POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

WYDZIAŁ INFORMATYKI

KATEDRA SIECI KOMPUTEROWYCH

SIECI URZĄDZEŃ MOBILNYCH

Patryk Ptasiński
Numer albumu 10623

Nawigacja w warunkach miejskich przy użyciu danych o lokalizacjach sieci WIFI

Praca inżynierska
dr. inż Michał Tomaszewski

Warszawa, czerwiec 2017

Spis treści

Streszczenie

Praca ta opisuje proces wytworzenia aplikacji internetowej oraz akompaniującej aplikacji mobilnej na platformę Android. W pracy omówione zostały problemy i wyzwania przy zbieraniu, przechowywaniu i analizowaniu informacji o lokalizacjach sieci bezprzewodowych. Na zakończenie skonfrontuję czy Warszawa posiada wystarczające zagęszczenie sieci bezprzewodowych oraz, czy możliwa jest nawigacja przy wykorzystaniu smartfona z systemem operacyjnym Android na podstawie lokalizacji sieci bezprzewodowych?

Wprowadzenie

Skąd pomysł na badanie tematu sieci bezprzewodowych? Technologia ta już rozprzestrzeniła się na całym świecie z ogromnym sukcesem. Więc można by postawić tezę, że w tym obszarze nie czeka na nas nic interesującego. Nic bardziej mylnego! Dopiero, teraz gdy sieci bezprzewodowe się rozprzestrzeniły na skalę masową, to możemy badać oraz projektować i tworzyć rozwiązania wykorzystując popularność tej technologii. W zasięgu przeciętnego mieszkania coraz częściej spotykamy się z dziesiątkami, a czasem nawet z setkami dostępnych w zasięgu sieci bezprzewodowych. Gdyby zapisać lokalizację tych wszystkich sieci bezprzewodowych, to utworzymy system, który umożliwia użytkownikom względne pozycjonowanie. W mojej pracy postaram się odpowiedzieć na pytanie, jak zbudować i korzystać z takiego systemu.

Rozdział 1

Część teoretyczna

W tym rozdziale zostaną przedstawione technologie, proces projektowania aplikacji internetowej oraz aplikacji mobilnej. Odwołam się do teorii, na których podstawie zostaną zaimplementowane algorytmy w obydwu aplikacjach. Na koniec postawię tezę, którą zweryfikuję w części praktycznej.

1.1 Sieć bezprzewodowa

To rozwiązanie technologiczne, które przy użyciu fal elektromagnetycznych (promieniowanie radiowe) umożliwia komunikację między urządzeniami. Dzięki takiemu rozwiązaniu możliwe jest tworzenie lokalnych sieci komputerowych bez kosztownego prowadzenia przewodów w ścianach czy podłogach. Technologia bezprzewodowych sieci komputerowych jest opisana przez IEEE 802.11™ - zestaw specyfikacji dla MAC oraz warstwy fizycznej (PHY). Specyfikacja dopuszcza częstotliwości takie jak 900 MHz oraz 2.4, 3.6, 5 i 60 GHz. Tablica 1.1

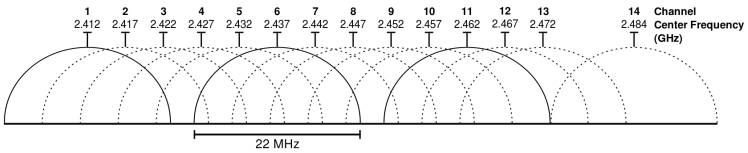
Stacją bazową bądź też punktem dostępu (z ang. access point) nazywamy urządzenie, które pracuje w trybie *Infrastructure*. Administrator punktu dostępu wybiera, na jakim kanale ma on operować. Aby można było się podłączyć do stacji bazowej, wykorzystywane są specjalne pakiety typu *Beacon frame* do rozgłaszania parametrów sieci bezprzewodowej. W takim pakiecie znajdują się in-

formacje między innymi o:

- Sposobie szyfrowania sieci
- Nazwie sieci (SSID)
- Możliwych prędkościach komunikacji ze stacją

Możliwe jest *ukrycie* sieci bezprzewodowej poprzez wyłączenie rozgłaszania sieci. Aby połączyć się z taką siecią, trzeba podać jej nazwę (SSID) oraz hasło (jeżeli dana sieć używa szyfrowania).

Najczęściej spotykaną konfiguracją sieci jest układ gwiazdy — jedna stacja bazowa, do której łączy się wiele urządzeń klienckich. Taki układ możemy spotkać w większości polskich domów oraz w małych firmach, gdzie jedna stacja bazowa jest wystarczająca do komunikacji bezprzewodowej na danej przestrzeni.



Rysunek 1.1: Graficzna reprezentacja rozmieszczenia kanałów sieci bezprzewodowych w paśmie 2.4GHz

Specyfikacja IEEE 802.11 opisuje również czym są kanały w sieciach bezprzewodowych. Kanałami nazywamy częstotliwości, na których mogą operować bezprzewodowe karty sieciowe. Dla częstotliwości 2.4GHz tych kanałów jest 14 i są one rozmieszczone co 5 MHz, zaczynając od 2412 MHz dla kanału 1 i kończąc na 2472 MHz dla kanału 13. Możemy jeszcze wyróżnić kanał 14, który jest dostępny tylko w Japonii i tylko dla specyfikacji opisanej w IEEE 802.11b. Znajduje się on na częstotliwości 2484 MHz. Dodatkowo kanały 12 i 13 są niedostępne w Ameryce Południowej. Warto jednak zauważyć, że do transmisji danych

potrzebne jest pasmo o szerokości 20MHz dla 802.11g/n lub 40MHz dla 802.11n co powoduje znaczne zmniejszenie ilości dostępnych kanałów, jeżeli nie chcemy mieć zakłóceń wynikających z pracy innej sieci bezprzewodowej na części pasma naszej częstotliwości.[4]

Jak wcześniej wspomniałem, technologia sieci bezprzewodowych korzysta z promieniowania radiowego — w częstotliwości określanej jako mikrofale:[14]

W zależności od długości fali radiowej jej propagacja zależy od różnorodnych zjawisk falowych np. dyfrakcji, refrakcji, odbicia, załamania, przenikanie np. od jonosfery itp.[14]

1.2 Geolokalizacja

W Słowniku Języka Polskiego możemy przeczytać, że to *ustalenie pozycji geograficznej lub adresu jakiegoś miejsca, lub osoby poprzez wykorzystanie GPS, lub adresu IP*[26]. Dokładniej i precyzyjniej pojęcie to zostało opisane w najpopularniejszym źródle wiedzy — Wikipedii. Znajdziemy tam listę *Sposoby wyznaczania pozycji*, w które znajduje się podpunkt *Pozycjonowanie względne — na podstawie widoczności innych obiektów o znanej pozycji (np. stacji bazowych przez komórkę czy ruterów Wi-Fi przez urządzenie). Ten sposób jest szczególnie istotny, jeśli urządzenie nie ma włączonego odbiornika GPS (oszczędność energii) lub w ogóle go nie posiada (np. laptop).*[15]. Podpunkt ten idealnie opisuje pojęcie geolokalizacji na potrzeby tej pracy.

Warto jednak wspomnieć, że niezależenie od sposobu, w jaki określamy pozycję lokalizowanego urządzenia, to lokalizowanie zawsze jest na jakimś poziomie dokładności. Na podstawie wniosków z badań Dr. Zandbergen-a z 2011 roku, możemy przyjąć dokładność odbiornika GPS zamontowanego w smartfonie z systemem operacyjnym Android na poziomie od 5 do 8 metrów.[28]

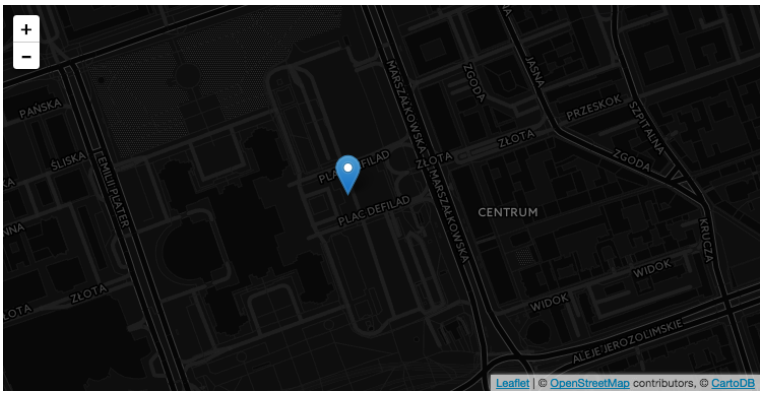
Tablica 1.1: Podział pasma radiowego

Nazwa fal	Skrót	Częstotliwość	Długość	Nazwa angielska	Skrót z ang.
		3-30 Hz	100 tys.-10 tys. km	Extremely low frequency	ELF
		30-300 Hz	10 tys.-1 tys. km	Super low frequency	SLF
		300-3000 Hz	1000-100 km	Ultra low frequency	ULF
fale myriametrowe, fale bardzo długie		3-30 kHz	100-10 km	Very low frequency	VLF
fale kilometrowe, fale długie	Dł, DF, D	30-300 kHz	10-1 km	Low frequency	LF
fale hektometrowe, fale średnie	Śr, ŚF, Ś	300-3000 kHz	1000-100 m	Medium frequency	MF
fale dekametrowe, fale krótkie	KF, KR, K	3-30 MHz	100-10 m	High frequency	HF
fale metrowe, fale ultrakrótkie	UKF	30-300 MHz	10-1 m	Very high frequency	VHF
mikrofale					
fale decymetrowe	VKF	**300-3000 MHz**	1000-100 mm	Ultra high frequency	UHF
fale centymetrowe		**3-30 GHz**	100-10 mm	Super high frequency	SHF
fale milimetrowe		**30-300 GHz**	10-1 mm	Extremely high frequency	EHF
fale submilimetrowe (fale terahercowe, promieniowanie terahercowe)		300-3000 GHz	1000-100 μm	Tremendously high frequency	THF

Przy dokładności geolokalizacji, warto wspomnieć, że ważnym jest jej zastosowanie. Nie wszystkie potrzeby geolokalizowania wymagają dokładności na wysokim poziomie. Niektóre aplikacje potrzebują informacji jedynie o kraju lub mieście danego urządzenia — użytkownika. Inne potrzebują bardzo dokładnej informacji — w przypadku nawigowania w pomieszczeniach budynków np. muzea, targi Expo.

1.2.1 Geografia

Przykładem dla tej pracy jest miasto Warszawa. Na potrzeby późniejszych rozdziałów, przyjąłem jako punkt centralny Warszawy z Wikipedii[21] jako współrzędne geograficzne w formacie DM S (Stopnie:Minuty:Sekundy) - 52°13’ 56″ N, 21°0’ 30″ E — lub odpowiadające im w formacie DM F (*Stopnie Setne zwane miaro miernymi*)[23] **52.232222, 21.008333**. Wypada on na środku Placu Defilad, od wschodniej strony Pałacu Kultury.1.2



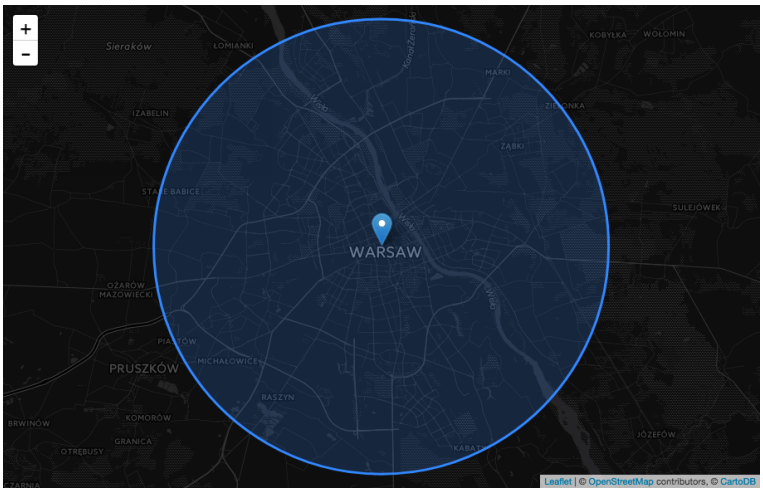
Rysunek 1.2: Prezentacja punktu o współrzędnych 52.232222, 21.008333 - centrum Warszawy - na mapie

Za obszar do obliczenia powierzchni Warszawy przyjąłem koło o podanym wcześniej punkcie centralnym i promieniu obliczonym na podstawie powierzchni Warszawy.[21] Takie obliczenia powinny być wykonywane na przybliżonym kształcie Ziemi — elipsoidzie. Jednak ze względu na statystyczne przeznaczenie do-

kładności potrzebnych informacji, przyjmuję, że kształtem obszaru tych obliczeń jest płaszczyzna. Więc wzór wygląda następująco:

$$A = \pi * r^2$$

Gdzie A to powierzchnia Warszawy, a natomiast r to szukany promień okręgu. Po odpowiednim przekształceniu wzoru i późniejszym podstawieniu *517,24 km^2 (1.01.2016)*[21] pod symbol A, otrzymujemy promień — r - o wielkości 12,8313km.



Rysunek 1.3: Prezentacja koła o promieniu 12,8313km o środku w punkcie o współrzędnych 52,232222, 21,008333 na mapie

1.2.2 Statystyka

Chcąc szacować ilość sieci bezprzewodowych w Warszawie, trzeba zacząć od ludności. Na stronie internetowej Urzędu Statystycznego w Warszawie stan ludności wynosił 1 753 977 osób (stan na dzień 31.12.2016). Dodatkowo w Warszawie funkcjonowało 424 195 przedsiębiorców. Posiłkując się informacją o przeciętnej liczbie osób w gospodarstwie domowym w województwie mazowieckim w kategorii miasta — wynosił 2,42.[24] Dzieląc ludność przez ilość przeciętnej ilości osób w gospodarstwie domowym, dostaniemy 724 784 gospodarstwa

domowe w Warszawie. Na podstawie tych wartości możemy szacować, że 395 350 (93,2%)[25] przedsiębiorców oraz 548 662 (75,7%) gospodarstwa domowe, które posiadają łącza szerokopasmowe, prawdopodobnie również posiadają przynajmniej jedno urządzenie oferujące sieć bezprzewodową. Daje to końcowy szacunek w okolicach **944 012** punktów bezprzewodowego dostępu do internetu.

1.2.3 Na podstawie sieci bezprzewodowych

W przypadku algorytmu estymującego lokalizację użytkownika na podstawie zarejestrowanych sieci bezprzewodowych musimy rozważyć wiele czynników takich jak:

- urządzenia wypożyczone konsumentom — dostawcy usług telekomunikacyjnych nierzadko przywracają sprawne urządzenia do kolejnego klienta bez zmiany identyfikatorów sprzętowych
- mobilne hotspoty — smartfony mają możliwość tworzenia punktów dostępu dla innych urządzeń. Są też dedykowane rozwiązania rozdawane do abonamentów przez operatorów telekomunikacyjnych, takie jak Huawei MIFI.
- migracja ludności — na własnym przykładzie mogę powiedzieć, że od 3 lat mam ten sam Punkt Dostępu, a od kiedy go zakupiłem, przeprowadziłem się już 2 razy. Wielokrotnie miałem sytuacje, w których estymowana lokalizacja z Google, pokazywała okolice mojego poprzedniego zamieszkania. Szczególnie przy drugiej przeprowadzce.

1.3 Aplikacja internetowa

Natomiast rolą aplikacji internetowej będzie geolokalizacja na podstawie podanych adresów MAC oraz siły sygnału widocznych sieci. Miejsce zapisu zebra-

nych danych, import dużych danych, wizualizacja, analiza, zapytania na bazie

1.3.1 Ruby

Ruby jest obiektowo-funkcyjnym interpretowanym językiem. Posiada zaimplementowane odśmiecanie pamięci (z ang. Garbage Collector). *Ruby bazuje na wielu językach, takich jak CLU, Eiffel, Lisp, Perl, Python czy Smalltalk. Składnia jest zorientowana liniowo i oparta na składni CLU oraz, w mniejszym stopniu, Perla.*[19]

Ruby został stworzony przez Yukihiro Matsumoto w 1995 roku. Od tego czasu doczekał się wielu poprawek i optymalizacji. Jest to też jeden z niewielu języków, który posiada wiele interpreterów spośród których najbardziej popularnymi implementacjami obecnie są:

- MRI (Matz's Ruby Interpreter) - jest wyjściową implementacją i jest de facto specyfikacją języka
- JRuby - w większości napisany w Javie. Osiągnął dosyć dużą popularność, ze względu na dobrą zgodność z oryginalną specyfikacją przy większej wydajności.
- Ruboto - Nakładka na JRuby, która pozwala tworzyć aplikacje na Androida.
- RBX (Rubinus) - Oparty na projekcie maszyny wirtualnej Smalltalk-80
- RubyMotion - komercyjna implementacja na iOS, Mac OS X oraz Android

Wiele implementacji interpreterów oraz przejrzysty kod, który czyta się prawie jak naturalny język angielski, spowodowały, że w latach 2008-2012 był najpopularniejszym językiem na GitHub.[7] Niestety w kolejnych latach jego popularność spadała. W 2013 był już na drugim miejscu, w latach 2014 oraz 2015 był

na trzecim miejscu[7], w 2016 plasował się na szóstym miejscu i tak pozostaje do dziś.[6]

Ruby on Rails

Ruby on Rails jest zestawem bibliotek, które tworzą szkielet (z ang. framework) do budowania aplikacji internetowych zgodnie ze wzorcem architektonicznym Model-Widok-Kontroler (MVC) oraz z paradygmatem projektowym (z ang. design paradigm) Konwencja-Ponad-Konfiguracją (z ang. Convention-Over-Configuration). Kolejną ideą jest *Nie powtarzaj się* (z ang. Don't Repeat Yourself) - sednem tej idei jest modularyzacja i poprawna strukturyzacja kodu. Dzięki takiemu podejściu kod aplikacji jest łatwiej czytać, modyfikować i testować.

Połączenie dużej refleksyjności Rubiego, świetnej biblioteki ORM, czystej i zorganizowanej struktury plików, która dzięki konwencji, w każdym projekcie jest zawsze taka sama stworzyło jedno z najbardziej lubianych i najlepiej ocenianych pod względem produktywności środowisk do wytwarzania aplikacji internetowych.

Ze względu na bardzo kosztowną naturę powstawania aplikacji internetowych opartych na rozwiązaniu Ruby on Rails, popularność tej technologii w morzu internetu jest niszowa.[27]

Tablica 1.2: Użycie Ruby on Rails w internecie

Kategoria	Liczba wykrytych z liczbą wszystkich	Procent
Top 10 tys. na postawie Quantcast	558 z 10 000	5,6%
Top 100 tys. na postawie Quantcast	3 415 z 100 000	3,4%
Cała baza builtwith.com	1,162,698 z 370,442,435	0,3%

Jak widać z Tablicy 1.2 z tego rozwiązania korzystają aplikacje internetowe, które są oceniane wysoko w rankingu Quantcast. Strony takie potrzebują wysokiej jakości dopasowanych do ich potrzeb rozwiązań. Oczywiście nie zaprzeczam, że

można korzystać z Ruby on Rails przy prostych potrzebach takich jak blogi z systemami zarządzania treścią lub przy kampaniach promocyjnych czy stronach produktów. Jednakże koszt powstania takiej strony internetowej jest niepotrzebnie duży. Przy takich potrzebach na rynku rozwiązań dużo lepszym rozwiązaniem byłby Wordpress, Joomla lub inny system do zarządzania treścią. Rozwiązania oparte na PHP mają mniejszy koszt infrastruktury w porównaniu do Ruby on Rails. Wsparcie techniczne i specjaliści są tańsi i łatwiej osiągalni.

1.3.2 HTML

Aplikacje internetowe nie istniałaby bez języka HTML. Ten już dość przestarzały język był wielokrotnie poprawiany i rozwijany na przestrzeni lat. Wciąż istnieje i nowoczesna jego odmiana w większości jest kompatybilna z poprzednimi wersjami.

1.3.3 JavaScript

Ze względu na ograniczenia interaktywności dokumentów HTML w pracy zostanie wykorzystany język JavaScript. W nowoczesnych aplikacjach internetowych jest to absolutnie normalne, że wykorzystuje się ten język do poprawienia użyteczności i ergonomiczności aplikacji. Pozwala tworzyć bardziej responsywne interfejsy użytkownika i wymieniać informacje z serwerem bez przeładowywania całej strony internetowej.

Język obecnie przeżywa drugie narodziny ze względu na wytworzony przez firmę Google nowy interpreter - V8 - który umożliwia uruchomienie JavaScriptu bez przeglądarki. Takie usamodzielnienie języka spowodowało wielki wybuch narzędzi konsolowych, wielu nowych bibliotek oraz szkieletów do budowania aplikacji (z ang. framework). Stał się nowoczesną technologią do wytwarzania aplikacji internetowych, ze względu na możliwość pisania kodu aplikacji po stro-

nie przeglądarki i po stronie serwera w jednym języku. Pozwala to na mniejsze wymagania wobec programistów i tym samym powoduje lepszą znajomość języka.

JSON

To specyfikacja opisująca składnię i format struktury danych. Zrodziła się na podstawie JavaScriptu i prawie cała jest wierną kopią notacji obiektowej tego języka.[17] Standard ten stał się nieodłącznym elementem przy komunikacji z interfejsami programistycznymi aplikacji internetowych, ze względu na większą czytelność w trakcie procesu rozwoju oprogramowania od rozwiązań takich jak XML czy HTML. Nie posiada zadeklarowanej struktury do przechowywania danych i między innymi dlatego, tak często się z niego korzysta przy wymianie prostych i małych pakietów informacji. Implementacja analizatorów składniowych oraz generatorów struktury do tego standardu znajduje się w większości języków programistycznych.

1.3.4 PostgreSQL

Źródłowo otwarty system zarządzania obiektowo-relacyjną bazą danych. Dla twórców ważną ideą jest zgodność ze standardami oraz możliwość rozszerzania funkcjonalności poprzez różne dodatki. Posiada implementacje na wszystkie najpopularniejsze systemy operacyjne takie jak Windows, Linux oraz Unix. Postgres (w skrócie od słowa PostgreSQL) dąży do pełnej zgodności ze standardem ACID.[18]

1.4 Aplikacja mobilna

Rolą aplikacji mobilnej będzie odpytywanie aplikacji internetowej o lokalizację na podstawie zeskanowanych sieci bezprzewodowych. Dodatkowo aplikacja

będzie umożliwiała obserwacje lokalizacji sieci bezprzewodowych i przesyłanie tych danych do aplikacji internetowej do zbadania zagęszczenia sieci bezprzewodowych na terenie Warszawy.

1.4.1 Android

Źródłowo otwarty system operacyjny tworzony przez firmę Google. Zaprojektowany z myślą o urządzeniach mobilnych takich jak smartfony i tablety. Głównymi narzędziami użytkownika są gesty na wyświetlaczu dotykowym, ale pomimo tego system posiada wsparcie dla klawiatur i myszek.[11]

1.4.2 Java

Choć rdzeń systemu Android jest napisany w C++, to aplikacje i interfejs użytkownika jest napisany w Javie.[11]

Java jest jednym z najpopularniejszych języków programistycznych na świecie. Według serwisu GitHut język ten jest na **drugim miejsce na całym świecie.**[6] Początkowym hasłem twórców tego języka było *Napisz raz, uruchom wszędzie* (z ang. *write once, run anywhere*). Cechuje się silnym typowaniem, obiektowością na podstawie klas oraz kompilacją do kodu bajtów, który jest uruchamiany na dedykowanej maszynie wirtualnej Javy (ang. Java Virtual Machine, w skrócie JVM).[22][16]

Dla systemu operacyjnego Android powstała specjalna wersja środowiska uruchomieniowego (z ang. runtime environment) — maszyny wirtualnej Javy — do uruchamiania programów napisanych w Javie. Dla wersji systemu Adroida równej 4.4 i wcześniejszych jest to Dalvik, dla późniejszych wersji jest to Android Runtime (w skrócie ART).[12][10]

1.4.3 XML

Gdy Android powstawał, standard XML był nieodłączną częścią Javy. Naturalne jest, że znalazł on miejsce jako pliki konfiguracyjne oraz pewnego rodzaju struktura definiująca w jaki sposób widok aplikacji systemu Android ma zostać zaprezentowany.

Standard ten jest coraz częściej wypierany przez JSON ze względu na kiepską czytelność i dużą pracochłonność przy ręcznym tworzeniu i modyfikowaniu struktury danych, ale mimo wszystko umiejętność pracy z XML jest nieodłącznym atrybutem programisty.

1.4.4 SQLite

Do tymczasowego przechowywania danych podczas zbierania informacji o lokalizacjach sieci bezprzewodowych, ta plikowa baza danych jest rozwiązaniem pasującym tutaj idealnie. W wielu aplikacjach Androidowych jest to podstawowy wybór, gdy pojawiają się relacyjne dane, które trzeba przechować lub filtrować.

1.5 Teza

Na podstawie przeprowadzonej analizy moja teza składa się z trzech części, które definiują kryteria, dla których nawigacja w warunkach miejskich może być skuteczna. Tezę uznam za potwierdzoną gdy spełnione zostaną wszystkich części w stopniu przynajmniej minimalnym.

1.5.1 Część pierwsza - precyzja geolokalizacji

Urządzenie z zainstalowaną aplikacją mobilną umożliwi precyzyjne określenie lokalizacji w warunkach miejskich. Poziom precyzji, który uważam za minimalny do skutecznej nawigacji pojazdu samochodowego przy prędkości nieprzekracza-

jącej 60 km/h to 50 metrów. Zmierzenie tej wartości zostanie wykonane przez porównanie pozycji z odbiornika GPS i z pozycją otrzymaną przez geolokalizację na podstawie sieci bezprzewodowych w zasięgu urządzenia.

1.5.2 Część druga - szybkość geolokalizacji

Urządzenie z zainstalowaną aplikacją mobilną umożliwi określenie lokalizacji z odpowiednią szybkością w warunkach miejskich. Szybkość określania geolokalizacji, którą uważam za spełniającą warunki do skutecznej nawigacji pojazdu samochodowego przy prędkości nieprzekraczającej 60 km/h to nie więcej niż 5 sekund między pełnymi cyklami. Z takiego cyklu składa się zlecenie skanowania sieci bezprzewodowych, asynchroniczne skanowanie, odebranie wyników skanowania oraz asynchroniczna geolokalizacja na podstawie wyników ze skanowania sieci bezprzewodowych. Wartość będzie mierzona na podstawie średniego czasu kolejno otrzymanych następujących po sobie wynikach geolokalizacji.

1.5.3 Część trzecia - zagęszczenie sieci bezprzewodowych na terenie Warszawy

Na terenie Warszawy zagęszczenie sieci bezprzewodowych powinno wynosić w okolicach 1 825 sztuk na kilometr kwadratowy. Taką wartość otrzymałem poprzez podzielenie **944 012** - szacowanej ilości sieci bezprzewodowych na terenie miasta Warszawa na podstawie statystycznych informacji o ludności, gospodarstwach domowych oraz informacji o dostępnie do internetu społeczeństwa — przez **517,24 km^2** - pole powierzchni miasta Warszawa. Oczywiście wartość ta będzie się różnić ze względu na różnorodność zabudowy Warszawy. Jednakże nie może być mniejsza niż 400 sieci bezprzewodowych na kilometr kwadratowy, bo oznaczałoby to dokładność geolokalizacji na poziomie 50 metrów lub więcej. Wartość będzie mierzona na podstawie zsumowania ilości unikalnych urządzeń

— punktów dostępu — na wybranych kilku obszarach o rozmiarze 1 km^2. Unikalność urządzeń zostanie ustalona na podstawie BSSID — adresów MAC — zeskanowanych przez aplikację mobilną oraz z publicznych zbiorów dotyczących lokalizacji sieci bezprzewodowych.

Rozdział 2

Część praktyczna

W rozdziale tym przedstawię proces utworzenia aplikacji internetowej oraz mobilnej. Omówię wyzwania oraz problemy, z którymi się spotkałem podczas implementacji. Opiszę też zasadę działania algorytmów potrzebnych do realizacji podstawowych założeń aplikacji mobilnej oraz internetowej. Na koniec przeprowadzę badania i wyciągnę wnioski, które skonfrontuje z postawioną tezą.

2.1 Przygotowanie

Aby rozpocząć prace nad utworzeniem aplikacji internetowej i mobilnej potrzebny jest warsztat sprzętowy oraz odpowiednie oprogramowanie zainstalowane i skonfigurowane na danym sprzęcie.

2.1.1 Sprzęt

Do wytworzenia aplikacji internetowej oraz mobilnej potrzebny jest komputer. W moim przypadku środowisko programistyczne zainstalowałem i skonfigurowałem na MacBooku Pro z 2011 roku. Do celów badawczych pracy potrzebny mi był smartfon z systemem operacyjnym Android. Wykorzystałem do tego zadania LG Nexus 5. Dodatkowo, jeżeli mamy smartfon z systemem Android to nie musimy konfigurować emulatora, co w przypadku mojego komputera mogłoby

bardzo ograniczyć zasoby pamięci RAM.

2.1.2 Oprogramowanie

Git

Korzystając z nowoczesnych narzędzi wytwarzania oprogramowania, naturalne jest korzystanie z systemu kontroli wersji. Osobiście mam doświadczenie z systemem Git. Aby korzystać z takiego systemu, potrzebujemy mieć zainstalowane narzędzie klienckie. Ponadto potrzebujemy mieć skonfigurowane repozytoria na odpowiednim serwerze. Ja do tego celu wykorzystałem darmowy plan na platformie Github. Utworzyłem dwa repozytoria, oddzielne dla aplikacji internetowej i aplikacji mobilnej.

Ruby

Spośród wielu implementacji oraz narzędzi do zarządzania środowiskiem uruchomieniowym Ruby wybrałem wersję 2.3.3, która na dzień tworzenia projektu była najnowsza. Do zarządzania wersją Ruby wykorzystuję narzędzie rbenv.

RubyMine

Mimo że Ruby jest językiem interpretowanym, a zestaw konsolowych narzędzi Ruby on Rails wykorzystuje się w sposób naturalny. Jednak osobiście bardzo lubię korzystać z tego edytora. Podświetlanie składni, refaktoryzacja kodu i integracja z Gitem powodują, że pisanie aplikacji internetowej nie jest toporne i żmudne. Wersja na dzień utworzenia projektu to 2017.1.

PostgreSQL

PostgreSQL na środowiska Mac OS można ściągnąć w postaci zwykłej aplikacji systemowej i tak też zrobiłem w swoim projekcie. Aplikacja zasugerowała naj-

nowszą wersję systemu bazy danych i na dzień utworzenia projektu oznaczenie wersji to 9.6.

Android Studio

Najprostszym sposobem wytwarzania oprogramowania dla systemów Android jest korzystanie z pakietu Android Studio. Najnowsza wersja na dzień utworzenia projektu to 2.3.2. W ramach instalacji tego środowiska programistycznego zostało zainstalowanych wiele bibliotek. Między innymi SDK Manager, w którym zainstalowałem SDK Platformy o oznaczeniu Android 7.1.1 (Nougat), której poziom API to 25.

Przeglądarka internetowa

Gdy mowa o aplikacji internetowej, to nie można zapomnieć o jej kliencie — przeglądarce internetowej. Przy rozwoju aplikacji opartej na Ruby on Rails, nie ma większego znaczenia, z jakiej przeglądarki skorzystamy. Osobiście jednak polecam, aby na komputerze posiadać przeglądarki kilku firm, ze względu na kompatybilność i różnice w wydajności przetwarzanych danych.

2.2 Aplikacja internetowa

Implementację rozpocząłem od importu istniejących danych o lokalizacjach sieci WIFI z różnych serwisów. Potem skupiłem się na prezentacji zaimportowanych danych. Na koniec stworzyłem ścieżkę, pod którą można geolokalizować na podstawie podanych adresów MAC i siły sygnałów podanych sieci bezprzewodowych.

Tablica 2.1: Rozmiary i format zrzutów baz danych z informacjami o lokalizacjach sieci bezprzewodowych

Adres serwisu	Rozmiar danych	Fromat
openwifi.su (openwlanmap.org)	245,7MB	CSV spakowany do tar.bz2
mylnikov.org	657 MB	CSV spakowany do zip
radiocells.org	1.6GB	baza danych sqlite

2.2.1 Struktura danych

W internecie można znaleźć trzy serwisy, które za darmo udostępniają zrzuty bazy z lokalizacjami bezprzewodowych sieci.

Na podstawie importowanych danych przygotowałem strukturę bazy danych obserwacji. Uwzględniłem też kolumny, które będą potrzebne przy zbieraniu obserwacji z aplikacji mobilnej. Tablica 2.2

Dodatkowym problemem stała się samo BSSID. W każdej z importowanych baz zostało zapisane w inny sposób. Okazało się, że w ramach importu bazy ze strony mylnikov.org sposób zapisu BSSID się różni. Czasem wszystkie litery były duże. Innym razem wszystkie były małe. Aby ujednolicić zapis, utworzyłem odpowiednią metodę klasową, którą potem wykorzystałem we wszystkich miejscach, skąd przyjmowałem informacje o BSSID.

```
def self.standardize_bssid(value) # value =
 ↪  '00:0A:E6:3E:FD:E1'
  result = value.to_s.upcase.gsub(/[^A-F0-9]/, '')
  return nil if result.size != 12
  result
end
```

Ze względu na optymalizację procesu importu obserwacji sieci bezprzewodowych zdecydowałem się zdenormalizować obserwacje punktów dostępów. Kolumna BSSID — opisująca adres sprzętowy urządzenia będzie posiadała powtarzające się dane. Dzięki temu mogłem użyć biblioteki activerecord-import, która

Tablica 2.2: Kolumny i typy przechowywania danych wraz z ich znaczeniem w aplikacji internetowej

Typ	Nazwa	Dodatkowe uwagi	Znaczenie (opcjonalne)
string	bssid	null: false	
string	ssid		
datetime	observed_at		Informacja, kiedy dana sieć została zaobserwowana
float	latitude	null: false	Szerokość geograficzna
float	longitude	null: false	Wysokość geograficzna
datetime	geolocated_at		Gdy urządzenie rozróżnia czas ostatniej pozycji GPS od czasu obserwacji sieci bezprzewodowej
string	source	default: "internal"	Nazwa źródła skąd przyszły dane. Jeżeli import to odpowiednia nazwa serwisu. Jeżeli z aplikacji mobilnej, to odpowiedni identyfikator urządzenia
string	id_of_source		Identyfikator w ramach pola source; Pozwoli na nie importowanie duplikatów lub możliwą aktualizację importowanych danych
json	raw_info		Wszystkie atrybuty, jakie były zadeklarowane do obserwacji. Jeżeli byśmy importowali jakieś dane, które nie mają odpowiedniej kolumny w bazie danych, to w przyszłości możemy te dane wyciągnąć z tej kolumny.
datetime	created_at		Data i czas utworzenia rekordu (domyślnie dodawane przez Ruby on Rails)
datetime	updated_at		Data i czas modyfikacji rekordu (domyślnie dodawane przez Ruby on Rails)

Tablica 2.3: Ilość i stosunek zaimportowanych danych o lokalizacjach sieci bezprzewodowych tylko dla regionu Warszawy

Zaimportowano	Wszystkich	Stosunek zaimportowanych do wszystkich	Źródło importowanych danych
32534	11260797	~3‰	radiocells.org
358916	37841215	~9‰	mylnikov.org
151365	15636591	~10‰	openwifi.su (openwlanmap.org)
łącznie			
542815	64738603	~8,385‰	-

pozwala na import danych do bazy danych poprzez bardzo praktyczny interfejs, ale przede wszystkim w sposób bardzo szybki. Dzięki tej decyzji import informacji o lokalizacjach sieci bezprzewodowych dla obszaru Warszawy zajął niecałe 6 godzin na moim komputerze. Warto tu wspomnieć, że zakres importowanych danych to mniej niż 1% z całego publicznego zbioru informacji o lokalizacjach sieci bezprzewodowych. Tablica 2.3

2.2.2 Prezentacja danych na mapie

Do prezentacji danych, jakimi są lokalizacje sieci bezprzewodowych, wykorzystałem gotową bibliotekę Leaflet. Wykorzystywanie gotowych elementów interfejsu często przyspiesza pracę nad projektami. Zwłaszcza tak skomplikowanego interfejsu, jakim jest mapa. Sama biblioteka implementuje obsługę współrzędnych geograficznych, przybliżania i oddalania oraz prostych znaczków i kształtów geograficznych. Podstawowym elementem, jaki musiałem dodać to grafiki mapy — tzw. płytki map (z ang. map tiles). Mój wybór padł na dostawcę darmowych grafik — serwis CartoDB. Udostępniają oni wysokiej jakości, schludne grafiki do wyboru w kilku odcieniach. Na potrzeby tej pracy wybrałem minimalistyczną i ciemną wersję o nazwie dark. Do Leafleta dodałem jeszcze dwa dodatki: jeden odpowiedzialny za pobieranie danych standardem AJAX oraz drugi, który jest

odpowiedzialny za klastrowanie dużej ilości znaczników na mapie w grupy.

Dla widoku mapy utworzyłem ścieżkę na serwerze, pod którą można pobrać znaczniki z lokalizacjami sieci bezprzewodowych. Zdecydowałem się na strukturę danych określoną w standardzie GeoJSON ze względu na kompatybilność z biblioteką Leaflet, dodatkowo stosując takie rozwiązanie, interfejs programistyczny jest bardziej przystępny do wykorzystania.

Ze względu na ogrom danych, jakimi są zapisane lokalizacje sieci bezprzewodowych, potrzebne jest wykorzystanie mechanizmu AJAX i ograniczenia przesyłanych danych do 25 tysięcy struktur GeoJSON. To ograniczenie wynika z wydajności przeglądarki do prezentacji tak dużej ilości danych. Zauważyć można było, że przeglądarka Firefox zawieszała się na kilka sekund podczas ładowania danych większych niż wcześniej opisane ograniczenie.

Samo generowanie struktury GeoJSON zaimplementowałem poprzez odpowiednie dwie metody:

```
def geojson_hash
  {
    type: 'Feature',
    geometry: {
      type: 'Point',
      coordinates: geojson_coordinates
    },
    properties: {
      id: id,
      bssid: bssid
    }
  }
end

def geojson_coordinates
  [longitude, latitude]
end
```

2.2.3 Algorytm geolokalizacji

Rozwiązanie tego problemu składa się z dwóch części. Pierwszą jest widok — perspektywa - w bazie danych, której zadaniem jest podanie lokalizacji dla najmocniej zarejestrowanego sygnału sieci bezprzewodowej. Dodatkowym elementem tego widoku jest sortowanie po roku oraz tygodniu, w którym sieć została zaobserwowana. Dzięki temu mamy pewność świeżości danych o lokalizacji sieci bezprzewodowej.

```
CREATE MATERIALIZED VIEW wifi_positions AS
  SELECT DISTINCT bssid, id, longitude, latitude,
      json_build_object(
        'type', 'Feature',
        'geometry', json_build_object(
          'type', 'Point',
          'coordinates', json_build_array(
            longitude,latitude
          )
        ),
        'properties', json_build_object(
          'id', id,
          'bssid', bssid,
          'ssid', ssid
        )
      )::text AS geojson, ssid
  FROM wifi_observations
  WHERE id = (
    SELECT wo.id
    FROM wifi_observations AS wo
    WHERE wo.bssid = wifi_observations.bssid
    ORDER BY
      (SELECT EXTRACT(YEAR FROM wo.observed_at)) DESC,
      (SELECT EXTRACT(WEEK FROM wo.observed_at)) DESC,
      COALESCE((raw_info->>'signal_level'),'100')::int
       ↪ ASC NULLS LAST
      LIMIT 1
```

Widok ten jest odświeżany wyzwalaczem przy każdej operacji modyfikującej

tabelę wifi_observations.

```
create or replace function refresh_wifi_positions()
returns trigger language plpgsql
as $$
begin
    refresh materialized view wifi_positions;
    return null;
end $$;
create trigger refresh_wifi_positions_trigger
after insert or update or delete or truncate
on wifi_observations for each statement
execute procedure refresh_wifi_positions();
```

Drugim elementem algorytmu jest obliczenie geograficznego środka ciężkości — punktu środkowego — dla znalezionych przez urządzenie lokalizacji sieci bezprzewodowych. Wykorzystałem do tego bibliotekę Geocoder.

```
positions = WifiPosition.where(bssid:
 ↪  params[:wifiAccessPoints].map do |ap|
  WifiObservation.standardize_bssid(ap[:macAddress])
end)

Geocoder::Calculations.geographic_center(
  positions.map(&:coordinates))
```

Warto jednak zajrzeć do implementacji metody geographic_center, ponieważ znajdziemy tam algorytm, który przelicza współrzędne geograficzne na trójwymiarowy układ kartezjański, po czym wylicza średnią wartość wyznaczonych punktów, a na koniec przelicza z powrotem na współrzędne geograficzne.

```
##
# Compute the geographic center (aka geographic
 ↪  midpoint, center of
# gravity) for an array of geocoded objects and/or
 ↪  [lat,lon] arrays
```

```
# (can be mixed). Any objects missing coordinates are
 ↪ ignored. Follows
# the procedure documented at
 ↪ http://www.geomidpoint.com/calculation.html.
#
def geographic_center(points)

  # convert objects to [lat,lon] arrays and convert
   ↪ degrees to radians
  coords = points.map{ |p|
   ↪ to_radians(extract_coordinates(p)) }

  # convert to Cartesian coordinates
  x = []; y = []; z = []
  coords.each do |p|
    x << Math.cos(p[0]) * Math.cos(p[1])
    y << Math.cos(p[0]) * Math.sin(p[1])
    z << Math.sin(p[0])
  end

  # compute average coordinate values
  xa, ya, za = [x,y,z].map do |c|
    c.inject(0){ |tot,i| tot += i } / c.size.to_f
  end

  # convert back to latitude/longitude
  lon = Math.atan2(ya, xa)
  hyp = Math.sqrt(xa**2 + ya**2)
  lat = Math.atan2(za, hyp)

  # return answer in degrees
  to_degrees [lat, lon]
end
```

2.2.4 Specyfikacja do Geolokalizacji użytkownika

Interfejs do wykonania geolokalizacji przez aplikację internetową utworzyłem na podstawie specyfikacji produktu Google Maps.[3] Oczywiście ta specyfikacja opisuje o wiele bardziej obszerne parametry do geolokalizacji. W ramach tej pracy

zaimplementowałem parametry określające sieci bezprzewodowe: BSSID oraz siłę sygnału odebraną przez urządzenie.

2.2.5 Widoki

Wszystkie widoki aplikacji internetowej wykorzystują Boostrap oraz jQuery. Wykorzystanie tych bibliotek pozwala na szybkie tworzenie aplikacji, których użyteczność i estetyka są na zadowalającym poziome. Dodatkowo style Boostrapa są przystosowane do prezentacji widoków aplikacji w różnych rozdzielczościach ekranów. Nawet dla przeglądarek internetowych na smartfonach strona prezentuje się poprawnie i schludnie.

2.3 Aplikacja mobilna

Implementację aplikacji mobilnej rozpocząłem od kreatora, w którym wybrałem opcję pojedynczego Activity. Na początku zaimplementowałem odpowiednie algorytmy skanujące sieci bezprzewodowe w zasięgu. W kolejnym kroku dodałem obsługę urządzenia GPS. Następnie stworzyłem tabelę w bazie danych aplikacji mobilnej z odpowiednią strukturą do przechowywania zebranych obserwacji sieci bezprzewodowych. Na koniec zaimplementowałem prezentację geolokalizacji uzyskanej z aplikacji internetowe oraz możliwość eksportu zebranych obserwacji lokalizacji sieci bezprzewodowych.

2.3.1 Uprawnienia na Android

Począwszy od Androida 6.0 (poziom API 23), użytkownicy przyznają uprawnienia do aplikacji podczas uruchamiania aplikacji, a nie podczas instalowania aplikacji. To podejście usprawnia proces instalacji, ponieważ użytkownik nie musi przyznawać uprawnień podczas instalowania lub aktualizacji. Daje to również użytkownikowi większą kontrolę nad funkcjonalnością aplikacji; Na przykład użytkownik może się zdecydować, czy udostępnić kamerę dostęp do aparatu, ale nie do lokalizacji urządzenia. Dodatkowo użytkownik może cofnąć uprawnienia w dowolnym momencie, przechodząc do ekranu *Ustawienia aplikacji*.[1]

Nowy system uprawnień wymaga dodatkowej uwagi twórców aplikacji mobilnych. Zanim

2.3.2 Skanowanie sieci bezprzewodowych

Klasycznym przykładem tego, jak Android został zaprojektowany, jest własnie skanowanie sieci bezprzewodowych. Aby uzyskać obiekty klasy ScanResults. Po pierwsze, musimy uzyskać instancję programu WifiManager. Następnie dzie-

dzicząc po klasie BroadcastReceiver zaimplementować metodę, która dostanie powiadomienie o zakończeniu przetwarzania. Taki asynchroniczny odbiornik — WifiScanReceiver — rejestrujemy z parametrem SCAN_RESULTS_AVAILABLE_ACTION i wreszcie rozpocząć skanowanie przez metodę startScan() na instancji WifiManager.

```
wifi_manager = (WifiManager)
 ↪ getApplicationContext().getSystemService(Context.WIFI_SERVICE);
wifi_scan_reciever = new WifiScanReceiver();
registerReceiver(wifi_scan_reciever, new
 ↪ IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
wifi_manager.startScan();
```

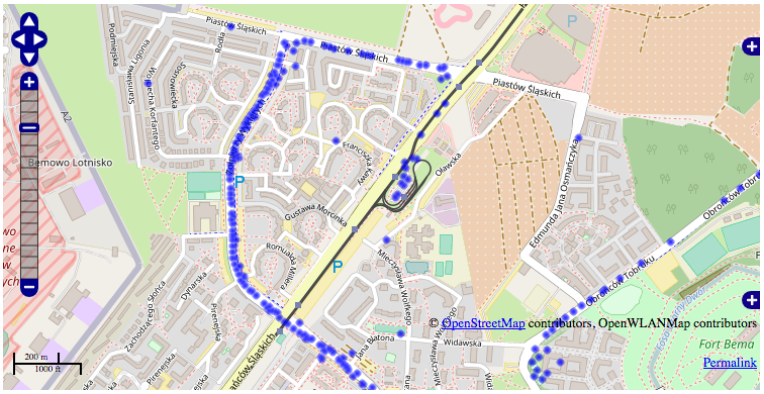
W Androidzie nie ma możliwości zparametryzowania, że chcemy uzyskiwać wyniki o zeskanowanych sieciach w sposób ciągły. Musimy więc w naszym WifiScanReceiver zlecić ponowne skanowanie zaraz po otrzymaniu wyniku ostatniego skanowania. Warto w tym miejscu dopisać sprawdzenie, czy współrzędne geograficzne ostatnio zapisane z urządzenia GPS są aktualne. W przeciwnym przypadku nie zapisywać informacji o lokalizacji ze względu na nieaktualność posiadanych współrzędnych.

```
private class WifiScanReceiver extends
 ↪ BroadcastReceiver {
    public void onReceive(Context context, Intent
     ↪ intent) {
        wifi_manager.startScan();
        if( last_location != null && ( (1000*10) >
         ↪ Calendar.getInstance().getTime().getTime()
         ↪ - last_location.getTime())){
            List<ScanResult> wifiScanList =
             ↪ wifi_manager.getScanResults();
            for (ScanResult wifi : wifiScanList) {
                if( wifi.SSID.contains("_nomap") ||
                 ↪ wifi.SSID.contains("_optout") ){
                    wifiScanList.remove(wifi);
```

W powyższym kodzie źródłowym widzimy usuwanie wyników zeskanowanych sieci z określonymi frazami w nazwie sieci. Wiąże się to z faktem, że firma Google zaproponowała dopisywanie do końca nazwy sieci frazę _nomap (z ang. nie mapuj), aby ich produkty i rozwiązania nie zbierały informacji o lokalizacji danych sieci bezprzewodowych.[5] Dodatkowego zamieszania dorzuciła firma Microsoft, gdzie dla swoich rozwiązań zaproponowali frazę _optout (z ang. wypisz mnie).[9]

Problem z określeniem kierunku sygnału

Nie jest możliwym stwierdzenie, z jakiego kierunku przyszedł sygnał sieci bezprzewodowej w przypadku, gdy urządzenie posiada jedną kartę sieciową i posiadamy tylko jedną informację z lokalizacją i sygnałem sieci. Powoduje to, że zbieranie informacji o lokalizacjach sieci bezprzewodowych jest skazane na dużą niedokładność. Dla algorytmów bardzo prymitywnych spowoduje to umiejscowienie wszystkich sieci bezprzewodowych na pozycji telefonu np. na środku ulicy lub na chodniku. Bardzo dobrze to widać na Rysunku 2.1



Rysunek 2.1: Prezentacja lokalizacji sieci bezprzewodowych na mapie - algorytm primitywny

Rozwiązanie lepsze, ale droższe, to użycie kilku urządzeń, najlepiej z antenami kierunkowymi, dzięki czemu moglibyśmy stwierdzić czy stacja bazowa

znajduje się po lewej stronie ulicy, czy po prawej na podstawie różnicy w sile sygnału. Dzięki technologii MIMO można określić, z którego kierunku przyszedł sygnał wykorzystując jedno urządzenie. Ze względu na denormalizację obserwacji sieci bezprzewodowych jesteśmy w stanie estymować kierunek, z którego przyszedł sygnał o ile mamy kilka zarejestrowanych skanów z różnymi siłami i lokalizacjami.

2.3.3 Określanie lokalizacji urządzenia

Do obsługi lokalizacji urządzenia poprzez GPS wykorzystałem bibliotekę — SmartLocation. Użycie prostego interfejsu biblioteki zaoszczędziło dużo czasu, który musiałbym poświęcić na zapoznawanie się z dokumentacją klas i interfejsów dotyczących nawigacji w Androidzie.

```
public void startGpsListener(){
    SmartLocation.with(this).location()
      .config(LocationParams.NAVIGATION).continuous()
      .start(gps_location_listener);
}
private class GpsLocationListener implements
 ↪ OnLocationUpdatedListener {
    @Override
    public void onLocationUpdated(Location location) {
        last_location = location;
    }
}
```

Sam proces uzyskania informacji o lokalizacji urządzenia jest asynchroniczny. Aby zniwelować efekt asynchroniczności, utworzyłem lokalną zmienną w ramach widoku, do której zapisuję ostatnią otrzymaną pozycję z SmartLocation. Do uzyskania najdokładniejszej pozycji skorzystałem z funkcji trybu nawigacji (z ang. navigate) z parametrem pracy ciągłej (z ang. continious). Trzeba pamiętać, że

takie ustawienie jest niekorzystne dla czasu pracy na baterii, ale gwarantuje to najświeższą informację o współrzędnych urządzenia.

2.3.4 Baza danych

Teraz gdy już mamy wyniki skanowania sieci bezprzewodowych i lokalizację urządzenia, to jesteśmy w stanie zapisać wyniki do bazy danych aplikacji mobilnej — aby potem przesłać do aplikacji internetowej — dla lepszej precyzji określania lokalizacji sieci bezprzewodowych, a tym samym użytkownika.

Wybrałem bibliotekę ActiveAndroid jako ORM ze względu na swoje podobieństwo do ActiveRecord ze środowiska języka programowania Ruby oraz ze względu na użycie bazy SQLite, która jest popularnym rozwiązaniem do przechowywania danych na Androidzie. Stosując się do dobrych praktych twórców tej biblioteki, umieściłem tworzenie obserwacji w transakcji:

```
ActiveAndroid.beginTransaction();
try {
    for (ScanResult wifi : wifiScanList) {
        WifiObservation wifiObservation = new
         ↪ WifiObservation(wifi, last_location);
        wifiObservation.save();
    }
    ActiveAndroid.setTransactionSuccessful();
} finally {
    ActiveAndroid.endTransaction();
}
```

Po raz kolejny podjąłem decyzję o denormalizacji danych. Dane takie będą zajmować więcej miejsca, ale umożliwią na dokładniejsze lokalizowanie miejsca, w którym jest umiejscowiony punkt dostępu. Wszystkie dane będą zapisane w pojedynczej tabeli.

Konstruktor klasy WifiObservation przyjmujący zeskanowaną sieć oraz ostatnią lokalizację zaimplementowałem następująco:

Tablica 2.4: Kolumny i typy przechowywania danych wraz z ich znaczeniem w aplikacji mobilnej

Typ	Nazwa	Znaczenie (opcjonalne)
String	ssid	
String	bssid	
int	signal_level	Wykryty poziom sygnału w dBm, znany również jako RSSI.[2]
String	capabilities	Opisuje schematy uwierzytelniania, zarządzania kluczami i szyfrowania obsługiwane przez punkt dostępu.[2]
Date	observed_at	Data i czas o sieć została zeskanowana
int	channel_frequency	Częstotliwość zeskanowanej sieci
double	latitude	Szerokość geograficzna pozycji, w której sieć zeskanowano
double	longitude	Wysokość geograficzna pozycji, w której sieć zeskanowano
Date	geolocated_at	Data i czas otrzymania informacji o współrzędnych geograficznych urządzenia
float	geolocation_accuracy	Dokładność współrzędnych geograficznych pozycji w któ®ej sieć zeskanowano
boolean	is_exported	Informacja o tym, czy sieć została wyeksportowana do aplikacji internetowej

```
public WifiObservation(ScanResult scanResult, Location
 ↪ location){
    super();
    this.ssid = scanResult.SSID;
    this.bssid = scanResult.BSSID;
    this.signal_level = scanResult.level;
    this.capabilities = scanResult.capabilities;
    this.channel_frequency = scanResult.frequency;

    this.observed_at = new Date();

    this.latitude = location.getLatitude();
    this.longitude = location.getLongitude();
    this.geolocated_at = new Date(location.getTime());
    this.geolocation_accuracy =
     ↪ location.getAccuracy();

    this.is_exported = false;
}
```

2.3.5 Określenie identyfikatora urządzenia

Przy zbieraniu informacji o lokalizacjach sieci bezprzewodowych chciałem, aby dane obserwacje były oznaczone, przez jakie urządzenie zostały zaimportowane. Stosując zalecane rozwiązanie, napisałem metodę, która przy pierwszym uruchomieniu generuje unikalny identyfikator i zapisuje go w przestrzeni ustawień aplikacji mobilnej. Każde następne uruchomienie, pobierze ten wcześniej wygenerowany identyfikator.[8]

```
public String getUniqueDeviceId(){
    SharedPreferences sharedPref =
     ↪ getPreferences(Context.MODE_PRIVATE);
    String device_id =
     ↪ sharedPref.getString(preferenceUniqueIdKey,
     ↪ null);
    if(device_id == null){
        SharedPreferences.Editor editor =
         ↪ sharedPref.edit();
```

```
            device_id = UUID.randomUUID().toString();
            editor.putString(preferenceUniqueIdKey,
             ↪ device_id);
            editor.apply();
        }
        return device_id;
    }
```

2.3.6 Prezentacja danych — mapa

Finalną częścią aplikacji mobilnej była prezentacja danych oraz zaimplementowanie algorytmów do zbadania tych danych. Wykorzystałem do tego problemu wcześniej wspomnianą bibliotekę leaflet. Stworzyłem Androidową kontrolkę WebView do której poprzez odpowiedni interfejs JavaScriptowy przekazałem aktualną pozycję z urządzenia GPS oraz sieci zeskanowane w zasięgu smartfonu. Ze względu na różnice w interfejsie klasy WebView, na przestrzeni lat zmienił się sposób uruchamiana JavaScriptu. Zaimplementowałem odpowiednie sprawdzenie, aby uzyskać zgodność ze starym oraz nowym interfejsem klasy WebView.

```
public void runJavascript(String code){
    if (android.os.Build.VERSION.SDK_INT >=
     ↪ android.os.Build.VERSION_CODES.KITKAT) {
        webView.evaluateJavascript(code, null);
    } else {
        webView.loadUrl("javascript:("+code+");");
    }
}
```

Ze względu na fakt, że wszystkie operacje są wykonywane w logice widoku, każde uśpienie aplikacji wstrzymywało pomiary i urządzenie GPS. Dodałem odpowiednią flagę do okna aplikacji, która poinformowała system o tym, żeby nie usypiać urządzenia w trakcie działania tej aplikacji.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    webView = new WebView(this);
    setContentView(webView);
    getWindow().addFlags(
      WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    // ...
}
```

Przy tworzeniu mapy, okazało się, że aplikacja internetowa nie posiadała odpowiednich nagłówków pozwalających na uruchamianie zapytań z innych domen. Rozwiązaniem tego problemu było dodanie odpowiednich nagłówków standardu Cross-Origin Resource Sharing. W przypadku aplikacji Ruby On Rails globalnie ten problem rozwiązałem przez import biblioteki rack-cors i konfigurację serwera aplikacji przy uruchamianiu:

```
module RailsAppStartingTemplate
  class Application < Rails::Application
    # ...
    config.middleware.insert_before 0, Rack::Cors do
      allow do
        origins '*'
        resource '*', :headers => :any, :methods =>
          ↪ [:get, :post, :options]
      end
    end
    # ...
  end
end
```

Samą prezentację oraz monitorowanie parametrów geolokacji rozwiązałem poprzez zapisanie ilości wykonanych geolokalizacji, zsumowanie wszystkich deklarowanych precyzji geolokalizacji (aby otrzymać średnią arytmetyczną) i zmierzenie otrzymanej lokalizacji w porównaniu do obecnej lokalizacji podanej przez GPS i zsumowanie jej do poprzednich wartości (dla średniej arytmetycznej).

```
geowifiTimes += 1;
geowifiAccuracy += parseFloat(data.accuracy || 0)
```

```
geowifiMeasuredAccuracy +=
 ↪  distance(gpsMarker.getLatLng().lat,
 ↪  gpsMarker.getLatLng().lng, data.location.lat,
 ↪  data.location.lng)
wifiMarker.setLatLng([data.location.lat,
 ↪  data.location.lng]);
fitMarkersIntoScreen();
```

Dzięki temu, że interfejs geolokalizacji mojej aplikacji internetowej, działa zgodnie ze specyfikacją interfejsu produkty Mapy firmy Google w części geolokalizacji na podstawie sieci bezprzewodowych, to mogłem dodać bardzo niskim kosztem ich serwer geolokacji i zmierzyć jego parametry. Dla rozróżnienia, nadałem mojej aplikacji nazwę geowifi. Obliczenie i prezentacja danych została zaimplementowana następująco:

```
this._div.innerHTML = '<strong>Średni czas:</strong>';
this._div.innerHTML += '<br>GeoWifi: ' + (  (((new
 ↪  Date()) - timeStart)/1000) / geowifiTimes
 ↪  ).toFixed(3) + ' s';
this._div.innerHTML += '<br>Google: ' + (  (((new
 ↪  Date()) - timeStart)/1000) / googleTimes
 ↪  ).toFixed(3) + ' s';

this._div.innerHTML += '<br><strong>Średnia
 ↪  deklarowana precyzja:</strong>';
this._div.innerHTML += '<br>GeoWifi: ' + parseInt(
 ↪  geowifiAccuracy / geowifiTimes ) + ' m';
this._div.innerHTML += '<br>Google: ' + parseInt(
 ↪  googleAccuracy / googleTimes ) + ' m';

this._div.innerHTML += '<br><strong>Średnia zmierzona
 ↪  precyzja:</strong>';
this._div.innerHTML += '<br>GeoWifi: ' + parseInt(
 ↪  geowifiMeasuredAccuracy / geowifiTimes ) + ' m';
this._div.innerHTML += '<br>Google: ' + parseInt(
 ↪  googleMeasuredAccuracy / googleTimes ) + ' m';
```

```
this._div.innerHTML += '<br><strong>Liczba
 ↪ geolokacji:</strong>';
this._div.innerHTML += '<br>GeoWifi: ' + parseInt(
 ↪ geowifiTimes );
this._div.innerHTML += '<br>Google: ' + parseInt(
 ↪ googleTimes );
```

2.4 Badanie i wnioski

Na tym etapie pracy zauważyłem już, że zaimportowane dane z publicznych baz danych były bardzo ograniczone. Dla obszaru Warszawy zaimportowanych zostało mniej niż **200 tysięcy unikalnych** pozycji sieci bezprzewodowych, co w porównaniu z szacowanymi **944 012** z części teoretycznej, zapowiadało porażkę. Dlatego dla obszarów mojego zamieszkania i miejsca pracy wykonałem kilka przejść z aplikacją i wyeksportowałem informacje o lokalizacjach sieci bezprzewodowych. Szacuję , że zagęszczenie w tych lokalizacjach zwiększyło się przynajmniej dwukrotnie.

2.4.1 Pomiar zagęszczenia sieci bezprzewodowych na terenie Warszawy

Pomiar zagęszczenia w Warszawie oparłem na 3 próbkach. Dwie wybrane ręcznie ze względu na ograniczony zestaw informacji o lokalizacjach sieci bezprzewodowych oraz jeden wybrany losowo. Kod źródłowy obliczający gęstość sieci na km^2 wyglądał następująco:

```
distance = 0.5
warsaw_spire_plac_europejski_1 = [52.232175,
 ↪ 20.984187]
losowy = [52.273075, 20.974593]
zeromskiego_1 = [ 52.275371, 20.960779]

[
  WifiPosition.within_bounding_box(
```

```
    Geocoder::Calculations.bounding_box(losowy,
     ↪ distance)).length,
  WifiPosition.within_bounding_box(
    Geocoder::Calculations.bounding_box(
    warsaw_spire_plac_europejski_1, distance)).length,
  WifiPosition.within_bounding_box(
    Geocoder::Calculations.bounding_box(zeromskiego_1,
     ↪ distance)).length
]
```

Tablica 2.5: Wyniki pomiaru gęstości sieci bezprzewodowych

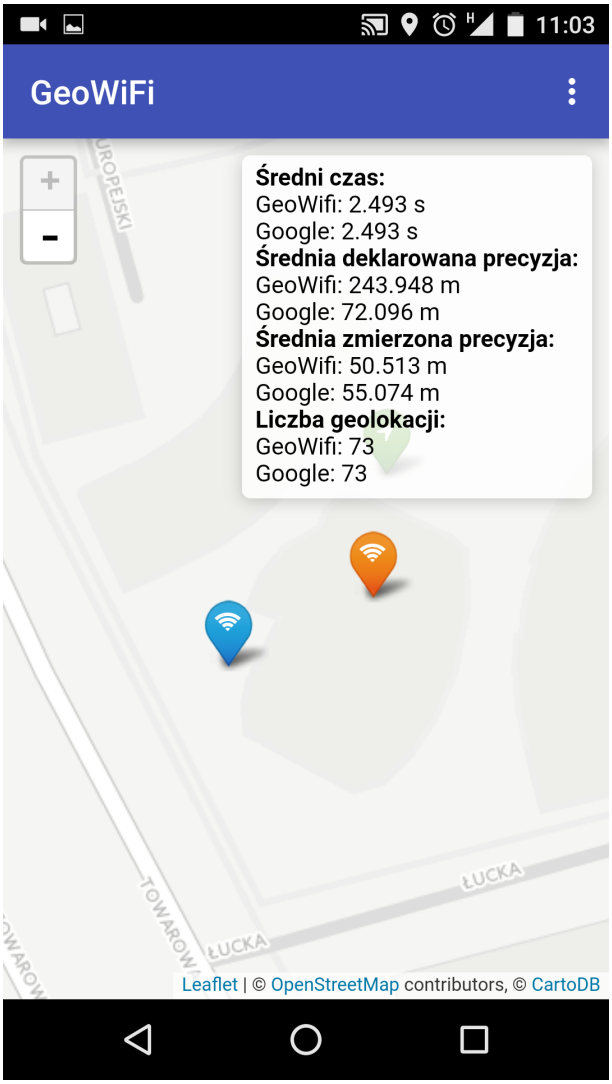
Lokalizacja	Ilość bezprzewodowych sieci na km^2
Losowy punkt	536
Plac Europejski 1	993
Żeromskiego 1	1129

Wartości z Tablicy 2.5 potwierdzają, że Warszawa posiada odpowiednią infrastrukturę sieci bezprzewodowych, aby nawigować po niej przy użyciu tylko lokalizacji sieci bezprzewodowych. Szacuję że dla rejonów centrum Warszawy powinniśmy zobaczyć wartości powyżej 3 lub 4 tysięcy. Jednak zeskanowanie lokalizacji sieci bezprzewodowych na obszarze kilometra kwadratowego to kosztowna operacja i podejrzewam to za główny powód niedokładności publicznych zbiorów danych.

2.4.2 Pomiar precyzji geolokalizacji

W kwestii precyzji wyniki nie były bardzo satysfakcjonujące, co widać na Rysunku 2.2 Zmierzona precyzja geolokalizacji nie osiągnęła minimalnej wymaganej określonej w tezie. Nie oznacza to, że nie można nawigować przy użyciu lokalizacji sieci bezprzewodowych. Oznacza to jedynie, że nie jest ona odpowiednią konkurencją dla urządzeń GPS. Osiągnięte wyniki jednak nie są bardzo złe. W przy średniej zmierzonej precyzji osiągnięto 50,513 metra dla sposobu przedstawionego w tej pracy. Oraz 55,074 dla platformy Google Maps. Zauważalna

jest duża różnica w stosunku do deklarowanej precyzji przez obydwie platformy. Odpowiednio 243,948 metra dla aplikacji internetowej (geowifi) oraz 72,096 dla platformy Google. W pomiarze wystąpiły pełne 73 cykle geolokalizacji.



Rysunek 2.2: Zrzut ekranu z aplikacji mobilnej na koniec pomiaru precyzji i szybkości geolokalizacji

2.4.3 Pomiar szybkości geolokalizacji

W każdym cyklu tym najwięcej czasu zajmowało zeskanowanie sieci bezprzewodowych przez urządzenie. Na badanym urządzeniu udało się osiągnąć wynik 2.493 sekund na pełen cykl geolokalizacji.(Rysunek 2.2) Jest to zadowalający wynik, jednak w aplikacjach nawigacyjnych podejrzewam, że trzeba byłoby kompensować tak niską częstotliwość odświeżania pozycji, szacunkami na podstawie kierunku i prędkości urządzenia.

2.4.4 Wnioski

Praca ta dowodzi, że Warszawa jest odpowiednio rozwiniętym i nowoczesnym miastem. Urządzenia są w stanie określać geolokalizację w odpowiedniej częstotliwości. Niestety jednak precyzja pozostawia wiele do życzenia. Możliwe jest stosowanie takiej geolokalizacji do urządzeń, w których niepotrzebna jest precyzyjna lokalizacja, a zmniejszone koszty zużycia baterii dla geolokalizowania są pożądane. Takie urządzenia i ich zastosowania to mogą być krokomierze, systemy lokalizowania skradzionych przedmiotów — w tych przypadkach, geolokalizacja na podstawie sieci bezprzewodowych może mieć nawet lepsze efekty niż urządzenia GPS ze względu na fakt, że technologia GPS wymaga dostępu do tzw. otwartego nieba.

Podsumowując: aby wykorzystać rozwiązanie geolokalizowania na podstawie sieci bezprzewodowych, trzeba opracować algorytmy, o lepszej precyzji i utworzyć dokładniejsze bazy danych o lokalizacjach. Ewentualnie można wykorzystać to rozwiązanie do określania lokalizacji z precyzją rzędu ponad 50 metrów.

Słownik pojęć

ORM • skrótowe oznaczenie dla "mapowanie obiektowo-relacyjne"(od angielskiego Object-Relational Mapping).

Punkt dostępu (od ang. Accesspoint) • Stacja bazowa sieci bezprzewodowej

Aplikacja internetowa • Aplikacja WWW/HTTP (webowa)

SSID • Nazwa punktu dostępu

BSSID • adres MAC stacji bazowej sieci bezprzewodowej

Klastrowanie • Zadaniem klastrowania jest pogrupowanie zbioru obserwacji w klastry tak, aby w ramach klastra obserwacje były do siebie, jak najbardziej podobne, a jednocześnie jak najbardziej różne od obserwacji w innych klastrach.

Kanał (w kontekście sieci bezprzewodowych i punktów dostępów) • Sieci bezprzewodowe operują na określonych częstotliwościach. W przypadku specyfikacji IEEE 802.11g jest ich 14. Te poszczególne częstotliwości nazywamy kanałami.

wardriving • Jest to czynność, która polega na zbieraniu jak największej ilości sieci bezprzewodowych. Wielokrotnie opisywana krytycznie ze względu na pomyłkę w intencji osoby wykonującej zbieranie informacji. Często mylona z uzyskiwaniem dostępu do sieci bezprzewodowych. [20]

MIMO (od ang. Multiple-Input-Multiple-Output) • rozwiązanie, w którym karta sieciowa posiada wiele wyjść i wiele wejść, przez co można nadawać i odbierać na specyficznym Wejściu/Wyjściu, które posiada lepsze parametry komu-

nikacji

denormalizacja • jest to wprowadzenie kontrolowanej nadmierności do bazy danych w celu przyśpieszenia wykonywania na niej operacji (np. obsługiwania zapytań); dzięki denormalizacji bazy unika się kosztownych operacji połączeń tabel[13]

Spis tablic

Bibliografia

[1] developer.android.com. Requesting permissions at run time. https://developer.android.com/training/permissions/requesting.html.

[2] developer.android.com. Scanresult. https://developer.android.com/reference /android/net/wifi/ScanResult.html.

[3] developers.google.com. The google maps geolocation api. https://developers.google.com/maps/documentation/geolocation/intro.

[4] en.wikipedia.org. List of wlan channels. https://en.wikipedia.org/wiki/List_of_WLAN_channels.

[5] P. Fleischer. Greater choice for wireless access point owners. https://googleblog.blogspot.com/2011/11/greater-choice-for-wireless-access.html, 2011.

[6] githut.info. Githut - programming languages and github. http://githut.info/.

[7] A. La. Language trends on github. https://github.com/blog/2047-language-trends-on-github.

[8] mbwasi stackoverflow.com. java - saving user id in shared preferences android - stack overflow. https://en.wikipedia.org/wiki/XML.

[9] P. Morris. Disable wi-fi sense password sharing in windows 10, here's how. http://www.redmondpie.com/disable-wi-fi-sense-password-sharing-in-windows-10-heres-how/, 2015.

[10] pl.wikipedia.org. Android runtime. https://pl.wikipedia.org/wiki/Android_Runtime.

[11] pl.wikipedia.org. Android (system operacyjny). https://pl.wikipedia.org/wiki/Android_$system_operacyjny$.

[12] pl.wikipedia.org. Dalvik (maszyna wirtualna). https://pl.wikipedia.org/wiki/Dalvik_(maszyna_wirtualna).

[13] pl.wikipedia.org. Denormalizacja bazy danych. https://pl.wikipedia.org/wiki/Denormalizacja_bazy_danych.

[14] pl.wikipedia.org. Fale radiowe. https://pl.wikipedia.org/wiki/Fale_radiowe.

[15] pl.wikipedia.org. Geolokalizacja. https://pl.wikipedia.org/wiki/Geolokalizacja.

[16] pl.wikipedia.org. Java. https://pl.wikipedia.org/wiki/Java.

[17] pl.wikipedia.org. Json. https://pl.wikipedia.org/wiki/JSON.

[18] pl.wikipedia.org. Postgresql. https://pl.wikipedia.org/wiki/PostgreSQL.

[19] pl.wikipedia.org. Ruby (język programowania). https://pl.wikipedia.org/wiki/Ruby_(j%C4%99zyk_programowania).

[20] pl.wikipedia.org. Wardriving. https://pl.wikipedia.org/wiki/Wardriving.

[21] pl.wikipedia.org. Warszawa. https://pl.wikipedia.org/wiki/Warszawa.

[22] pl.wikipedia.org. Wirtualna maszyna javy. https://pl.wikipedia.org/wiki/Wirtualna_maszyna_Javy.

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

[24] G. U. S. D. B. D. I. R. PRACY. Prognoza gospodarstw domowych na lata 2016 – 2050. http://stat.gov.pl/download/gfx/portalinformacyjny/pl/defaultaktualnosci/5469/9/4/1/prognoza_gospodarstw_domowych_na_lata_2016-2050_002.pdf.

[25] G. U. S. D. B. D. I. R. PRACY. Społeczeństwo informacyjne w polsce w 2016 r. http://stat.gov.pl/download/gfx/portalinformacyjny/pl/defaultaktualnosci/5497/2/6/1/si__sygnalna_2016.pdf, 2016.

[26] sjp.pl. geolokalizacja. http://sjp.pl/geolokalizacja.

[27] trends.builtwith.com. Ruby on rails usage statistics. https://trends.builtwith.com/framework/Ruby-on-Rails.

[28] S. Zandbergen, P. & Barbeau. Positional accuracy of assisted gps data from high-sensitivity gps-enabled mobile phones. *Journal of Navigation, 64(3), 381-399. doi:10.1017/S0373463311000051*, 2011.