobilnej oraz serwera, które razem tworzą system lokalizujący użytkownika w budynku na podstawie sieci Wifi.  
 Aplikacja ma dwa tryby. Pierwszy z nich służy do określania własnej lokalizacji na podstawie sieci Wifi w danym pomieszczeniu. Natomiast drugi służy do tworzenia nowych pomieszczeń i dokonywania niezbędnych pomiarów do użytkowania w trybie pierwszym.  
Aplikacja mobilna została zaimplementowana na platformę Android w języku Android/Java przy użyciu narzędzi oferowanych przez Android Studio.  
Serwer został zaimplementowany jako aplikacja desktopowa w technologii C# na platformę .NET przy użyciu narzędzi oferowanych przez Visual Studio.  
 O poprawności zaimplementowanej logiki świadczy zgodny z oczekiwaniami, pomyślny przebieg testów.**

Contents

[1. Wstęp 5](#_Toc512372169)

[1.1. Indoor positioning system (IPS) 5](#_Toc512372170)

[1.2. Korzyści jakie daje IPS 5](#_Toc512372171)

[1.3. Cel pracy 5](#_Toc512372172)

[2. Wifi based positioning system (WPS) 6](#_Toc512372173)

[2.1. Zarys ogólny 6](#_Toc512372174)

[2.2. Istniejące techniki 6](#_Toc512372175)

[2.2.1. RSSI 7](#_Toc512372176)

[2.2.2. AoA 7](#_Toc512372177)

[2.2.3. ToF 8](#_Toc512372178)

[2.2.4. Fingerprinting 9](#_Toc512372179)

[2.3. Wybór techniki 9](#_Toc512372180)

[3. Architektura systemu oraz wykorzystane technologie 9](#_Toc512372181)

[3.1. Zarys komunikacji w aplikacji 10](#_Toc512372182)

[3.2. Komunikacja klient - serwer 10](#_Toc512372183)

[3.2.1. Model TCP/IP 10](#_Toc512372184)

[3.2.2. Protokół HTTP 12](#_Toc512372185)

[3.2.3. REST 18](#_Toc512372186)

[3.2.4. Dlaczego REST? 21](#_Toc512372187)

[3.3. Aplikacja mobilna 21](#_Toc512372188)

[3.3.1. System Android 22](#_Toc512372189)

[3.3.2. MVC i pochodne 24](#_Toc512372190)

[3.3.3. Komunikacja z serwerem 26](#_Toc512372191)

[3.4. Aplikacja serwera 26](#_Toc512372192)

[3.4.1. C#/.NET 26](#_Toc512372193)

[3.4.2. Komunikacja z klientem 28](#_Toc512372194)

[3.4.3. Komunikacja z bazą danych 29](#_Toc512372195)

[3.5. MySql 29](#_Toc512372196)

[4. Implementacja systemu 30](#_Toc512372197)

[4.1. Sposób działania systemu 30](#_Toc512372198)

[4.2. Baza danych 30](#_Toc512372199)

[4.3. Aplikacja serwera 30](#_Toc512372200)

[4.3.1. Struktura solucji 31](#_Toc512372201)

[4.3.2. REST API 34](#_Toc512372202)

[4.3.3. Uruchomienie aplikacji 35](#_Toc512372203)

[4.4. Aplikacja mobilna 35](#_Toc512372204)

[4.4.1. Struktura projektu 35](#_Toc512372205)

[5. Testy 44](#_Toc512372206)

[6. Wnioski 44](#_Toc512372207)

[Bibliography 45](#_Toc512372208)

# Wstęp

## Indoor positioning system (IPS)

W dzisiejszych czasach bardzo powszechne jest używanie modułu GPS do lokalizacji. Każdy z nas używa go podczas w podróży lub przy poruszaniu się po mieście. Niestety sygnał GPS nie zawsze działa zbyt dobrze. W budynkach jego dokładność będzie stosunkowo mała. W przypadku gdy znajdziemy się na targach lub w galerii możemy mieć problem z bezpośrednim dostaniem się do danego punktu np. stoiska lub sklepu. W takim przypadku dobrym rozwiązaniem jest zastosowanie IPS (ang **Indoor positioning system**), który umożliwia lokalizacje w budynkach z dużo większą dokładnością niż GPS (poniżej 1 metra).   
Mając zaimplementowany taki system możemy zaoferować użytkownikowi określenie własnej lokalizacji w budynku. Do określania lokalizacji stosuje się różne technologie, działają one na podstawie sygnałów takich jak:

* Fale radiowe
* Pole magnetyczne
* Sygnały akustyczne

Popularnym jest użycie sygnałów radiowych, najczęściej **Bluetooth** oraz **Wifi**.

## Korzyści jakie daje IPS

Dedykowana aplikacja do lokalizacji w budynku może oferować znacznie więcej niż tylko lokalizację. Można ją zaprojektować w taki sposób aby wysyłała na serwer informacje gdzie i w jakich godzinach znajduje się dana osoba. Aplikacja mogłaby również otrzymywać powiadomienia np. o promocji w pobliskim sklepie. Przykładowa Galeria Handlowa oferując taki system z mapą gdzie byłaby wskazana obecna lokalizacja, mogłaby znacząco ułatwić użytkownikom poruszanie się po obiekcie.

Natomiast implementując powiadomienia, których przykład dałem parę zdań wcześniej, mogłaby zwiększyć ilość sprzedawanego towaru. Widać zatem, iż wprowadzenie systemu IPS w przypadku dużych obiektów handlowych zaowocowałoby ułatwionym poruszaniem się po nich jak i zwiększonym dochodem pieniężnym.

## Cel pracy

W niniejszej pracy zajęto się lokalizacją na podstawie sygnału sieci Wifi. System lokalizacji, który używa tego sygnału jest nazywany **WPS,** czyli **Wifi** based **Positioning System**.   
Celem jest implementacja aplikacji umożliwiającej lokalizację w budynkach na podstawie wektora **wskaźników** **mocy (RSSI) odbieranych sygnałów radiowych (Wifi)** generowanych przez sieć w danym budynku. Nazwa stworzonego systemu to **Wifi Network Localizator**. Posiadacz danej aplikacji znajdując się w budynku, dla którego zostały wykonane pomiary, może określić w jakim punkcie w przestrzeni obecnie się znajduje, poprzez komunikację ze zdalnym serwerem - niekoniecznie hostowanym przez sieć w danym budynku, może być to serwer hostowany publicznie przez dowolną organizację.

# Wifi based positioning system (WPS)

## Zarys ogólny

System **WPS**, który jest tematem pracy, jest używany tam gdzie lokalizacja GPS jest niewystarczająca ze względu na wszelkiego rodzaju zakłócenia oraz blokadę sygnału wewnątrz budynków. WPS’owi dużo korzyści przyniósł gwałtowny rozwój **WAP** na początku XXI wieku.

**Wireless Access Point (WAP)** – bezprzewodowy punkt dostępu, czyli urządzenie zapewniające dostęp do sieci komputerowej innym urządzeniom posiadającym moduł Wifi. Nośnikiem transmisyjnym w takiej komunikacji są fale radiowe.

Najbardziej popularną i rozpowszechnioną metodą używaną do pozycjonowania jest pomiar mocy otrzymanego sygnału (**RSSI**). Użyteczne parametry do geolokacji hotspot’u Wifi lub access pointu to między innymi:

* SSID – identyfikator sieci, jest to potocznie mówiąc nazwa danej sieci. Dodatkowo wszystkie access point’y, które mają pracować jako jedna sieć muszą używać tego samego SSID.
* MAC address – jest to unikalny identyfikator danego urządzenia WAP. Tutaj w przeciwieństwie do SSID, każdy access point pracujący w jednej sieci ma inny, unikalny adres MAC.

Dokładność pomiaru zależy od liczby pozycji umieszczonych w bazie danych. Baza hotspot’ów   
jest wypełniana poprzez korelacje danych o lokalizacji GPS z urządzenia przenośnego oraz adresami MAC hostspot’ów Wifi. Możliwe jest występowanie fluktuacji sygnału, która zwiększa niedokładność. Jednak istnieją specjalne techniki, które pozwalają filtrować te zakłócenia.

## Istniejące techniki

Generalnie problem polega na lokalizacji urządzenia w odniesieniu do dostępnych access point’ów. Spośród technik, które to realizują można wydzielić następujący podział:

* **RSSI** - Received Signal Strenghtand lateration
* **AoA** - Angle of Arrival
* **ToF** - Time of flight
* **Fingerprinting**

W większości z wyżej wymienionych technik, pierwszym krokiem jest określenie odległości pomiędzy urządzeniem klienta, a kilkoma dostępnymi access point’ami. Znając te odległości możliwe jest, metodą **trilateracji**, określić relatywną pozycję urządzenia klienta w odniesieniu do pozycji access point’ów. Jako alternatywa możliwe jest wykorzystanie kąta nadchodzących sygnałów do urządzenia klienckiego. Wtedy relatywną pocyzję klienta określamy za pomocą metody **triangulacji**.

### RSSI

W tej technice, lokalizacja polega na pomiarze mocy sygnałów (RSSI) z urządzenia klienta dla kilku różnych access point’ów. Następnie użycie tych informacji dla określenia odległości od nich. Metoda **trilateracji**, może być użyta do obliczenia przybliżonej pozycji urządzenia klienta, w odniesieniu do znanej wcześniej pozycji access pointów.

Jest to jedno z najłatwiejszych i najtańszych rozwiązań do implementacji. Jego główną wadą jest to, iż nie zapewnia zbyt dużej dokładności. Jest to około od 2 do 4 metrów. Dzieje się tak ponieważ pomiary **RSSI** mają skłonności do zakłóceń w związku ze zmianami środowiska (zmiany infrastruktury w budynku).

### AoA

Wraz z nadejściem użytkowania **MIMO Wifi**, możliwe stało się określenie **AoA**, otrzymywanego sygnału radiowego z urządzenia klienta. Odbywa się to przy użyciu anten w access point’ach (MIMO). Następnie przy zastosowaniu metody **triangulacji** obliczana lokalizacji urządzenia klienta.

**MIMO** – jest to rozwiązanie polegające na wieloantenowej transmisji po stronie nadawczej jak i odbiorczej. Możliwe dzięki temu jest zwiększenie między innymi przepustowości sieci. W przypadku tej metody pomiarowej ważne jest aby MIMO było zaimplementowane po strony odbiorcy sygnału klienta, czyli access point’u.  
  
**AoA** – jest to metoda określenia kierunku propagacji fali radiowej na podstawie pomiaru kąta nadejścia fali. Metoda ta, do określenia kierunku używa pomiarów różnicy czasów nadejścia sygnału odbieranego poprzez poszczególne elementy szyku antenowego. Następnie znając te opóźnienia obliczany jest kąt nadejścia sygnału.

Incident signal 
1 
Antenna Array 
d 
d.sin9 
2 
Linear array of antennas receiving a signal. The phase-shift difference of the received signal arriving at 
antennas equally separated by a "d" distance is used to compute the angle of arrival of the signal. Picture 
reproduced from 

Rys. 1 Tablica anten odbierająca sygnał. Przesunięcie fazowe otrzymywanego sygnału jest używane do wyznaczenia kąta nadejścia sygnału. [1]

### ToF

Metoda ta wykorzystuje czas na otrzymanie informacji zwrotnej w komunikacji poprzez sieć Wifi urządzeń klienta oraz access point’u. Zostaje tutaj wyznaczona relatywna pozycja urządzeń klienta względem access point’ów. Pomiary takich czasów są rzędu nanosekund. Systemy, które używają tych technik są obarczone błędem rzędu 2m, w wyznaczaniu lokalizacji. Technika ta jest stosowana w budynkach gdzie taka dokładność (2-3m) jest wystarczająca. Pomiar czasu w tej komunikacji bazuje na fakcie, iż fale radiowe przemieszczają się z prędkością zbliżoną do prędkości światła. Prędkość ta pozostaje praktycznie niezmienna w większości ośrodków propagacji w pomieszczeniach zamkniętych. W związku z tym ta metoda nie zależy aż tak bardzo od zmian środowiska pomiarowego jak metoda **RSSI and lateration**. Podobnie jak w metodzie **RSSI**, jest tutaj wyznaczana odległość między klientem a acces point’ami. Następnie metodą **trilateracji** wyznacza się lokalizacje urządzenia w odniesieniu do access point’ów.

M easuring Station 
Target Client Device 
DATA 
DATA 
T MEASURE 
T ACK 
ACK 
ACK 
Figure showing a measuring station sending a DATA frame to a client station and waiting until 
receiving the ACK. is the scheduling delay (offset) originated at the target client device, and it 
depends on how much time it takes for the ACK to be scheduled. T_P is the signal propagation time 
between transmitter and receiver, and is usually assumed to be the same on the way to the target and 
back. T ACK is the time needed to transmit the ACK frame. The time of flight corresponds to the 
[16] 
T_MEASURED. Picture reproduced from 

Rys. 2 Stacja pomiarowa wysyła dane - DATA, do klienta. Następnie oczekuje na wiadomość zwrotną ACK. ϭ jest opóźnieniem pochodzącym z urządzenia klienta i zależy od tego jak dużo czasu zajmuje rozpoczęcie wykonywania ACK. T\_P jest czasem propagacji sygnału pomiędzy nadawcą, a odbiorcą i zazwyczaj zakłada się iż jest to ten sam czas w drodze od nadawcy do klienta i z powrotem. T\_ACK jest czasem potrzebnym do przekazania ramki ACK. Czas „lotu” danej informacji odpowiada czasowi T\_MEASURED. [2]

### Fingerprinting

Metoda ta również korzysta z RSSI, jednak polega na pomiarze mocy sygnałów z kilku access point’ów w zasięgu, a następnie zapisywaniu tych informacji w bazie danych razem ze znanymi współrzędnymi urządzenia klienta. Podczas wykonywania śledzenia własnej pozycji obecny wektor RSSI w nieznanej lokalizacji, porównywany jest do tych zapisanych uprzednio w bazie danych. Najbliższy pasujący wynik jest zwracany jako przybliżona lokalizacja klienta. Systemy tego typu zapewniają dokładność od okłodo 0,5 do 1,3m.

Główną zaletą tej metody jest to, iż dokonując pomiarów nie musimy wiedzieć praktycznie nic o obiekcie, w którym się znajdujemy. Nie musimy znać współrzędnych access point’ów, których sygnałów będziemy używać ani tego jak tłumiony ich jest sygnał i ile przeszkód znajduje się po drodze. Interesuje nas tylko wektor RSSI sygnałów w danym punkcie w przestrzeni.

Natomiast największą wadą tej metody jest fakt, iż każda zmiana środowiska np. dodanie lub usunięcie jakiegoś mebla w budynku, może spowodować konieczność aktualizacji bazy danych.

## Wybór techniki

Techniką, którą wybrano do implementacji jest **Fingerprinting**. Głównym powodem jest, prostota implementacji oraz brak koniczności zagłębiania się w infrastrukturę budynku, aby określić położenie wybranych access point’ów w przestrzeni. Ta metoda umożliwia zmapowanie wektorów RSSI na punkty w przestrzeni w dowolnym budynku, a następnie umożliwienie śledzenia własnej lokalizacji na podstawie pomiarów. Potrzebne jest tylko urządzenie odbierające sygnały sieci Wifi – w moim przypadku telefon komórkowy, oraz komunikacja z bazą danych, aby zapisywać i odczytywać pomiary.

# Architektura systemu oraz wykorzystane technologie

## Zarys komunikacji w aplikacji



Rys. 3 Architektura komunikacji w systemie

System składa się z trzech głównych warstw. Taki podział pozwala na pozwala na zwiększenie obszaru zastosowań systemu. Z systemu będą mogły korzystać nie tylko telefony komórkowe z określonym systemem, ale każde urządzenie mające zdolność do komunikacji przez sieć Wifi oraz implementujące określony protokół komunikacyjny. Zaprezentowane wyżej warstwy składają się na:

* Warstwę aplikacji mobilnej, czyli klienta. Tutaj następuje interakcja użytkownika z systemem, poprzez interfejs graficzny. Komunikuje się ona z serwerem drogą sieciową.
* Warstwa serwera aplikacji, która przyjmuje zapytania od klienta i poprzez komunikację określonym protokołem z bazą danych wykonuje żądania klienta.
* Warstwa bazy danych, w która wykonuje zapytanie warstwy serwera aplikacji wyspecyfikowane w języku zapytań SQL.

## Komunikacja klient - serwer

W tym podrozdziale zostaną opisane modele oraz protokoły użyte do komunikacji pomiędzy klientem a serwerem.

Do komunikacji użyto protokołu **HTTP**, który opiera się on na modelu **TCP/IP**. Dodatkowo komunikację tę zaimplementowano na wzór architektury **REST**.

### Model TCP/IP

Model **TCP/**IP jest podstawą struktury Internetu. Wiele protokołów sieciowych takich jak HTTP, FTP czy Telnet jest na nim oparta. TCP/IP jest teoretycznym modelem warstwowej struktury protokołów komunikacyjnych. Jest on złożony z czterech warstw: Aplikacji, Transportowej, Internetowej oraz Dostępu do Sieci. Każda warstwa w modelu TCP/IP odpowiada jednej lub więcej warstwie w siedmiowarstwowym modelu OSI.



Rys. 4 Architektura modelu TCP/IP [3]

**Network Interface Layer**

Jest to warstwa dostępu do sieci. Warstwa ta jest odpowiedzialna wysyłanie i odbieranie pakietów TCP/IP w komunikacji pomiędzy fizycznymi urządzeniami sieciowymi. Zwykle takimi jak karty sieciowe lub modemy.

**Internet Layer**

Jest to warstwa Internetowa, która posiada wiedzę dotyczącą topologii sieci. Najbardziej powszechnym protokołem komunikacyjnym tej warstwy jest protokół IP. Odpowiada za trasowanie/routing, czyli określenie optymalnej trasy i wysłanie nią pakietu danych. A także za fragmentację danych i adresowanie IP.

**Transport Layer**

Jest to warstwa transportowa, w której mają zastosowanie dwa protokoły komunikacyjne: **TCP** oraz **UDP**.

* **TCP** zapewnia niezawodną komunikacje jeden do jednego. Jest odpowiedzialne za ustanawianie połączenia TCP, oraz za sekwencjonowanie i potwierdzanie wysyłanych pakietów. Oferuje także odzyskiwanie pakietów utraconych podczas transmisji.
* **UDP** zapewnia zawodną komunikację jeden do wielu lub jeden do jednego. Jest używany kiedy wysyłana jest mała ilość danych, gdy nie pożądane jest ustanawianie połączenia TCP, lub gdy protokoły wyższych warstw zapewniają niezawodność dostarczania informacji.

**Application Layer**

Warstwa aplikacji zapewnia aplikacjom dostęp do usług oferowanych przez resztę warstw oraz wybranie protokołu komunikacyjnego do wymiany danych. Najbardziej znanymi protokołami warstwy aplikacji są:

* HTTP zazwyczaj używany do przesyłania plików budujących strony internetowe.
* FTP jest używany do dwukierunkowego transferu plików w układzie klient serwer.
* SMTP używany do transferu załączników i wiadomości email.
* Telnet używany do obsługi odległego terminala w architekturze klient-serwer.

### Protokół HTTP

Jest to protokół warstwy aplikacji w modelu TCP/IP. HTTP w warstwie transportowej używa protokołu TCP lub - gdy wymagane jest szyfrowanie połączenia - TLS. Pozwala on na pobieranie zasobów takich jak dokumenty HTML. Jest także wykorzystywany przez interfejsy WebAPI.



Rys. 5 Wykorzystanie HTTP w sieci [3]

Jest protokół klient-serwer, oznacza to, że komunikacja jest nawiązywania z żądania klienta. Nie będzie takiej sytuacji, w której to serwer wyśle dane do klienta, jako pierwszy inicjując komunikację. Serwer musi zostać odpytany, aby rozpocząć komunikacje z klientem. Ten protokół jest powszechnie wykorzystywany przez przeglądarki internetowe. Są nim zarówno tekst jak i obrazy, filmy i wiele innych.



Rys. 6 Komunikacja protokołem HTTP [3]

Klient i serwer nie nawiązują stałego połączenia. Używając protokołu HTTP wymieniają się wiadomościami. Wiadomość przesłana przez klienta jest nazywana *request*, natomiast wiadomość zwrotną od serwera określa jako *response*.  
W trakcie wysyłania komunikacji z serwerem, wysłany *request* nie trafia bezpośrednio do niego. Pomiędzy klientem, a serwerem znajdują się routery, modemy itp. Jednakże dzięki warstwowej architekturze sieci (model TCP/IP) byty pośredniczące są ukryte w warstwach niższych, czyli Internetowej oraz Transportowej. Warstwy te są zazwyczaj nieistotne z punktu widzenia warstwy protokołu HTTP.

**Podstawowe cechy protokułu HTTP**

* **Prostota –** HTTP został zaprojektowany tak aby być czytelnym dla ludzi. Wiadomości przesyłane tym protokołem są stosunkowo łatwe do analizy co ułatwia pracę deweloperom.
* **Rozszerzalność –** nagłówki wprowadzone w HTTP/1.0 uczyniły ten protokół łatwym do rozszerzania i eksperymentowania.
* **Bezstanowość –** HTTP jest bezstanowy. Nie ma żadnego powiązania pomiędzy dwoma *requestami* przeprowadzonymi na tym samym połączeniu. Jednakże nie oznacza to braku możliwości utworzenia sesji. Mechanizm ciasteczek (*cookies*) pozwala utworzyć sesje w danej komunikacji. Serwer wysyłając żądanie utworzenia ciasteczka na dysku użytkownika, wysyła w nagłówku polecenie „Set-Cookie” oraz informacje dotyczące jego nazwy, czasu ważności i innych ograniczeń. Następnie klient komunikując się z serwerem szuka niewygasłych ciasteczek dla danej domeny i dołącza je go nagłówka HTTP, podtrzymując tym samym sesję.
* **Łączność** – połączenie, w protokole HTTP, jest realizowane przez warstwę transportową (model TCP/IP). HTTP wymaga od warstwy transportowej jedynie tego, aby była niezawodna. Nie może sobie pozwolić na straty w przesyłaniu wiadomości. Istnieją dwa najbardziej popularne protokoły komunikacyjne TCP, który jest niezawodny, oraz UDP, który jest zawodny. Zatem z oczywistych względów wybrano TCP. HTTP ustanawia połączenie TCP za każdym *requestem* i *responsem*. Główną wadą takiego rozwiązania jest spowolnienie procesu komunikacji, ze względu na ciągłe ustalanie od nowa tego samego połączenia (w komunikacji z danym klientem). Jednakże można zwiększyć efektywność tego procesu poprzez wysyłanie kilku wiadomości w regularnych odstępach czasu, ponieważ „gorące” połączenia są bardziej efektywne niż „zimne”. Dlatego HTTP/1.1 wprowadziło „*pipelining*” oraz stałe połączenia. Połączenie TCP może być częściowo kontrolowane poprzez pole „Connection” w nagłówku HTTP. W HTTP/2 poczyniono dalsze kroki i wprowadzoni „*multiplexing*”, który pomaga utrzymać połączenie „gorące” i bardziej efektywne.

**HTTP pipelining** – technika, polegająca na wysłaniu wielu żądań HTTP jednocześnie w ramach jednego połączenia TCP. Odpowiedzi muszą być zwrócone w tej samej kolejności w jakiej zostały wysłane żądania. Dzięki tej technice została zwiększona wydajność w środowiskach z połączeniami o dużych opóźnieniach. Wadą takiego rozwiązania jest fakt, iż kolejność odpowiedzi jest określona z góry. Może być to przyczyną opóźnień, gdy żądanie o obiekt o dużej objętości zostanie wysłane przed żądaniami o inne niezbędne „lekkie” obiekty.

**HTTP multiplexing –** technika, podobnie jak pipelining, polega na wysyłaniu wielu żądań HTTP jednocześnie w ramach jednego połączenia TCP. Jednakże odpowiedzi mogą być zwracane w dowolnej kolejności, niezależnie od kolejności wysłanych żądań. Pozwoliło to jeszcze bardziej zwiększyć wydajność komunikacji.



Rys. 7 Porównanie technik pipelining oraz multiplexing [7]

**Metody HTTP**

HTTP definiuje metody, aby oznaczyć co dana akcja będzie robić w odniesieniu do danego zasobu. Utrzymywanie poprawnej konwencji ułatwia zrozumienie komunikacji podczas analizy wiadomości HTTP.

* GET – pobieranie zasobu określonego przez URI
* HEAD – pobieranie informacji o zasobie określonym przez URI
* PUT – tworzenie nowego zasobu lub jego aktualizacja
* POST – wysłanie żądania i zawartości stworzenia nowego zasobu
* DELETE – usunięcie zasobu określonego przez URL
* OPTIONS – zwraca metody HTTP, które serwer oferuje dla określonego URL
* TRACE – analiza kanału komunikacji
* CONNECT – rozpoczyna dwustronną komunikację z określonym zasobem. Może być użyte do tunelowania
* PATCH – aktualizacja części danych

**Wiadomości HTTP**

Wiadomości dla wersji HTTP/1.1 i wcześniejszych są czytelne dla człowieka. W wersji HTTP/2, wiadomości opakowane są w strukturę binarną, ze względów optymalizacyjnych takich jak kompresja i multiplexing. Jednakże nawet jeśli wiadomość jest w formacie HTTP/2, to po stronie klienta następuje rekonstrukcja do formatu HTTP/1.1.

Są dwa typy wiadomości *request* (żądanie)oraz *response* (odpowiedź).

**Request**

Rys. 8 Budowa wiadomości odpytującej - żądania [3]

* **Method** – metoda określająca akcje wykonaną na zasobie określonym przez URL
* **Path** – ścieżka do danego zasobu, na którym wykonywana jest operacja
* **Version of the protocol** – wersja protokołu HTTP
* **Headers** (opcjonalnie) – znajdują się tu dodatkowe informacje dla odbiorcy
* **Body** (opcjonalnie)– wykorzystywane w metodach takich jak PUT/POST, zawiera dane zasobu którymi chcemy nadpisać/zaktualizować istniejący, lub stworzyć nowy.

**Response**

Rys. 9 Budowa wiadomości zwrotnej – odpowiedzi [3]

* **Version of the protocol –** wersja protokołu HTTP
* **Status code –** mówi o tym czy request był poprawny, jeśli nie, informuje również dlaczego
* **Status message –** krótki opis **Status code**
* **Headers** (opcjonalnie) – znajdują się tu dodatkowe informacje dla odbiorcy
* **Body** (opcjonalnie) – zawiera pobierany przez klienta zasób

Istnieje wiele rodzajów **Status codes / Status messages**, które dobieramy w zależności od przeprowadzonej operacji i stanu zasobu.



Rys. 10 Status codes razem z wiadomościami

### REST

Representational State Transfer – jest to styl architektury oprogramowania. Określa on sposób projektowania oprogramowania przy wykorzystaniu możliwości, jakie daje protokół **HTTP**. Standard REST, ułatwia wzajemną komunikację systemom, które są na nim oparte. „RESTful systems”, charakteryzują się między innymi jednorodnym interfejsem, bezstanowością, zasobami oraz separacją klienta i serwera.

**Separacja klienta i serwera**

Oznacza to, że klient i serwer implementują swoje systemy niezależnie. Kod po stronie klienta może być zmieniony i nie wymaga to żadnych zmian po stronie serwera. Działa to też w drugą stronę.

Dopóki klient i serwer wiedzą, w jakim formacie wysyłać między sobą wiadomości, mogą pozostać odseparowani. Zwiększa to rozszerzalność aplikacji, zarówno serwera jak i klienta. Mogą one ewoluować w swoim własnym, niezależnym tempie.

**Bezstanowość**  
  
Oznacza to, że ani serwer, ani klient nie muszą nic wiedzieć o swoim stanie. Powinni być zdolnie do odczytania każdej otrzymanej wiadomości, bez względu na to, jaka była poprzednia. Bezstanowość odnosi się głównie do zasobów, do których klient odwołuje się za pomocą metod HTTP. Oczywiście przy odwoływaniu się do zasobów możemy wymagać, aby tylko niektórzy użytkownicy mieli do nich dostęp. Jak wtedy połączyć to z bezstanowością? Otóż, najczęstszym rozwiązaniem jest przysyłanie informacji identyfikującej w nagłówku HTTP, przy każdym odwoływaniu się do zasobu. Może być to token, wygenerowany i wysłany do użytkownika przez serwer podczas operacji logowania.

**Wysyłanie żądań przez klienta**

Klient wysyła żądanie, aby pobrać lub zmodyfikować dany zasób. W skład żądania wchodzą:

* Metoda HTTP definiująca, jaka operacja ma być przeprowadzona na zasobie, najpopularniejsze to: GET, POST, PUT, DELETE.
* Nagłówek, który zawiera dodatkowe informacje jak interpretować żądanie.
* Ścieżka do zasobu.
* Opcjonalnie ciało żądania, zawierające dane odnoszące się do zasobu.

Jeśli w ciele żądania wysyłamy dane w specjalnym formacie, możemy poinformować o tym odbiorcę poprzez dodanie w nagłówku pola **Content-Type**. Natomiast jeśli chcemy odebrać dane w określonym przez nas formacie dodajemy w nagłówku pole **Accept**. Oba te pola wypełniamy jednym lub wieloma tzw. *MIME types* (Multipurpose Internet Mail Extensions). Przykładowo jeśli chcemy wysłać lub otrzymać wiadomość w formacie JSON, MIME type jakiego użyjemy to: „application/json”. Często używane MIME types:

* Obraz – image/png, image/jpeg, image/gif
* Dźwięk – audio/wav, audio/mpeg
* Wideo – video/mp4, video/ogg
* Aplikacje – application/json, application/xml, application/pdf

W RESTful API, ścieżki do zasobu powinny być zaprojektowane w taki sposób, aby ułatwić użytkownikowi zrozumienie co jest konsekwencją danej operacji. Zgodnie z konwencją pierwsza część ścieżki powinna być rzeczownikiem w liczbie mnogiej.   
Dla ścieżki:

*library.com/books/32/order/112*

Jest jasne, iż odwołujemy się do zamówienia o id 112, dla książki o id równym 32. Metody HTTP w żądaniu oznaczałyby operację bezpośrednio na tym zasobie.  
Kiedy z kolei, chcielibyśmy odwoływać się do listy lub jakiegoś innego kontenera danych zasobów. Nie użylibyśmy id, wystarczyłoby:

*library.com/books*

Przy metodzie GET, oznaczałoby to pobranie listy dostępnych książek. Natomiast przy metodzie POST, dodanie nowych książek oraz wygenerowanie dla nich id.

**PUT vs POST**

O ile dla metod GET oraz DELETE od razu wiadomo do czego służą, to z doborem metody PUT czy POST mogą wystąpić pewne problemy.   
Metody GET, DELETE oraz PUT są idempotentne. Oznacza to, że mogą być wywoływane dla danego zasobu wiele razy, a i tak efekt końcowy będzie ten sam.  
Natomiast metoda POST idempotentna już nie jest.   
Jeśli będziemy dodać nową książkę do bazy danych, to używając metody POST, w ciele żądania określimy np. tytuł danej książki, ale już po stronie serwera będzie nadanie jej określonego id. Oznacza to, że jeśli wykonamy dwa razy metodę POST na tym samym zasobie z tym samym ciałem żądania, to zostaną dodane dwie książki o tym samym tytule, lecz ich id będą różne. W metodzie PUT, sami określamy id książki, jaka zostanie dodana do bazy danych. Wtedy wykonanie dwa razy operacji PUT na określonym zasobie zaskutkuje co najwyżej jego aktualizacją.

* PUT: *library.com/books/3 –* dodanie lub aktualizacja książki o id równym 3.
* POST: *library.com/books –* dodanie nowej książki, dla której serwer sam wygeneruje nowe id.

**Wysyłanie odpowiedzi**

Format odpowiedzi podobnie jak dla żądania określamy przy pomocy pola **Content-Type** umieszczonego w nagłówku. Pole może być wypełnione za pomocą *MIME types* takimi jak dla żądania. W odpowiedzi na żądanie możemy przesłać informacje o przebiegu danej operacji, używają **Status code** oraz **Status message** z protokołu HTTP. Powinny one jak najlepiej obrazować to, co stało się po stronie serwera. Dla najczęściej używanych metod HTTP, oczekiwane kody i wiadomości zwrotne to:

* GET – 200 (OK)
* POST – 201 (CREATED)
* PUT – 200 (OK)
* DELETE – 204 (NO CONTENT)

**Przykłady komunikacji**

1. Pobieranie dostępnych książek

**Żądanie**

*GET http://library.com/books*

*Accept: application/json*

**Odpowiedź:**

*Status Code: 200 (OK)*

*Content-type: application/json*

*Body:*

*[*

*{*

*"bookId": 1,*

*"bookName": "Książka 1",*

*"bookAuthor": "Autor 1"*

*},*

*{*

*" bookId": 2,*

*" bookName": "Książka 2",*

*"bookAuthor": "Autor 2"*

*},*

*{*

*" bookId": 3,*

*" bookName": "Książka 3",*

*"bookAuthor": "Autor 3"*

*}*

*]*

1. Dodanie nowych książek do bazy danych

**Żądanie**

*GET http://library.com/books*

*Body:*

*[*

*{*

*" bookName": "Moja Książka",*

*"bookAuthor": "Ja"*

*},*

*{*

*" bookName": "Twoja książka",*

*"bookAuthor": "Ty"*

*}*

*]*

**Odpowiedź:**

*Status Code: 200 (OK)*

### Dlaczego REST?

Wybór komunikacji padł na HTTP w architekturze REST, ze względu na prostotę tego rozwiązania. Obecnie panuje trend w tworzeniu aplikacji webowych opartych o REST. Aplikacje te są rozszerzalne i elastyczne oraz łatwe w utrzymaniu.   
Co do formatu danych, pod uwagę były brane dwa bardzo popularne formaty  
XML oraz JSON. Wybrano format JSON, ze względu na jego lekkość w porównaniu do formatu XML.

## Aplikacja mobilna

### System Android

Aplikacja mobilna została napisana pod system Android w języku Java.  
Android to mobilny system operacyjny, który został wyprodukowany przez Google. Jest to projekt open-source’owy, bazujący na jądrze Linux’owym. Dzięki temu jest dostępny dla szeregu urządzeń mobilnych.



Rys. 11 Główne komponenty systemu Android [4]

Na załączonej grafice widnieje podział na główne elementy systemu Android. Są to:

**Linux Kernel**

Jest to podstawa systemu Android. Stanowi ono abstrakcyjną warstwę pomiędzy oprogramowaniem, a zasobami sprzętowymi. Jądro zapewnia usługi systemowe takie jak zarządzanie pamięcią, zarządzanie procesami, obsługa stosu sieciowego.

**Hardware Abstraction Layer (HAL)**

Warstwa ta zapewnia standardowe interfejsy, które udostępniają funkcjonalność sprzętu dla wyższych warstw framework’u Java. W tej warstwie znajduje się wiele bibliotek. Każda implementuje interfejs dla określonego typu sprzętu, takiego jak bluetooth czy wifi. Gdy API wyższej warstwy odwołuje się danego sprzętu, system Android wczytuje potrzebne biblioteki takiej operacji.

**Android Runtime**

Dla urządzeń, które pracują z systemem Android w wersji 5.0 lub wyższej, każda aplikacja uruchamia swój własny proces razem z własną instancją środowiska uruchomieniowego ART. Warstwa ta jest napisana w taki sposób, aby umożliwić uruchomienie wielu takich maszyn wirtualnych dla urządzeń z małą pamięcią. Maszyna wirtualna wykonuje pliki DEX. Jest to format kodu bajtowego zaprojektowany dla Androida, zoptymalizowany pod kątem jak najmniejszego zużycia pamięci. Wbudowane narzędzia kompilują kod, otrzymany przez kompilatory Javy, do formatu DEX, który z kolei może być uruchomiony na platformie Android.

**Native C/C++ Libraries**

Wiele komponentów i usług systemu takich jak wcześniej opisane HAL lub ART, są zbudowane na bazie kodu natywnego, który wymaga natywnych bibliotek napisanych w C i C++. Platforma Android zapewnia udostępnienie niektórych funkcjonalności natywnych bibliotek przez API framework’u Java.

**Java API Framework**

Wszystkie funkcjonalności oferowane przez system Android są dostępne dla deweloperów poprzez API napisane w języku Java. Te API tworzą podstawowe elementy wymagane do stworzenia systemu Android. Dzięki temu uproszczone jest używanie podstawowych komponentów i usług systemu, takich jak:

* **View System** – komponenty wizualne, które mogą być wykorzystane do tworzenia UI aplikacji.
* **Resource Manager** – zapewnia dostęp do nie-programistycznych zasobów. Między innymi pliki graficzne, layout’y oraz zdefiniowane ciągi znaków.
* **Notification Manager** – umożliwia aplikacjom wyświetlanie własnych powiadomień.
* **Activity Manager –** umożliwia zarządzanie cyklem życia aplikacji
* **Content Providers** – umożliwia aplikacjom dostęp do danych z innych aplikacji, na przykład Kontakty

**System Apps**

Android zawiera w sobie zestaw wbudowanych aplikacji systemowych takich jak email, kalendarz, komunikator SMS, przeglądarka internetowa. Aplikacje te nie posiadają specjalnego statusu wśród innych zainstalowanych przez użytkownika. Zatem inna aplikacja może zastąpić wbudowaną i stać się domyślną przeglądarką lub komunikatorem SMS.

### MVC i pochodne

MVC – Model-View-Controller, jest to wzorzec architektury oprogramowania. Pozwala on w prosty sposób oddzielić logikę od widoku, oraz w jasny sposób określić odpowiedzialność każdej z trzech warstw.

* **Model** – warstwa ta zawiera w sobie definicję wykorzystywanych klas oraz logikę biznesową aplikacji. Jest odpowiedzialna za komunikację z serwerem. Serializację danych, wysyłanie zapytań, następnie odbieranie odpowiedzi i deserializację danych. Model nie wie nic o innych warstwach, dlatego może być wykorzystywany w wielu kontekstach
* **View** – warstwa widoku, reprezentująca model. Tworzy ona interfejs użytkownika (UI), oraz komunikuje się z kontrolerem, gdy użytkownik używa aplikacji. W MVC im widok mniej wie o modelu, tym lepiej. Najlepiej, gdy nie wie nic o logice leżącej w modelu, wtedy staje się mniej z nią związany, a co za tym idzie bardziej rozszerzalny i łatwiejszy w modyfikacji. Inaczej mówiąc, widok nie powinien wiedzieć, co jest konsekwencją naciśnięcia przycisku, powinien wiedzieć tylko, który przycisk został naciśnięty i taką informację przekazać do kontrolera. W MVC na aplikacje Android’owe, widok zazwyczaj identyfikuje się z „*layout’ami*”, czyli XMLowymi plikami zawierającymi w sobie informacje, w jaki sposób, jakie elementy mają być rozmieszczone w widoku.
* **Controller** – jest to klej, który scala aplikację w jedno. Warstwa ta kontroluje, co dzieje się w aplikacji. Kiedy użytkownik kliknie przycisk, widok poinformuje kontroler, który przycisk został kliknięty. Następnie kontroler zadecyduje jakieś odwołać się do warstwy modelu w kontekście tego zdarzenia. Kiedy kontroler otrzyma informację zwrotną po wywołaniu modelu, może uaktualnić widok w odpowiedni sposób. W MVC na aplikacje Android’owe, kontroler zazwyczaj identyfikuje się z Activity.



Rys. 12 MVC w Androidzie

Kontroler z racji tego, iż jest ściśle związany z API androida jest trudny to testowania przy pomocy testów jednostkowych. Brak elastyczności i trudność utrzymania czystego kodu powoduje również fakt naturalnej zależności pomiędzy layout’em oraz activity, która wynika z architektury systemu Android. Jeśli zmienimy widok będziemy musieli zmienić również kontroler. Z pomocą przychodzą pochodne wzorca MVC, które sprawdzają się lepiej dla systemu Android. Istnieją dwie równie popularne pochodne MVC, jest to **MVP** oraz **MVVM**.

**MVP**

Model-View-Presenter – ten wzorzec zmienia zakres widoku, tak aby nie naruszać naturalnego związku pomiędzy layout’em i activity w Androidzie.

* **Model** – tak samo jak w MVC
* **View** – jedyną zmianą jest zmiana zakresu, który obejmuje widok. Warstwa ta zawiera teraz w sobie zarówno layouty jak i activities. Dobrą praktyką jest, aby activity implementowało interfejs widoku, który sami stworzymy. Wtedy warstwa prezentera, nie jest ściśle związana z daną implementacją widoku.
* **Presenter** – pełni bardzo podobną funkcjonalność jak kontroler w MVC. Główną różnicą jest fakt, iż nie jest związany z konkretną implementacją widoku, tylko z jego interfejsem. Ułatwia to testy jednostkowe oraz zwiększa elastyczność tej warstwy.

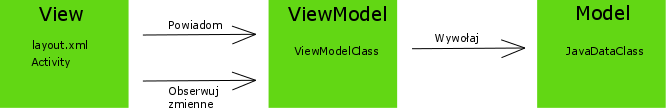
Rys. 13 MVP w Androidzie

MVP jest łatwe w testowaniu. W porównaniu do MVC, charakteryzuje się lepszym rozdzieleniem warstw aplikacji.

**MVVM**

Model-View-ViewModel – w tym wzorcu, odpowiednik kontrolera (ViewModel), nie informuje widoku o konieczności aktualizacji. To widok sam obserwuje ViewModel, i ustawia „*event*”, który reaguje na zmiany wartości obserwowanych pół i tym samym aktualizuje widok.

* **Model** – tak samo jak MVC
* **View** – obserwuje zmienne wystawione przez ViewModel i w zależności od nich zmienia widok
* **ViewModel** – warstwa odpowiedzialna za obsługę i odwołania do modelu. ViewModel aktualizuje obserwowalne wartości, które są wykorzystywane przez widok.

W MVVM przeprowadzanie testów jest jeszcze łatwiejsze, ponieważ nie ma powiązania pomiędzy odpowiednikiem kontrolera a widokiem. Dlatego podczas testów jednostkowych nie musimy „mockować” widoku.

Rys. 14 MVVM w Androidzie

**Wybór wzorca**

W implementacji aplikacji mobilnej, wybrano **MVVM**, ze względu na łatwość implementacji, wyraźną separację warstw, oraz brak ścisłej zależności pomiędzy nimi.

### Komunikacja z serwerem

Podczas komunikacji z serwerem protokołem HTTP, wykorzystano bibliotekę **OkHttp**, która tworzy klienta HTTP. Natomiast do obsługi formatu komunikacji pomiędzy klientem a serwerem, jakim jest JSON, wykorzystano bibliotekę **Gson**. Obie biblioteki są projektami open-source, opartymi na języku Java.

Podczas komunikacji z serwerem protokołem HTTP, wykorzystano następujące biblioteki:

* **OkHttp** – wykonuje niezbędne operacje do utworzenia klienta HTTP, oraz wywoływania serwisów o danym URL. Żądania i odpowiedzi do/z serwisów przyjmuje jako zwykły tekst – String format.
* **Gson** – wykonuje operacje serializacji obiektów Java do formatu JSON i odwrotnie. Biblioteka ta wykorzystana jest przy wysyłaniu żądań i odbieraniu odpowiedzi przez klienta HTTP, w wyżej wspomnianej bibliotece **OkHttp**.

## Aplikacja serwera

### C#/.NET

Aplikacja serwera została zaimplementowana przy użyciu języka C# na platformę .NET.

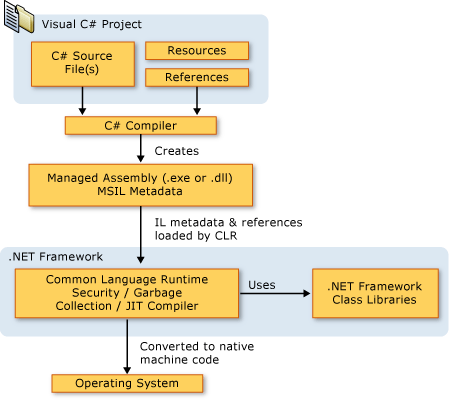
**Język C#**

C# jest wysokopoziomowym językiem programowania. Składnia jest podobna do języków takich jak C, C++ lub Java. Składnia C# umożliwia wykonanie wielu operacji w sposób o wiele prostszy niż byłoby to możliwe przy użyciu C++. Głównymi cechami C# są:

* **Obiektowość –** C# wspiera enkapsulacje, dziedziczenie i polimorfizm.  
  Klasa może dziedziczyć tylko po jednej klasie, może natomiast implementować wiele interfejsów. „Override’owanie” wirtualnych metod odbywa się z użyciem słowa kluczowego „*override*” aby uniknąć redefinicji.  
  C# umożliwia także tworzenie struktur, które są lżejszym odpowiednikiem klasy. Struktury mogą implementować interfejsy, jednak nie wspierają dziedziczenia.
* **Garbage Collector** – zarządzanie pamięcią przez środowisko uruchomieniowe. Dla programisty oznacza to brak konieczności samodzielnego przydzielania i zwalniania pamięci
* **Delegaty i Eventy** –odpowiedniki wskaźników na funkcję
* **Refleksje i Atrybuty** – umożliwiają analizę struktury klas i kodu z poziomu tego właśnie kodu.
* **Typy „nullowalne” –** możliwe jest przypisanie nulla do typów prostych takich jak Integer czy Bollean.
* **Typy generyczne**
* **Linq –** Language-Integrated Query
* **Właściwości, indeksery –** umożliwiają w prosty sposób, implementację dostępu do pól prywatnych w klasie. Innymi słowy określają gettery i setery dla danych pól.

**Platforma .NET**

Programy napisane w języku C#, działają na platformie .NET. Zawiera ona w sobie wirtualną maszynę, zwaną CLR – Common Language Runtime oraz określony zestaw bibliotek.  
Kod źródłowy napisany w C#, kompilowany jest do języka pośredniego. Kod tego języka, oraz zasoby takie jak stringi czy bitmaty są przechowywane na dysku jako pliki wykonawcze zazwyczaj z rozszerzeniem .exe lub .dll. Tego typu pliki, nazywane *assembly* zawierają informacje o typie assembly, kulturze, wersji, oraz wymaganiach bezpieczeństwa.  
Kiedy program jest wykonywany przez platformę .NET, CLR ładuje daną assembly, zgodnie z zawartymi w niej informacjami. Kiedy wszystkie wymagania są spełnione CLR wykonuje kompilację JIT – just in time, która konwertuje język pośredni do natywnego kodu maszynowego. CLR zapewnia również usługi takie jak Garbage Collector, zarządzanie zasobami czy obsługa wyjątków.



Rys. 15 Działanie platformy .NET [5]

Dodatkowo na platformie .NET mogą działać programy napisane nie tylko w języku C#. Wspiera ona również języki taki jak: C++/CLI, F#, J#, Delphi 8 oraz Visual Basic.

### Komunikacja z klientem

Do komunikacji z klientem użyto framework’u **ASP.NET WebAPI**, który wspiera tworzenie interfejsu komunikacyjnego w architekturze REST na platformę .NET.   
Serwer napisany w C# wystawia endpointy, poprzez które użytkownik odwołuje się do zasobów. Wymiana danych w komunikacji z klientem zachodzi w formacie JSON. Jest to format domyślny dla ASP.NET Web API. Aplikację od strony logicznej tworzą dwa główne komponenty:

* **Model** – przechowuje typy danych używanych w projekcie, oraz tworzy logikę biznesową. Ewentualnie komunikuje się z innymi warstwami jak na przykład bazą danych
* **Controller** – jest to warstwa wystawiająca endpointy do świata zewnętrznego. Odpowiada za przetwarzanie żądań klienta i generowanie odpowiedzi razem ze statusem danej operacji – Status code oraz Status message w protokole HTTP.

### Komunikacja z bazą danych

Do komunikacji z bazą danych użyto **Entity Framework**, ułatwia on komunikację z bazą danych. Entity Framework jest oparty na technologii ADO.NET. Programista może odwoływać się do bazy danych za pomocą obiektów, propercji i innych elementów charakterystycznych dla języka C#.

Istnieją dwie formy obsługi bazy danych:

* **Code-First** – jak sama nazwa wskazuje, baza danych tworzona jest na podstawie uprzednio napisanego kodu. Przy pomocy specjalnych atrybutów EF, tworzymy klasy, które następnie zostaną zmapowane przez framework do tabel w bazie danych. Mamy także możliwość określania relacji pomiędzy tabelami.
* **DB-First –** w tej metodzie odwołujemy się do istniejącej już bazy danych. EF wygeneruje odpowiednie klasy na podstawie istniejących już tabel i relacji między nimi w bazie danych.

Dzięki stosowaniu Entity Framework aplikacja jest łatwiejsza w implementacji, jest bardziej rozszerzalna, a także łatwiejsza w debuggowaniu. Dodatkowo dzięki LINQ – Language Integrated Query, możemy w prosty sposób odwoływać się do tabel w bazie danych, bez znajomości dokładnej składni zapytań SQL.

Dodatkowo, ponieważ zastosowana baza danych to MySQL Server, konieczne jest skonfigurowanie Entity Framework’a, aby komunikował się z nią w sposób prawidłowy, oraz aby używał prawidłowego protokołu.

## MySql

W projekcie, jako serwer bazy danych został użyty **MySQL Server**. Jest to open-sourcowy system zarządzania relacyjnymi bazami danych. MySQL został napisany w C oraz C++. Działa w wszystkich popularnych platformach systemowych. Dodatkowo jest dostępny w wersji źródłowej, co umożliwia przeprowadzenie kompilacji do innej, dowolnej platformy. Najważniejszymi cechami MySQL są:

* Wydajność
* Obsługa transakcji – opcjonalna
* Praca na maszynie wieloprocesorowej – wielowątkowość
* B-drzewa z kompresowanymi indexami
* Osadzenie serwera MySQL w aplikacji
* Wiele typów danych do wyboru w kolumnach
* Przejrzysty system zabezpieczeń
* Obsługa klauzuli agregujących i grupujących

# Implementacja systemu

## Sposób działania systemu

## Baza danych

Na potrzeby tego projektu została stworzona prosta baza danych, pozwalająca spełnić podstawowe założenia systemu. Składa się ona z dwóch tabel:

* **determinantmacids** – tabela ta zawiera dane o pomieszczeniach, w których zostały lub zostaną przeprowadzone pomiary mocy sygnału Wifi (**RSSI**).   
  Każde pomieszczenie ma swój klucz główny czyli, **Id**, jest ono bezpośrednio identyfikowane z danym pomieszczeniem. Dodatkowo dla danego pomieszczenia możemy określić nazwę i dodać ją w kolumnie **RoomName**.  
  Użytkownik dodając pomieszczenie do bazy danych, musi umieścić tam cztery wybrane przez siebie adresy mac, które posłużą, jako wyznaczniki do wykonywania pomiarów, oraz określania położenia, na podstawie pomiarów już dokonanych.
* **measurmentpoints** – tabela ta zawiera punkty pomiarowe, dla dostępnych pomieszczeń. Każdy z punktów pomiarowych ma swój klucz główny – **Id**, oraz klucz obcy – **RoomId –** łączący pomiar z pomieszczeniem, w którym został on wykonany. Każdy pomiar to zmierzenie mocy RSSI czterech sygnałów Wifi, w danym punkcie XY. Cztery sygnały Wifi są dla których wykonywane są pomiary, nawiązują do tych z tabeli **determinantmacids**.

Jeden pokój z określonymi access pointami reprezentowany przez tabelę **determinantmacids**, może mieć wiele pomiarów RSSI, reprezentowanych przez tabelę **measurmentpoints.** Zatem tabele tworzą relację Jeden-Do-Wielu. Baza danych została wygenerowała przez **Entity-Framework** – Code Fitst.

## Aplikacja serwera

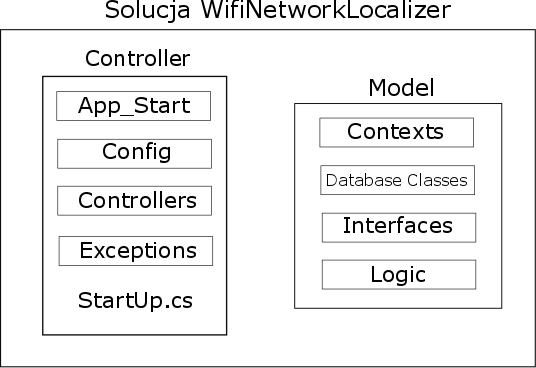
Serwer został zaimplementowany przy użyciu środowiska **Visual Studio**.   
Solucja serwera **WifiNetworkLocalizer**, składa się z dwóch projektów:

* Projekt Kontrolera – **Controller**
* Projekt Modelu – **Model**

Paczki „NuGet’owe”, które zostały zastosowane w projekcie to:

* AutoMapper - version=6.2.2
* EntityFramework - version=6.0.0
* Microsoft.AspNet.WebApi - version=5.2.3
* Microsoft.AspNet.WebApi.Client - version=5.2.4
* Microsoft.AspNet.WebApi.Core - version=5.2.4
* Microsoft.AspNet.WebApi.SelfHost - version=5.2.4
* Microsoft.AspNet.WebApi.WebHost - version=5.2.3
* Microsoft.Web.Infrastructure - version=1.0.0.0
* MySql.Data - version=6.9.11
* MySql.Data.Entity - version=6.9.11
* Newtonsoft.Json - version=6.0.4
* Unity.Abstractions - version=3.3.0
* Unity.AspNet.WebApi - version=5.0.13
* Unity.Container - version=5.7.0

### Struktura solucji



Rys. 16 Struktura solucji

**Model**

W projekcie modelu klasy zostały wydzielone do odpowiednich folderów, w celu zachowania porządku. Model odpowiada za bezpośrednią komunikację z serwerem bazy danych, tj. formułowanie zapytań oraz transferowanie danych do/z bazy danych. Jest to projekt typu ClassLibrary, oznacza to, że po skompilowaniu utworzy plik o rozszerzeniu .dll, który będzie mógł zostać ponownie użyty przez inny program wykonawczy.

**Contexts**

Folder ten zawiera kontekst bazy danych, używany do jej utworzenia – gdy nie istnieje, lub do pobierania/aktualizowania jej wartości. W kontekście wybieramy sposób inicjowania bazy danych, oraz klasy zawarte w folderze **Database Classes**, definiujemy tabele, jakie w niej występują. Folder ten zawiera jedną klasę:

* **WifiLocalizerContext.cs** – poprzez te klasę, Entity Framework tworzy opisaną wcześniej, bazę danych skonfigurowaną pod **MySql**.

**Database Classes**

Folder ten zawiera klasy, używane przez bazę danych, zarówno do definiowania tabel, jak i do realizowania zapytań oraz zwracania ich wyników. Klasy służące do definiowania tabel, jak i ich właściwości, są ozdobione specjalnymi atrybutami oferowanymi przez Entity Framework. Dzięki nim możemy skonfigurować proces tworzenia tabeli. Przykładem jest jawne określenie, która właściwość ma reprezentować klucz główny, a która klucz obcy, na podstawie atrybutów: **Key** oraz **ForeignKey**.   
Natomiast klasy służące do transferowania danych przy zapytaniach, to zwykłe klasy z polami o odpowiednich typach. Zawartość folderu:

* **DeterminantMacIds.cs** oraz **RSSIMeasurmentPoint.cs** – są to klasy służące bezpośrednio do stworzenia opisanych wcześniej tabel: **determinantmacids** i **measurmentpoints**
* **Point.cs, RoomInfo.cs, FourMacIds.cs, FourRSSISignals.cs** – to klasy służące do transferowania danych w komunikacji z bazą danych poprzez kontroler

**Interfaces**

Tutaj zawarty jest interfejs, który zawiera w sobie metody abstrakcyjne dające obraz funkcjonalności logiki serwera systemu:

* **ILocalization.cs** – interfejs deklarujący metody, które generują zapytania do bazy danych, a tym samym:

- zwracają dostępne pomieszczenia  
- obliczają najbliższy punkt XY, dla danej konfiguracji sygnałów RSSI  
- zwracają cztery pomiarowe adresy mac dla danego pomieszczenia  
- dodają nowe pomieszczenie z wymaganymi informacjami do bazy danych  
- dodają pomiar mocy sygnałów RSSI, w danych pomieszczeniu.

**Logic**

Folder ten zawiera implementację interfejsu znajdującego się w folderze **Interfaces**. Dodatkowo zawiera też klasę ułatwiającą obsługę bazy danych.

* **Localization.cs** – klasa implementująca interfejs **ILocalization**
* **DatabaseHandler.cs** – klasa pomagająca w obsłudze bazy danych. Zawiera w sobie generyczne metody, które mogą posłużyć do logowania na konsoli wykonywanych operacji. Metody te logują dodanie elementów do bazy danych oraz ich pobieranie. Przy każdej z tych operacji logowane są pola transferowanego obiektu.

**Controller**

W projekcie kontrolera klasy zostały wydzielone do odpowiednich folderów, w celu porządku. Kontroler odpowiada za wystawienie endpointów dla świata zewnętrzengo, poprzez które klient może odwoływać się do zasobów bazy danych protokołem HTTP.

**App\_Start**

Ten folder został wygenerowany przez framework **Unity**, który pomógł w zastosowaniu Dependency Injection, dla klas kontrolera. App\_Start zawiera dwie klasy:

* **UnityConfig.cs** – klasa ta określa konfiguację dla głównego kontenera **Unity** (inaczej mówiąc, kontenera **Dependency Injection** z funkcjonalnością frameworka **Unity**)
* **UnityWebApiActivator.cs –** klasa ta zapewnia integrację frameworka Unity z WebAPI, gdy jest hostowane w ASP.NET.

**Config**

Ten folder zawiera plik konfiguracyjny **MySqlConnectionsStrings.config**, który zawiera konfigurację, pozwalającą na połączenie się z serwerem bazy danych – **MySqlServer**.

**Controllers**

Tutaj, mieści się kontroler, który transferuje dane z/do bazy danych przy pomocy logiki oferowanej przez jeden z interfejsów zawartych w folderze **Interfaces** w projekcie **Model**.

* **LocalizationController.cs** – jest to klasa dziedzicząca po **ApiController**, który jest klasą oferowaną przez ASP.NET WebAPI. Wstrzykiwaniem zależności poprzez konstruktor, **LocalizationController** otrzymuje implementację **Mapper’a** oraz interfejsu **ILocalization**. Mapper służy do mapowania obiektów, dwóch różnych klas pomiędzy sobą. Natomiast implementacja **ILocalization**, udostępniona przez folder **Logic** projektu **Model**, zapewnia dostęp do logiki komunikującej się z bazą danych. **LocalizationController** określa metodę HTTP oraz **Route**, dla każdego serwisu, czyli adres **URL**, poprzez który wywoływana jest dana operacja.

**Exceptions**

Folder ten zawiera informacje dotyczące wyjątków, wyrzucanych w trakcie działania systemu. Na potrzeby projektu zawiera on tylko jedną klasę:

* **CustomExceptionLogger.cs** – klasa ta dziedziczy po klasie **ExceptionLogger**, która jest klasą domyślną do obsługi wyjątków dla mechanizmu obsługi komunikacji HTTP. **CustomExceptionLogger** „override’uje” metodę logowania błędów, która teraz będzie logowała każdy błąd wyrzucony poprzez logikę wołaną przez kontroler, na konsoli, kolorem czerwonym.

Ostatnim elementem jest klasa „rootująca” – **StartUp.cs**. Klasa ta jest odpowiedzialna za uruchamianie serwera HTTP. Bezpośrednio przed uruchomieniem:

* Wykonuje konfigurację **Entity Framework’a** do komunikacji z bazą danych MySql
* Określa port, na którym serwisy będą dostępne
* Konfiguruje **Mapper**
* Przygotowuje **UnityContainer**– kontener IoC (Inversion of Control), który ułatwia wstrzykiwanie zależności. W przypadku tego projektu, jest określane, która implementacja interfejsu **ILocalization,** oraz interfejsu **IMapper** ma zostać wstrzyknięta przez konstruktor kontrolera

### REST API

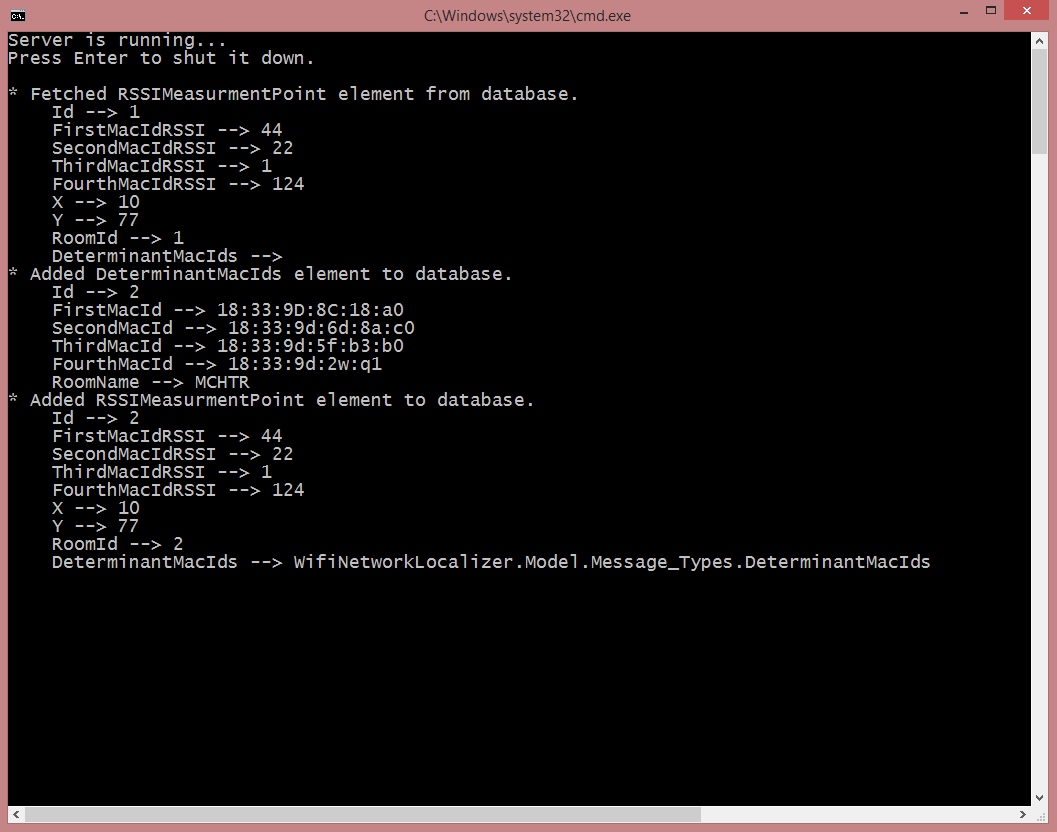
Poniżej przedstawiono zasoby, stworzone na potrzeby projektu, do których klient może odwoływać się za pomocą metod HTTP.

1 Tabela zasobów do których może odwoływać się klient

|  |  |  |
| --- | --- | --- |
| **Metoda HTTP** | **Zasób** | **Operacja** |
| GET | http://localhost:1471/localization/rooms | Pobranie dostępnych pomieszczeń |
| GET | <http://localhost:1471/localization/rooms>/{roomId} | Pobranie 3 pomiarowych adresów MAC dla danego pomieszczenia |
| POST | <http://localhost:1471/localization/rooms> | Dodanie nowego pokoju do bazy danych |
| POST | <http://localhost:1471/localization/rooms>/{id}/point | Dodanie nowego pomiaru RSSI w określonym punkcie XY dla danego pomieszczenia |
| GET | http://localhost:1471/localization/rooms/{id}/point ?firstMacId=80&secondMacId=-61&thirdMacId=-57&fourthMacId=-33 | Pobranie najbliższego punktu XY w danym pokoju, dla określonych mocy RSSI |

Wartości w klamrach muszą zostać uzupełnione przez klienta, zgodnie z faktycznym ich stanem w bazie danych.

### Uruchomienie aplikacji

Uruchomiona aplikacja działa w konsoli, logując na bieżąco, przebieg operacji, wykonywanych na bazie danych. Poniżej przedstawiono zrzut ekranu, prezentujący uruchomiony serwer.

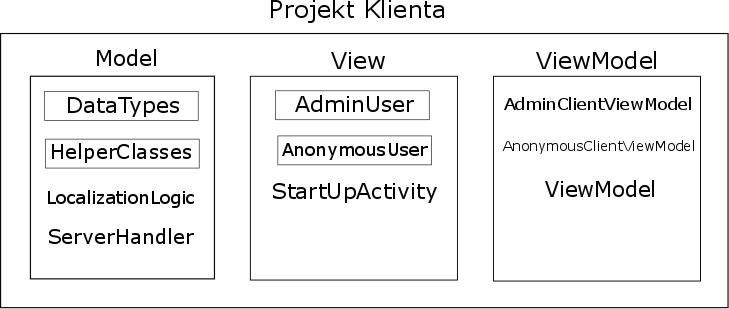
Rys. 17 Konsola reprezentująca uruchomiony serwer

## Aplikacja mobilna

Aplikacja klienta została zaimplementowana przy użyciu środowiska Android Studio. W projekcie. Jako wzorzec architektury zastosowano **MVVM**.

### Struktura projektu

Projekt został podzielonych na trzy główne „package”, które z kolei dzielą się dalej na bardziej szczegółowe.



Rys. 18 Struktura projektu klienta

**Model**

**DataTypes**

Ten package zawiera klasy, których obiekty służą do transferu danych w aplikacji. Są to klasy: **Point, MeasurmentPoint, RoomInfo,ThreeMacIds, ThreeRSSISignals**.

**HelperClasses**

Tutaj zawarte są klasy pomocnicze:

* **HttpCommunicationHandler** – klasa ta zapewnia komunikację HTTP z serwerem, przy pomocy biblioteki **OkHttpClient**.
* **WifiHandler –** klasa ta zawiera logikę, zapewniającą obsługę modułu Wifi. Pozwala pobrać dane o dostępnych sygnałach Wifi, takie jak: **SSID, RSSI, BSSID**.

**ServerHandler**

Jest to SingleTon, zrealizowany przy użyciu enuma – rozwiązanie możliwe od Java 5. Zawiera on w sobie metody obsługujące odwoływanie się do zasobów oferowanych przez serwer, protokołem HTTP. W nich następuje serializacja transferowanych obiektów z lub do formatu JSON, przy pomocy biblioteki **Gson**.

**LocalizationLogic**

Klasa ta, łączy funkcjonalność klasy **ServerHandler** z klasami z folderu **HelperClasses**. Zawiera w sobie metody, które np. filtrują odpowiednio otrzymane informacje, lub obliczają średnią z trzech kolejnych pomiarów i wysyłają je do bazy danych.

**View**

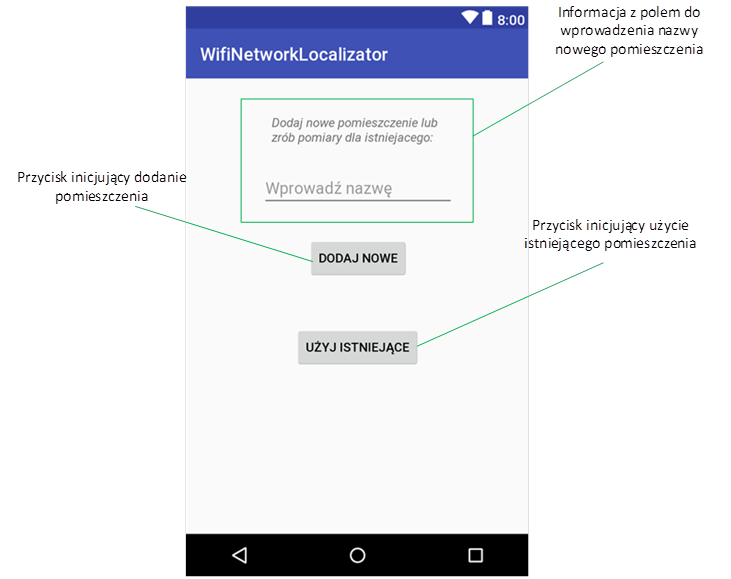
Tutaj znajdują się klasy definiujące jak mają zachowywać się Activity.



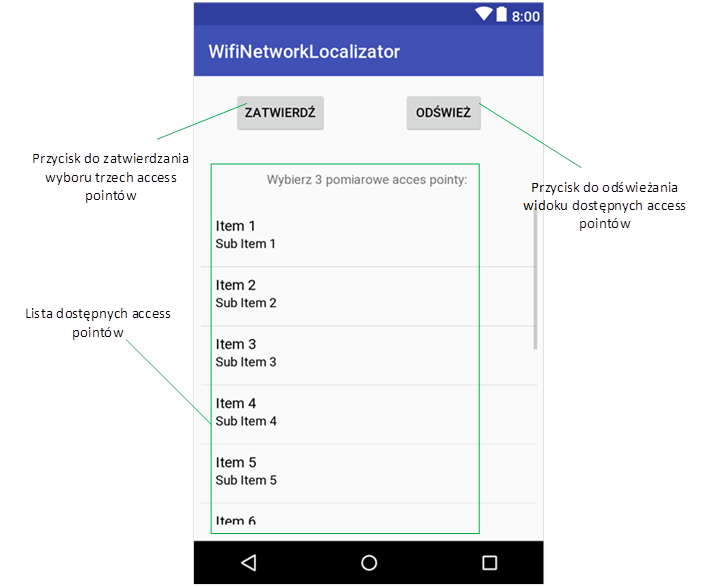
Rys. 19 Activity pojawiające się jako pierwsze po włączneiu aplikacji. Mamy do wyboru dwa tryby. Pierwszy - lokalizacji przekieruje nas do ścieżki AnonymousUser. Natomiast drugi tryb - tryb zarządzania, przekieruje nas do ścieżki AdminUser.

**AdminUser**

Tutaj znajdują się Activity, ukazujące się po wybraniu trybu zarządzania.



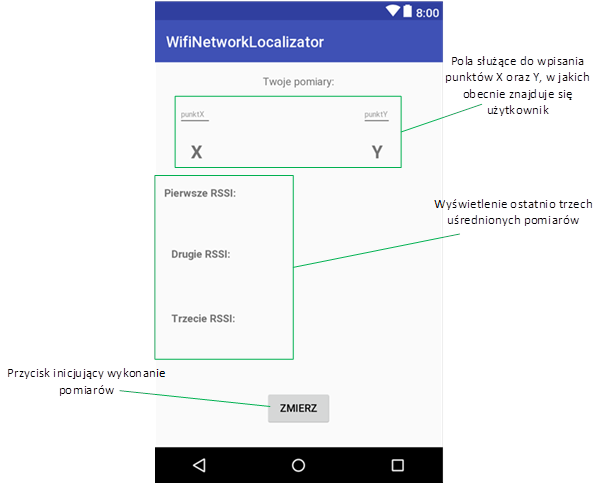
Rys. 20 Activity do wyboru, czy chcemy wykonywać pomiary na istniejącym pomieszczeniu, czy może chcemy stworzyć nowe. Wtedy musimy określić jego nazwę.



Rys. 21 Activity do wyboru 3 pomiarowych acces pointów, które będą charakterystyczne dla danego pomieszczenia.



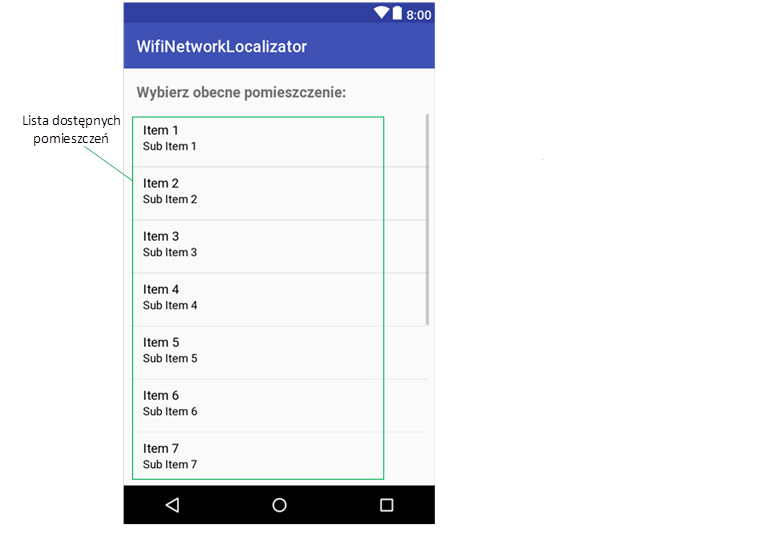
Rys. 22 Activity do wyboru pomieszczenia z listy dostępnych pomieszczeń, pobranych z serwera.



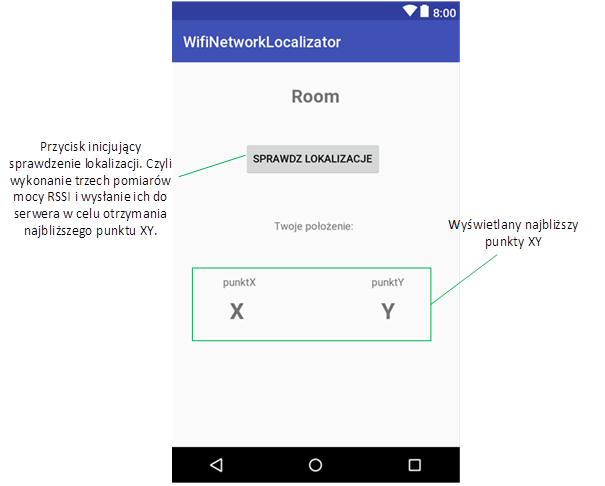
Rys. 23 Activity służące do wykonywania pomiarów. Należy tutaj określić punkt w jakim się obecnie znajdujemy, a następnie kliknąć przycisk ZMIERZ.

**AnonymousUser**

Tutaj znajdują się Activity związane z trybem lokalizacji.



Rys. 24 Tak jak w przypadku AnonymousUser, Activity służące do wybrania pomieszczenia obecnego pomieszczenia z dostępnych w bazie danych



Rys. 25 Activity służące do wyznaczenia najbliższego ze znanych punktów w bazie danych. Po naciśnięciu przycisku, system wykona odpowiednie pomiary i wyśle zapytanie do serwera.

**ViewModel**

Tutaj zawarte są klasy, do których odwołują się Activity.

* **AdminClientViewModel –** klasa ta zawiera pola oraz metody, do których odwołują się Activity z package **View.AdminUser**. Są to między innymi metody do:

- pobierania dostępnych access pointów  
- tworzenia nowego pokoju  
- sprawdzania unikalności nazwy  
- wykonywania pomiarów

* **AnonymousClientViewModel –** w tej klasie zawierają się pola i metody, do których odwołują się Activity z package **View.AnonymousUser**. Między innymi:

**-** pobranie najbliższego punktu o podobnym wektorze sygnałów  
- pobranie dostępnych pokoi

* **ViewModel** – jest to interfejs, który implementują dwie poprzednie klasy. Zawiera on podstawowe, metody które mogą być używane przez każde Activity, czyli: OnCreate, OnPause, OnResume, OnDestroy

# Testy

# Wnioski

Patrząc na wyniki testów, można stwierdzić, iż aplikacja działa poprawnie, oraz spełnia swoje założenia.

Główną zaletą stworzonego systemu jest brak konieczności znajomości infrastruktury danego budynku do wykonywania pomiarów w trybie zarządzania. Użytkownik chcąc dodać nowy budynek do bazy danych, nie musi wiedzieć gdzie rozlokowane są w nim access pointy, wystarczy, że odbiera ich sygnał. Następnie wykonuje serie pomiarów, przesyłając je na serwer.

System jednak cechuje się małą dokładnością. Może być to spowodowane przez czynniki takie jak:

* Zbyt duża niepewność pomiaru mocy RSSI sygnału Wifi przez urządzenie mobilne
* Zbyt duże wahania mocy sygnału
* Zjawisko FADING !!! OPISZ TO
* Poleganie tylko na trzech access pointach

Ze wszystkich wymienionych czynników najbardziej prawdopodobnym wydaje się ostatni. Gdyby implementowany system polegał na większej liczbie access pointów niż trzy, system zyskałby na dokładności jak i wydajności.

# Bibliography

|  |  |
| --- | --- |
| [1] | Kotaru, Manikanta; Joshi, Kiran; Bharadia, Dinesh; Katti, Sachin, „SpotFi: Decimeter Level Localization Using WiFi”. |
| [2] | A. Marcaletti, M. Rea, D. Giustiniano, V. Lenders i A. Fakhreddine, „Filtering Noisy 802.11 Time-of-Flight Ranging Measurements”. |
| [3] | „Developer Mozilla,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. |
| [4] | „Android Developers,” [Online]. Available: https://developer.android.com/guide/platform/index.html. |
| [5] | „Microsoft,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. |
| [6] | „Microsoft TechNet,” [Online]. Available: technet.microsoft.com/en-us/library/cc958821.aspx. |
| [7] | „Kemp technologies,” [Online]. Available: https://kemptechnologies.com/emea/solutions/http2/. |