Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

(повна назва інституту/факультету)

Кафедра телекомунікацій

(повна назва кафедри)

«На правах рукопису»

УДК _____

До захисту допущено
В.о. завідувача кафедри
_____ Явіся В.С.___
(підпис) (ініціали, прізвище)
“___”_____2018_р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 172 Телекомунікації та радіотехніка,

спеціалізація Апаратно-програмні засоби електронних комунікацій

на тему: «Розробка покращеного алгоритму оцінки якості стільникового покриття»_____

Виконав студент_2_ курсу, групи ТЗ-71мп
Михайлов Сергій Олександрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Міночкін Дмитро Анатолійович_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 рік

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем
(повна назва)

Кафедра телекомунікацій
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність 172 Телекомунікації та радіотехніка
(код і назва)

Спеціалізація Апаратно-програмні засоби електронних комунікацій

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____Явіся В.С.
(підпис) (ініціали, прізвище)

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Михайлову Сергію Олександровичу

1. Тема дисертації: «Розробка покращеного алгоритму оцінки якості стільникового покриття»
науковий керівник дисертації: д. т. н. Міночкін Дмитро Анатолійович
затверджені наказом по університету від «_06_» «_11_» 2018р. № _4095-с_
2. Строк подання студентом дисертації _____
3. Об’єкт дослідження: безпроводові сенсорні мережі.
4. Предмет дослідження: розвиток методів синхронізації в безпроводових сенсорних мережах.
5. Перелік завдань, які потрібно розробити
1) Аналіз існуючих методів збору даних для оцінки якості стільникового покриття.
2) Знаходження шляхів підвищення точності даних про якість стільникового покриття.
3) ___ Проектування мобільного додатку на базі Xamarin для збору даних про якість стільникового покриття.
4) ___ Проектування веб застосунку на базі .Net Core Framework для збереження даних про якість стільникового покриття та підвищення їх точності. 6. Орієнтований перелік ілюстративного матеріалу
6. Орієнтований перелік ілюстративного матеріалу _____
_____
_____
7. Орієнтований перелік публікацій _____
_____
_____
8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи)	Строк виконання етапів роботи	Примітка
1	Пошук та опрацювання літератури з теми наукового дослідження.	Вересень-жовтень 2017	
2	Огляд сучасних матеріалів та публікацій по результатам досліджень в сфері оцінки якості стільникового звязку.	Листопад-грудень 2018	
3	Знаходження шляхів підвищення точності даних про якість стільникового покриття.	Січень 2018	
4	Проектування мобільного додатку на базі Xamarin для збору даних про якість стільникового покриття.	Лютий- червень 2018	
5	Проектування веб застосунку на базі .Net Core Framework для збереження даних про якість стільникового покриття та підвищення їх точності.	Липень - жовтень 2018	
6	Підбиття підсумків, оформлення та редагування розділів роботи.	Листопад-грудень 2018	

Студент _____ Михайлов С. О.
(підпис)

Керівник проекту (роботи) _____ Міночкін Д. А.
(підпис)

РЕФЕРАТ

Робота містить 100 сторінок, 21 рисунок, 4 таблиці. Було використано 29 джерел.

Мета дослідження: розробка мобільного додатку для збору даних про якість стільникового покриття.

Об’єкт дослідження: фреймворк для кросплатферменої мобільної розробки Xamarin.

Предмет дослідження: використання фреймворку для кросплатферменої розробки Xamarin для збору даних про якість стільникового покриття.

В роботі розглянуто особливості розробки мобільного додатку на базі фреймворку Xamarin розробки веб застосунку на базі .Net Core Framework. Зазначений додаток дозволяє у фоновому режимі збирати дані про якість стільникового зв’язку мобільного апарату користувача та передавати ці дані на сервер, де точність даних підвищується завдяки використанню сторонніх мікросервісів та агрегуються разом з даними від інших користувачів.

Ключові слова: Xamarin, .Net Framework, .Net Core, Rest, Http, QoS, Android.

ABSTRACT

The work contains 100 pages, 21 figures, 4 tables, 29 sources have been used.

Goal: is the development of a mobile application for the collection of data on the quality of cellular coverage**.**

Object of research: framework for cross platform mobile development Xamarin.

Subject of the study: use of the framework for cross-fertilization Xamarin to collect data on the quality of cellular coverage**.**

The work considers the development of a mobile application based on the Xamarin framework for the development of a Web application based on .Net Core Framework. This application allows in the background to collect data on the quality of the cellular communication of a mobile device user and transfer this data to the server, where the accuracy of the data increases with the use of the side of the micro services and aggregated together with data from other users.

Key words: Xamarin, .Net Framework, .Net Core, Rest, Http, QoS, Android.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

QoS – (Quality of Service) – Якість обслуговування

VoIP – (Voice over IP) – Голос через IP

FTP – (File Transfer Protocol) – Протокол передачі файлів

SMTP – (Simple Mail Transfer Protocol) – Простий Протокол Пересилання Пошти

IPTV – (Internet Protocol Television) – Цифрове інтерактивне телебачення в мережах передачі даних за протоколом IP

GPS – (Global Positioning System) – Система глобального позиціонування

РЧЦ – Радіочастотний центр

GPRS – (General Packet Radio Service) – Загальний сервіс пакетної радіопередачі

EDGE – (Enhanced Data Rates for GSM Evolution) – Технологія передачі даних, що забезпечує передачу великих обсягів інформації в мережі мобільного зв'язку

UMTS – (Universal Mobile Telecommunications System) – Універсальна Мобільна Телекомунікаційна Система

ISDN – (Integrated Services Digital Network) – Цифрова мережа з інтегрованими службами (послугами)

MOS – (mobile operating system) – Мобільна операційна система

PESQ – (Perceptual Evaluation of Speech Quality) – Перцептуальна оцінка якості мовлення

POLQA – (Perceptual Objective Listening Quality Analysis) – Аналіз аудиторської якості

UWP – (Universal Windows Platform) – Універсальна платформа Windows

OS – (Operating system) – Операційна система

SOAP – (Simple Object Access Protocol) – Протокол обміну структурованими повідомленнями

XML – (Extensible Markup Language) – Розширювана мова розмітки

RPS – (Remote Procedure Call) – Виклик віддалених процедур

DDoS – (Distributed Denial of Service) – Атака на відмову в обслуговуванні, розподілена атака на відмову в обслуговуванні

URL – (Uniform Resource Locator) – Уніфікований локатор ресурсів

СУБД – Система керування базами даних

ID – (Identity document) – Ідентифікатор

GUID – (Globally Unique Identifier) – Статистично унікальний 128-бітний ідентифікатор

JSON – (JavaScript Object Notation) – Запис об'єктів JavaScript

ВСТУП

Драйв-тест (далі ДТ) є одним із методів вимірювання якості послуг (QoS) в мережах мобільного зв'язку разом зі стаціонарними вимірами, а також збором статистики оператором. На відміну від інших методів, ДТ є виміром, що відбувається під час руху.

Мета драйв-тесту - якомога точніше відобразити умови, в яких клієнт користується мобільним зв'язком (переміщення з постійною і змінною швидкістю, мінливі радіо умови), а також поведінку самого клієнта (висота розташування антен вимірювальних комплексів, поєднання телефонних з'єднань і використання інтернет-послуг згідно сценарію, заснованому на спостереженнях за діями користувачів). Завдяки цьому вимірювання наближені до реального досвіду користувача мобільного мережі.

Одним з найбільш ефективних способів проведення ДТ є використання для збору даних мобільних апаратів звичайних користувачів послуг стільникового зв’язку.

Результати ДТ можуть бути використані як стільниковими операторами для підвищення якості надаваних послуг, так і НКРЗІ (Національна комісія, що здійснює державне регулювання у сфері зв’язку та інформатизації) при опрацюванні звітів операторів про якість телекомунікаційних послуг та їх відповідності встановленим рівням якості послуг мобільного зв’язку [1].

Отже, проведення ДТ є актуальним і важливим способом оцінки якості стільникового покриття.

1. АНАЛІЗ ЗАДАЧІ ОЦІНКИ ЯКОСТІ СТІЛЬНИКОВОГО ПОКРИТТЯ

1.1 Актуальність задачі оцінки якості надаваних послуг

QoS визначає гарантований рівень якості послуг, що надаються користувачеві стільникового зв'язку протягом усього періоду дії договору у даного мобільного оператора [2]. У сфері телекомунікаційних послуг QoS визначається трьома параметрами: доступом, передачею інформації і роз'єднанням, а також трьома критеріями: швидкістю, точністю і надійністю. Оператор зобов'язаний забезпечувати якісне надання послуг. Звичайні користувачі найчастіше стикаються з терміном QoS в домашніх маршрутизаторах з підтримкою QoS. Наприклад, досить логічно дати високий пріоритет пакетам VoIP і низький - пакетам FTP, SMTP і клієнта файлообмінної мережі.

Приклади технологій, що вимагають високий QoS [3]:

1. IPTV;
2. Потокові аудіо сервіси;
3. VoIP;
4. Відеоконференції;
5. Додатки, що забезпечують стабільну роботу системи або її відновлення;
6. Динамічні онлайн-ігри в реальному часі.

У цьому контексті технологія QoS покликана забезпечити при проходженні трафіку більш "високої" категорії задані значення параметрів незалежно від інтенсивності трафіку інших категорій. Ця технологія необхідна саме для захисту найбільш пріоритетного трафіку від різного роду "зазіхань" з боку менш пріоритетного трафіку, а не просто для "використання мультимедіа в мережі".

Забезпечення QoS дуже важливо для справедливого розподілу мережевих ресурсів і вироблення у користувачів звички очікувати від мережі саме тих значень параметрів, які вони просили. Потреба у використанні даної технології збільшується в міру того, як численні магістралі мереж з різними характеристиками

і системами пріоритетів об'єднуються в процесі створення єдиної мережі в масштабах одного або декількох підприємств.

1.2 Теорія оцінки якості стільникового покриття

Тести за якістю мобільного зв'язку фіксують найчастіше такі параметри:

1. Якість голосового зв'язку. Включає в себе частку неуспішних спроб встановлення з'єднання, частку обірваних викликів, розбірливість мови. Якість мови вимірюється за технологією POLQA (Perceptual Objective Listening Quality Assessment), затвердженої Міжнародним союзом електрозв'язку. Результатом обдзвону «великої четвірки» є сукупність log-файлів вимірювальних комплексів. Оцінка кожного голосового виклику здійснюється протягом 3 хвилин.

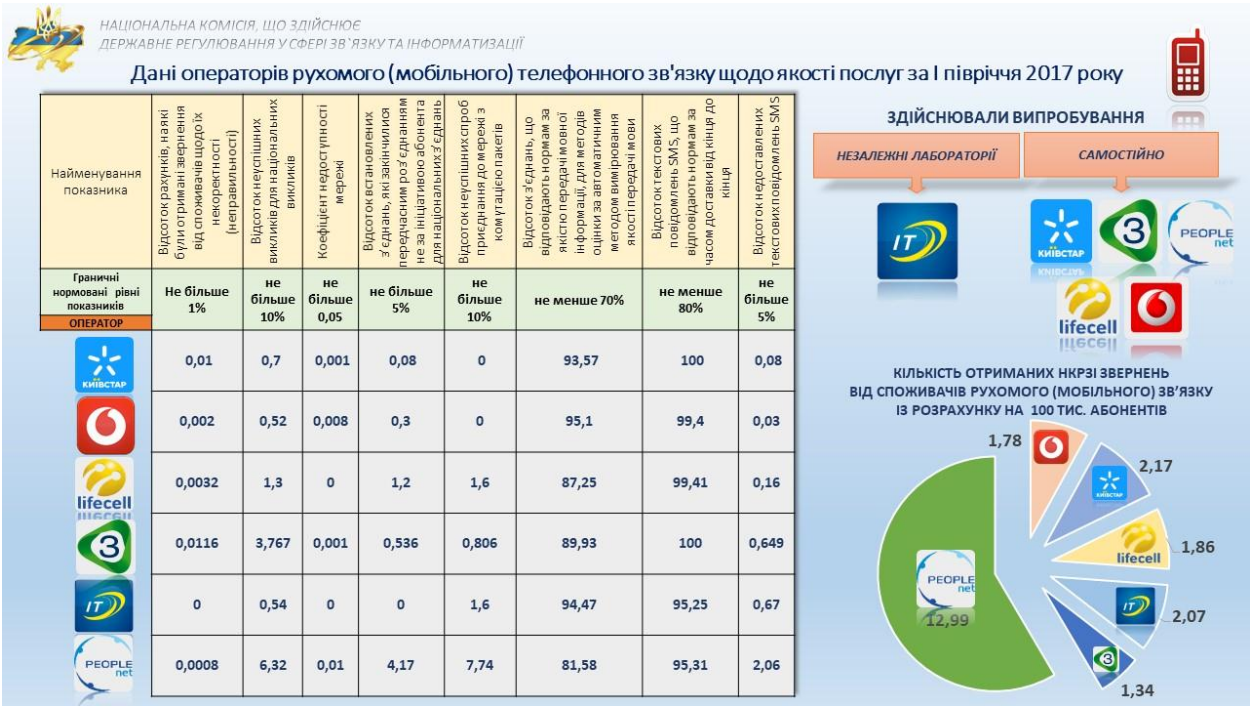
2. Доставку SMS (Short Message Service) -повідомлень (частка втрачених повідомлень адресату, а також середній час доставки).

3. Передачу даних: частка неуспішного TCP / IP (Transmission Control Protocol/Internet Protocol) з'єднання з сервером (HTTP IP-Service Access Failure Ratio); частка неуспішних сесій по протоколу HTTP (HTTP Session Failure Ratio); середнє значення швидкості передачі даних до абонента (HTTP DL Mean User Data Rate) в кбіт / с; тривалість успішної сесії (HTTP Session Time) в мс. Процедура виконується для трьох стандартів: GSM (2G), UMTS (3G) і LTE (4G). Розмір файлу для завантаження по протоколу HTTP для GSM (Groupe Spécial Mobile) і UMTS (Universal Mobile Telecommunications System) становить 3 МБ, а для LTE (Long-Term Evolution) 100 МБ.

4. Загальний скан покриття, наявність і рівень сигналу (вимірюваний в дБм).

Слід зазначити, що дані про якість мобільного зв’язку повинні збирати самі оператори зв’язку, потім отримані матеріали аналізуються, формується протокол, після чого вони відправляються на сайт НКРЗІ, що аналізує та публікує отримані дані. Зокрема ці показники відображені на звіті НКРЗІ за І півріччя 2017 року (рис 1.2.1) [4].

Подібні заміри якості надаваних послуг у тому чи іншому вигляді існують у більшості країн світу, і майже для всіх країн також зазначається, що заміри не можна проводити під час аномального навантаження на мережу: наприклад, в свята, під час масових заходів або великих подій. Також не можна проводити нічні виміри (https://habr.com/post/406595/).



Найменування показника	Відсоток рахунків, назні були отримані звернення від споживачів щодоїх некоректності (неправильності)	Відсоток неуспішних викликів для національних викликів	Коефіцієнт недоступності мережі	Відсоток встановлених з'єднань, які закінчилися передчасним роз'єднанням не за ініціативою абонента для національних з'єднань	Відсоток неуспішних спроб приєднання до мережі з комутацією пакетів	Відсоток з'єднань, що відповідають нормам за якістю передачі мовної інформації, для методів оцінки за автоматичним методом вимірювання якості передачі мови	Відсоток текстових повідомлень SMS, що відповідають нормам за часом доставки від кінця до кінця	Відсоток недоставлених текстових повідомлень SMS
Граничні нормовані рівні показників	Не більше 1%	не більше 10%	не більше 0,05	не більше 5%	не більше 10%	не менше 70%	не менше 80%	не більше 5%
ОПЕРАТОР								
Київстар	0,01	0,7	0,001	0,08	0	93,57	100	0,08
	0,002	0,52	0,008	0,3	0	95,1	99,4	0,03
lifecell	0,0032	1,3	0	1,2	1,6	87,25	99,41	0,16
3	0,0116	3,767	0,001	0,536	0,806	89,93	100	0,649
IT	0	0,54	0	0	1,6	94,47	95,25	0,67
PEOPLE net	0,0008	6,32	0,01	4,17	7,74	81,58	95,31	2,06

Рис. 1.2.1 Дані операторів мобільного зв'язку щодо якості послуг за І півріччя 2017 року.

Проведення таких вимірів є обов'язковим для всіх мобільних операторів України. Як видно з рис. 1.2.2, у 2017 році 100% операторів мобільного (рухомого) зв'язку України надали дані з якості надаваних послуг.

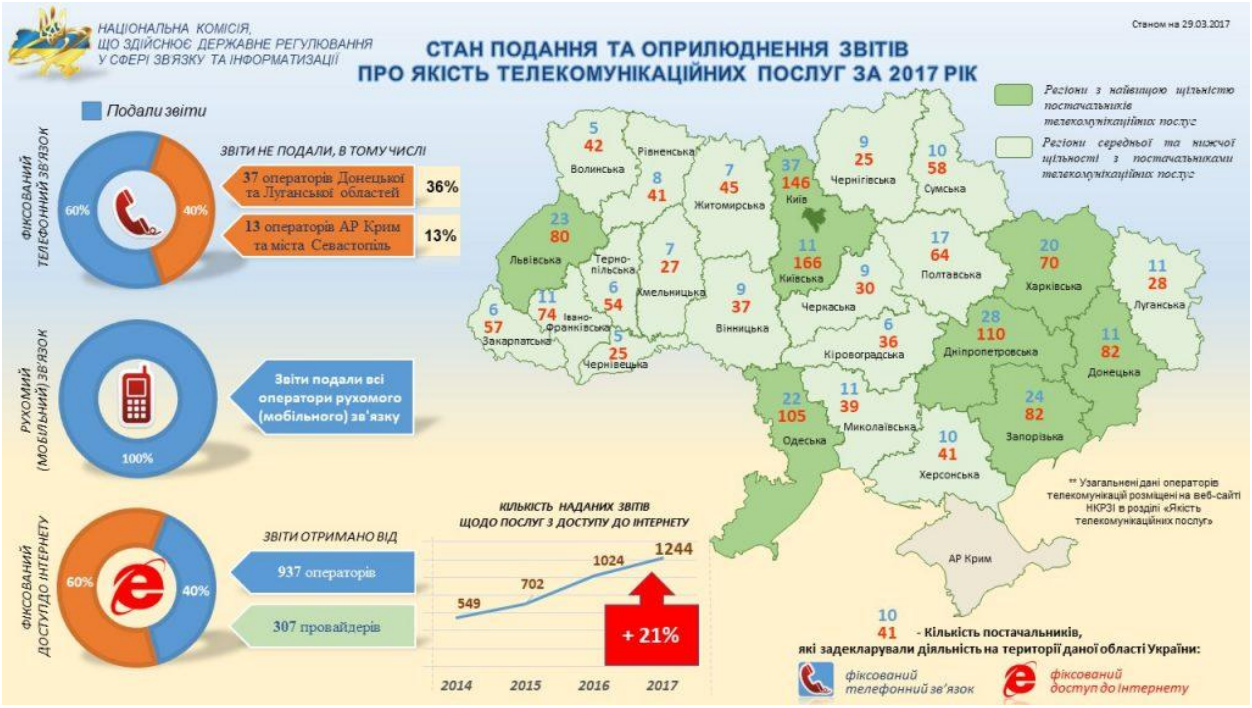


Рис. 1.2.2 Стан подання та оприлюднення звітів про якість телекомунікаційних послуг за 2017 рік.

Аналіз отриманих НКРЗІ даних показав дотримання всіма (100%) операторами рухомого (мобільного) зв'язку встановлених рівнів якості [5], затверджених наказом Мінтрансзв'язку від 19.03.2010 № 147.

Наразі 11 операторів телекомунікацій фіксованого телефонного зв'язку (7% від кількості поданих звітів) зазначили окремі показники якості, рівні яких є нижчими за мінімальні граничні значення, встановлені наказом Мінтрансзв'язку від 22.02.2010 № 91.

Також 4 оператори, які надають послуги доступу до мережі Інтернет (0,5% від кількості поданих звітів), зазначили окремі показники якості, значення яких також є нижчими за граничні норми, встановлені наказом Адміністрації Держспецзв'язку від 28.12.2012 № 803.

Як видно з рис. 1.2.3, звернення з питань надання послуг мобільного зв'язку у загальній кількості звернень складають 24,2% (2579 звернень). У зверненнях споживачами порушувались у більшій масі питання щодо незгоди зі зняттям

коштів з особового рахунку, відсутності доступу (покриття), організації надання послуг, зміни тарифного плану, неякісного надання послуг.

Рис. 1.2.2 Стан подання та оприлюднення звітів про якість телекомунікаційних послуг за 2017 рік.

Таким чином, оператори мобільного зв'язку виявились не лише найбільш відповідальними постачальниками телекомунікаційних послуг, а ще й сегментом ТК послуг, у якому найбільша частина звернень користувачів пов'язана не з якістю зв'язку (відсутності покриття і т.п.), а саме з якістю сервісу (зміни тарифного плану і т.п.)

1.3 Аналіз методу оцінки якості стільникового покриття за допомогою драйв-тестів

Драйв-тест являється одним з найбільш ефективних методів вимірювання якості послуг (QoS), оскільки вимірювання наближені до реального досвіду користувача мобільної мережі.

Вимірювання найчастіше охоплюють території так званих «кластерів». Для кожного кластера готується окремий маршрут (англ. Drive route). При прокладанні маршруту враховують серед іншого: територію, на якій буде проводитися вимір (urban, rural і т.п.), досліджувані послуги, основний рух абонентів, що є для вимірювання часу і багато інших чинників.

Вимірювальний автомобіль рухається по прокладеному маршруту. Це машина, спеціально підготовлена до вимірювань мережі стільникового зв'язку, оснащена вимірювальним комплексом, який дозволяє виміряти всі параметри мережі, необхідні для оцінки якості послуг. Актуальне становище машини визначається за допомогою GPS-приймача, яким оснащена машина. Автомобілю не можна їхати швидше, ніж це було погоджено під час прокладання маршруту.

В драйв-тестах використовується тільки вимірювальне обладнання, що відповідає стандарту ETSI. Кожен автомобіль додатково повинен бути оснащений радіосканерами, які надають можливість провести одночасно незалежне вимірювання сигналу в трьох доступних технологіях: 2G, 3G і LTE.

Завдяки таким вимірам, а також подальшому аналізу їх результатів, оператор мобільного зв'язку отримує інформацію, яка допоможе оптимізувати мережу, тобто провести зміни, що дозволяють збільшити радіопокриття, поліпшити якість або запропонувати клієнтам нові послуги.

1.4 Аналіз існуючих методів проведення драйв-тестів для вимірювання якості послуг

1.4.1 ФУДП Головний радіочастотний центр

Одним із прикладів організацій, що проводять ДТ є ФУДП «ГРЧЦ» (федеральне унітарне державне підприємство «Головний радіочастотний центр»), що підвідомче Роскомнадзору та веде свою діяльність на території РФ. Вимірювальний мобільний комплекс ГРЧЦ рухається по заздалегідь складеним маршрутам, які включають в себе основні міські магістралі та центральні вулиці.

Контрольні виклики здійснюються за допомогою комплексу ASCOM TEMS Automatic (рис.1.3.1) на стаціонарний телефонний номер (call-генератор на території РЧЦ), який також підключений до цифрових ліній E1 фіксованої мережі зв'язку. При цьому комплекс включає до свого складу 4 однакових апарати для дзвінків, оснащених сім картами чотирьох операторів. Таким чином, дзвінки здійснюються синхронно в один і той же час, з одного і того ж місця, що виключає привілейовані умови для будь-якого оператора.

Рис. 1.4.1.1 ASCOM TEMS Automatic змонтований на базі автомобіля ГРЧЦ.

1.4.2 Beeline

Мобільний оператор Beeline, що діє на території РФ, має власні мобільні вимірювальні апаратні комплекси, змонтовані на базі автомобілів Volkswagen Transporter. На рис.1.4.2.1 зображені антени спеціальних стільникових терміналів, розташованих в такому вимірювальному комплексі.

Рис. 1.4.2.1 Антени стільникових терміналів, розташованих у вимірювальному комплексі оператору Beeline.

Мобільні лабораторії «моніторять» покриття на вулицях Москви і області щодня, тим самим регулярно доповнюючи і актуалізуючи карту покриття мережі «Білайн». В обов'язки кожного дослідника входить:

- Контроль працездатності базових станцій після введення в дію нових або модернізації старих;
- Вимірювання радіопокриття в цілях проектування і оптимізації мережі;
- Тестування ділянки мережі після впровадження нового обладнання, або нових налаштувань на мережі, апгрейда системного ПО;
- Контроль доступності мережі та вимірювання якості передачі мови, їх порівняння з іншими операторами;
- Тестування послуг з передачі даних на основі GPRS / EDGE / UMTS і їх порівняння з іншими операторами;
- Оптимізація системних параметрів;
- Пошук джерел електромагнітних завад;

- Пошук і рішення проблем на мережі;
- Вимірювальна підтримка робіт на мережі, що проводяться різними підрозділами технічної дирекції.

Вся отримана інформація записується в режимі реального часу і виводиться на монітор підключеного до комплексу ноутбука (рис. 1.4.2.2). За допомогою цього ноутбука здійснюється контроль параметрів мережі, запуск/зупинка програми радіовимірів, внесення змін в програму вимірювань і т.д. Тобто в кожній точці маршруту пишуться дані про покриття (рівні сигналу) і якість мови. Тривалість дзвінків і всі необхідні паузи регулюються з софта.

Рис. 1.4.2.2 Комплекс вимірювання якості стільникового покриття компанії Beeline.

Дані досліджуються і аналізуються із застосуванням спеціального програмного забезпечення. Досліджуються не тільки виміряні параметри радіосигналу і якості передачі мови, а й кількість зірваних дзвінків, недозвонів, а також їх причини. Оброблені дані передаються в різні підрозділи компанії

«ВимпелКом», наприклад, у відділ оптимізації мережі, служби планування, відділ частотного планування або служби маркетингу.

Слід зазначити, що знаходитись у такому вимірювальному комплексі під час його роботи абсолютно безпечно: по-перше, всі антени знаходяться на даху і орієнтовані так, що діаграма спрямованості своїми пелюстками йде убік і вгору; по-друге, самі антени далеко від людини; по-третє, металевий дах добре екранує. Ефект такий же, як знаходитися в одній кімнаті з людиною, що розмовляє по телефону, тобто щільність випромінювання в десятки раз менше, ніж просто при розмові по телефону без гарнітури.

1.4.3 RFBENCHMARK

Однією з комерційних компаній, що надають результати своїх вимірів операторам мобільного зв’язку є компанія RFBENCHMARK [6]. На рис. 1.4.3.1 зображено вимірювальний комплекс цієї компанії.

Рис. 1.4.3.1 Комплекс вимірювання якості стільникового покриття компанії RFBENCHMARK.

Під час проведення драйв-тесту здійснюються голосові виклики між рухомим терміналом (тестовим) і приймачем (наприклад, ISDN або інший рухомий термінал). Вимірювальні пробні виклики можуть виходити з вимірювального терміналу і надходити в нього - в залежності від встановленої вимірювальної

секвенції. Під час вимірювання для оцінки якості голосу використовується 5-бальна шкала MOS.

Щоб забезпечити об'єктивні і повторювані результати вимірювань, використовується один з двох методів естімації суб'єктивної оцінки якості мовного зв'язку (MOS): PESQ або POLQA. Крім того, збирається інформація про кількість вдалих і невдалих з'єднань, а також прирівняних викликів, про час з'єднання, типи кодування мовного сигналу і багато іншого.

Одночасно з вимірами якості голосових послуг здійснюються вимірювання інтернет-послуг. Приблизна послідовність вимірювання:

1. FTP DL (5x 5MB) - Завантаження файлу розміром 5 Mb — 5 паралельних сесій тестового сервера
2. FTP UL (5x 5MB) - Відправка файлу розміром 5Mb — 5 паралельних сесій тестового сервера
3. HTTP Browsing: Kepler - Завантаження тестової веб-сторінки
4. HTTP Get (10x 5MB) - Завантаження файлу розміром 5Mb – 10 паралельних сесій тестової веб-сторінки
5. Ping (10x 32B) - 10 сесій PING при розмірі пакету 32B
6. Video: YouTube 30sec - Відеострімінг з YouTube упродовж 30 с.

Одними із найважливіших параметрів, що вимірюються, з точки зору користувача, є параметри: Throughput і RTT (Round Trip Delay Time). На основі аналізу двох цих параметрів ми отримуємо інформацію про швидкість інтернету, яку пропонує даний оператор, а також про те, для чого ми можемо використовувати це підключення до мережі. Збираються також дані про кількість вдалих і невдалих сесій FTP, HTTP, Ping, з якої причини сесія не відбулася, за якою технологією працював термінал і багато іншого.

Висновки

Після аналізу існуючих рішень оцінки якості стільникового зв’язку можна зробити висновок, що перспективним є вимірювання якості стільникового

покриття безпосередньо за допомогою користувацьких мобільних апаратів. Адже виміри в такому випадку виробляє багатотисячна аудиторія програми в усіх середовищах, в різні відрізки часу і в будь-яких географічних локаціях.

Але, незважаючи на ці плюси, краудсорсингові виміри містять один великий мінус: їх результати дуже легко оскаржити. Оператор завжди зможе послатися на спотворюючі сигнал чинники. Адже простий користувач, на відміну від спеціальної лабораторії, ніяк не зможе заміряти «чистий сигнал».

Звісно, рухливі лабораторії операторів або спеціальних компаній проводять одночасне паралельне тестування в мережах усіх операторів зв'язку, використовуючи однотипне обладнання. Таким чином на результати оцінки не впливають такі фактори, як різні можливості абонентського обладнання і різні профілі абонентського трафіку в часі і просторі.

Однак, як показав аналіз ринку таких вимірювань, навіть найбільші ТК оператори здатні одночасно використовувати не більше декількох рухомих лабораторій.

Таким чином використання користувацьких мобільних апаратів є доцільним для швидкого виявлення проблем, пов’язаних з якістю стільникового зв’язку та подальшого направлення туди спеціалізованої лабораторії.

2. ОПИС ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ РОЗРОБЛЮВАНОГО ПРОЕКТУ

Для мотивування користувачів встановити додаток, що збиратиме дані про якість покриття стільникового зв’язку необхідно додати до нього функціональність, корисну безпосередньо самому користувачу. Тому розробимо додаток, що буде збирати статистику про переміщення мобільного телефону та виводити її на екран смартфону у вигляді кривої, на якій різними кольорами відокремленої ділянки з різною швидкістю.

Застосунок має 4 основні сторінки:

1. «Головне меню» – дозволяє користувачу розпочати запис шляху, відкрити список збережених записів та перейти на сторінку налаштувань;
2. «Активний запис» – дозволяє користувачу побачити поточний статус активного запису та зупинити його.;
3. «Збережені записи» – дозволяє користувачу отримати інформацію про всі збережені записи та відкрити сторінку «Карта» для будь-якого з них
4. «Карта» - дозволяє користувачу побачити на карті збережений шлях у вигляді ламаної кривої. Колір сегментів кривої залежить від швидкості користувача на цій ділянці шляху.
5. «Налаштування» - дозволяє користувачу виставити налаштування для застосунку, такі як одиниця виміру швидкості і т.п.

Ключовою відмінністю розроблюваного додатку від вже існуючих на ринку є застосування у своїй роботі Google Roads API та Google Elevation API, що дозволяє значно підвищити точність даних щодо переміщень користувача, завдяки прив’язці до існуючих доріг. Це також дозволяє значно підвищити точність наданих користувачу даних щодо його швидкості, оскільки з’являється можливість більш точної оцінки висоти відносно рівня моря у різних точках маршруту користувача.

Має бути реалізована можливість виводити швидкість користувача у метричній або імперській системі мір. Зібрані дані про подорож користувача має зберігатись у локальній базі даних на телефоні користувача.

Водночас з цим додаток має збирати дані про номер бази стільникового зв’язку та рівень сигналу. Зібрані дані про зафіксований смартфоном користувача рівень стільникового зв’язку має зберігатись у окремій базі даних, що розташована на сервері.

Висновок

Виходячи з наведених функціональних можливостей розроблюваного додатку, він має значні переваги відносно існуючих конкурентних продуктів, що дозволить залучитись широким колом користувачів і це в свою чергу дозволить збільшити охоплення клієнтів різних операторів та отримувати дані від декількох користувачів з одного місця одночасно. Це значно підвищить точність отримуваних даних про якість стільникового зв’язку. В свою чергу це є перевагою даного додатку як додатку для збору даних про якість стільникового зв’язку.

3. РОЗРОБКА АРХІТЕКТУРИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ АНАЛІЗУ ЯКОСТІ ПОКРИТТЯ СТІЛЬНИКОВОГО ЗВ'ЯЗКУ

3.1 Загальний огляд архітектури застосунку

Мобільний застосунок складається з двох частин

1. Клієнтська частина;
2. Серверна частина.

Розділення застосунку на дві частини необхідне для збереження у таємниці від користувача ключів доступу для сторонніх ресурсів. Адже для отримання таких даних як карти нам необхідно викликати API (Application programming interface) [7] компанії Google, що у разі виконання такого запиту безпосередньо з мобільного застосунку, може призвести до втрати ключів доступу та неліцензійного доступу до ресурсів Google. Таким чином схема, що зображує взаємодію між мобільним додатком, серверним застосунком та сторонніми API зображена на рис. 3.1.1.

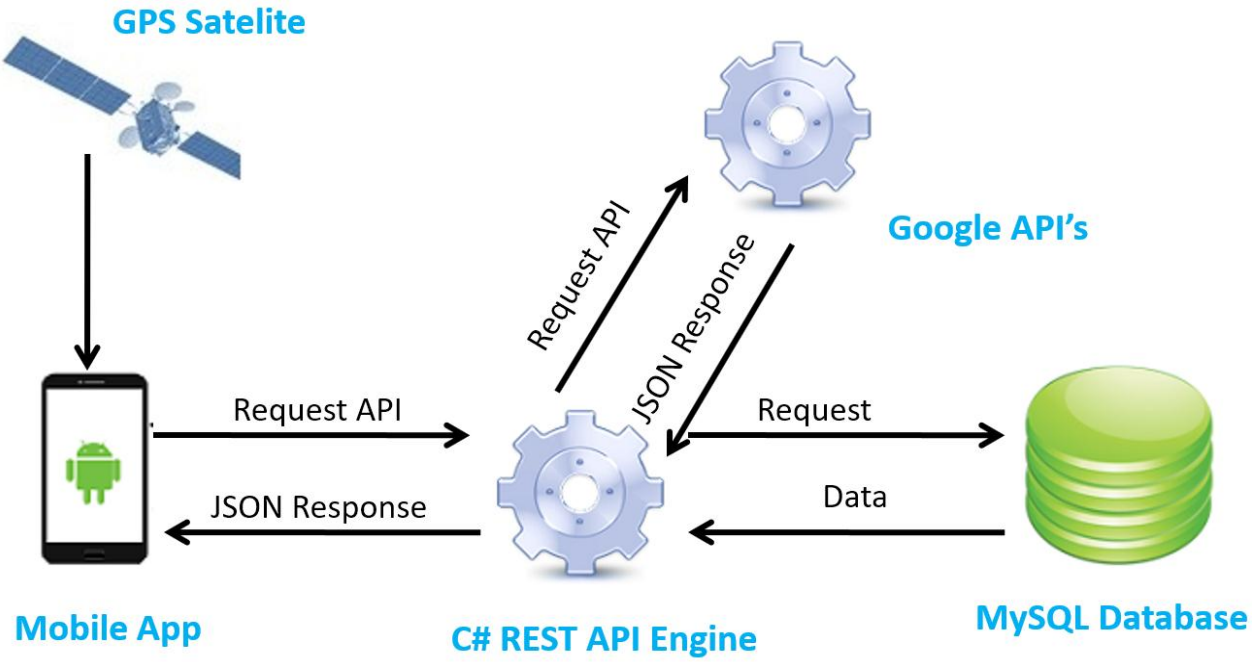


Рис. 3.3.1 Взаємодія між мобільним додатком, серверним застосунком та сторонніми API.

3.2 Розробка архітектури клієнтської частини застосунку

Клієнтську частину застосунку реалізуємо за допомогою фреймворку для кросплатформеної розробки Xamarin [8]. Основними перевагами Xamarin є [9]:

1. Простота освоєння – Xamarin підтримує можливість розробки застосунку за допомогою технології Xamarin.Forms, що використовує для розмітки UI (user interface) мову XAML та C# для бізнес логіки застосунку, знайому більшості .Net розробників;

2. Зниження витрат на проект – за допомогою Xamarin можна розробити застосунки для ОС Android, iOS та UWP, що будуть мати до 90% спільного коду на мові C#;

3. Оптимальні умови для тестування – Xamarin має потужний вбудований емулятор мобільних пристроїв;

4. Вичерпна документація – незважаючи на низьку популярність серед спілки програмістів Xamarin має вичерпну документацію завдяки підтримці компанії Microsoft.

Недоліки Xamarin:

1. Неможливість використовувати негативні елементи UI для кожної з ОС;

2. Більший розмір застосунку у порівняні з застосунками на основі Java.

Окрім зазначених переваг слід зазначити, що Xamarin є оптимальним вибором у випадку інтеграції з серверною частиною застосунку на базі .Net, оскільки дозволяє імпортувати без змін використані на сервері моделі даних.

Слід зазначити, що додатки на базі Xamarin прийнято розроблювати з використанням патерну MVVM (Model - View - ViewModel) [10], що ґрунтується на поділі функціональної частини програми на три ключові компоненти:

1. View - представлений або призначений для користувача інтерфейс;

2. Model - модель або дані, які використовуються в додатку;

3. ViewModel - проміжний шар між поданням і даними, який забезпечує їх взаємодію.

Перевагою використання даного патерну є менша зв'язаність між компонентами і поділ відповідальності між ними. Тобто Model відповідає за дані, View відповідає за графічний інтерфейс, а ViewModel - за логіку програми. Схема, що відображає зв’язки між компонентами програми зображена на рис. 3.2.1.

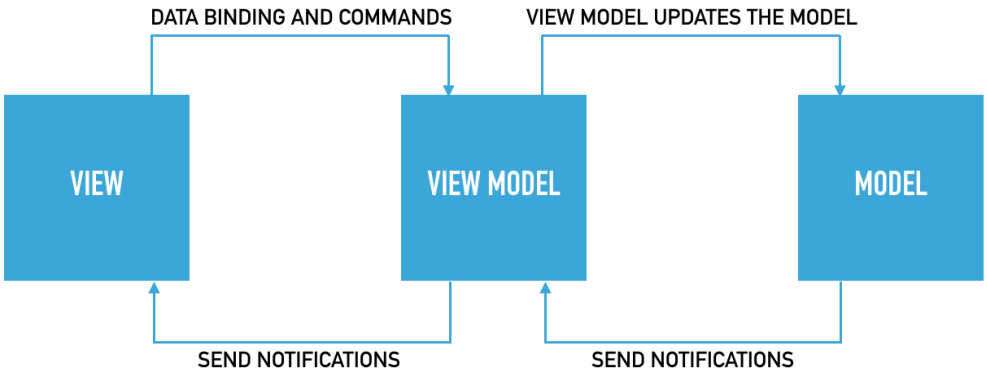


Рис. 3.2.1 Model - View - ViewModel pattern [11].

Слід зазначити, що фреймворк Xamarin використовує для опису користувацького інтерфейсу мову XAML [12] (англ. eXtensible Application Markup Language). XAML – це декларативна мова опису розмітки. З точки зору моделі програмування фреймворку .NET Framework мова XAML спрощує створення користувацького інтерфейсу для програм на базі даного фреймворку, тобто на базі технологій WPF (Windows Presentation Foundation) та Xamarin. За допомогою XAML з’являється можливість створити видимі елементи інтерфейсу користувача у вигляді файлу, описаного декларативною мовою розмітки XAML, а потім відокремити визначення користувача інтерфейсу від безпосередньо шару бізнес логіки, використовуючи файли коду програмної частини у форматі .cs, приєднані до розмітки за допомогою визначень розділених класів (тобто класів, описаних з використанням ключового слова «partial»). Мова XAML надає можливість створення екземплярів об'єктів з набору резервних типів, визначених у межах збірки. У цьому полягає відмінність XAML від більшості інших мов розмітки (таких як XML), які, як правило, є інтерпретованими мовами без зв'язку з системою

описаних у програмі типів. Мова XAML забезпечує робочий процес, що дозволяє декільком учасникам одночасно працювати над користувацьким інтерфейсом та бізнес логікою програми, уникаючи навіть проблем при злиті своїх змін у єдиний репозиторій, оскільки зміни відбуваються у різних файлах, що дозволяє уникати конфліктів версій.

При поданні у вигляді тексту файли XAML є XML-файлами, які зазвичай мають розширення .xaml. Файли можна зберігати в будь-якому кодуванні, що підтримує XML, але зазвичай використовується кодування UTF-8.

3.3 Розробка архітектури серверної частини застосунку

Серверну частину застосунку побудовано за допомогою платформи ASP.NET Core [13].

Одним із характерних моментів платформи ASP.NET Core є застосування патерну MVC [14]. Причому остання версія MVC-фреймворка, який застосовується в ASP.NET Core, має номер 2.1. Тому важливо не плутати ASP.NET MVC 5, який застосовується в ASP.NET 4.5-4.8, і фреймворк MVC, який застосовується в ASP.NET Core. Хоча в багатьох аспектах ці фреймворки будуть збігатися.

Сам патерн MVC (рис. 3.1.1) не є якоюсь новою ідеєю в архітектурі додатків, він з'явився ще в кінці 1970-х років в компанії Xerox як спосіб організації компонентів в графічному додатку на мові Smalltalk.

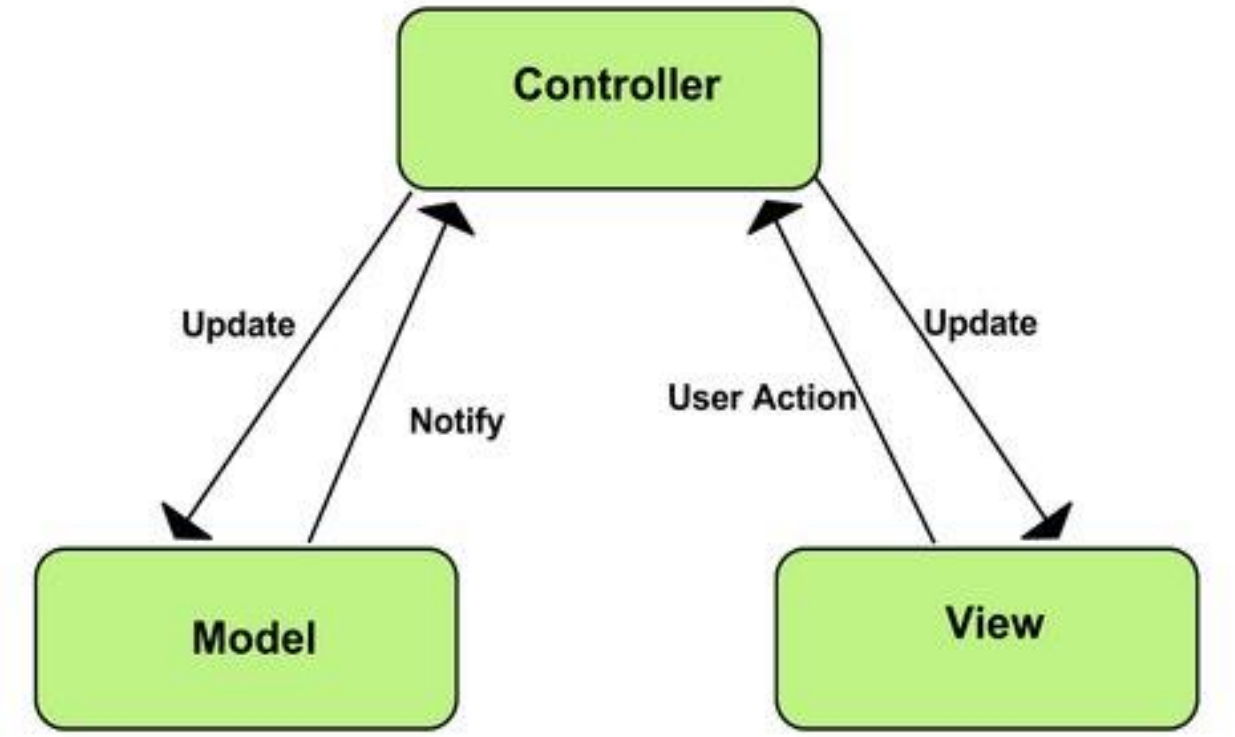


Рис. 3.3.1 Model-View-Controller pattern [15].

Концепція патерна MVC (Model-View-Controller) передбачає поділ застосунку на три компоненти:

1. Модель (model): описує використовувані в додатку дані, а також логіку, яка пов'язана безпосередньо з даними, наприклад, логіку валідації даних. Як правило, об'єкти моделей зберігаються в базі даних.

У MVC моделі представлені двома основними типами: моделі уявлень, які використовуються уявленнями для відображення і передачі даних, і моделі домену, які описують логіку управління даними.

Модель може містити дані, зберігати логіку управління цими даними. У той же час модель не повинна містити логіку взаємодії з користувачем і не має визначати механізм обробки запиту. Крім того, модель не повинна містити логіку відображення даних в поданні.

2. Подання (view): відповідають за візуальну частину або користувацький інтерфейс, нерідко html-сторінка, через який користувач взаємодіє з додатком. Також уявлення може містити логіку, пов'язану з відображенням даних. У той же

час уявлення не повинно містити логіку обробки запиту користувача або управління даними.

3. Контролер (controller): представляє центральний компонент MVC, який забезпечує зв'язок між користувачем та програмою, поданням і сховищем даних. Він містить логіку обробки запиту користувача. Контролер отримує введені користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді відповіді, наповненої даними моделей.

3.4 Проектування взаємодії серверного застосунку та мобільного додатку

Комунікацію серверного застосунку та мобільного додатку реалізуємо за допомогою протоколу HTTP (HyperText Transfer Protocol)[16].

Останніми роками великої популярності при проектуванні розподілених гіпер-медіа систем набули застосунки розроблені з урахуванням принципів REST (Representational State Transfer).

REST - це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який, як правило, використовується для побудови веб-служб [17].

У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат.

Як відбувається управління інформацією сервісу - це цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайний HTTP. Так ось, для HTTP дія над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST.

Обов’язкові вимоги до застосунків з REST архітектурою:

1. Модель клієнт-сервер.

Першим обмеженням є приведення архітектури до моделі клієнт-сервер. Розмежування потреб є принципом, що лежить в основі даного обмеження. Відділення потреби інтерфейсу клієнта від потреб сервера, що зберігає дані, підвищує переносимість коду клієнтського інтерфейсу на інші платформи, а спрощення серверної частини покращує масштабованість.

2. Відсутність стану

Протокол взаємодії між клієнтом і сервером вимагає дотримання наступного умови: в період між запитами клієнта ніяка інформація про стан клієнта на сервері не зберігається (Stateless protocol). Всі запити від клієнта повинні бути складені так, щоб сервер отримав всю необхідну інформацію для виконання запиту. Стан сесії при цьому зберігається на стороні клієнта

3. Кешування

Проміжні вузли, можуть виконувати кешування відповідей сервера. Відповіді сервера, в свою чергу, повинні мати явне або неявне позначення актуальності метою запобігання отримання клієнтами застарілих або невірних даних у відповідь на наступні запити. Правильне використання кешування здатне повністю або частково усунути деякі клієнт-серверні взаємодії, ще більше підвищуючи продуктивність і розширюваність системи.

4. Одноманітність інтерфейсу

Наявність уніфікованого інтерфейсу є фундаментальним вимогою дизайну REST-сервісів. Уніфіковані інтерфейси дозволяють кожному з сервісів розвиватися незалежно.

5. Шари абстракції

Клієнт зазвичай не здатний точно визначити взаємодіє він безпосередньо з сервером або ж з проміжним вузлом в зв'язку з ієрархічною структурою мереж (маючи на увазі, що така структура утворює шари). Застосування проміжних серверів здатне підвищити масштабованість за рахунок балансування навантаження і розподіленого кешування. Проміжні вузли також можуть підкорятися політиці безпеки з метою забезпечення конфіденційності інформації.

6. Код за вимогою

REST може дозволити розширити функціональність клієнта за рахунок завантаження коду з сервера у вигляді сценаріїв. Це обмеження дозволяє проектувати архітектуру, підтримуючу бажану функціональність в загальному випадку, за винятком деяких контекстів.

Таким чином архітектура REST дуже проста в плані використання. По виду запиту відразу можна визначити, що він робить, не розбираючись в форматах (на відміну від SOAP, XML-RPC). Дані передаються без застосування додаткових шарів, тому REST вважається менш ресурсномістких, оскільки не треба просити запит, щоб зрозуміти, що він повинен зробити і не треба переводити дані з одного формату в інший.

Оскільки наш проект потенційно може використовуватись як агрегатор даних для мобільного додатку з функцією навігації, то будемо дотримуватись принципів REST.

Висновок

Отже, у якості серверного застосунку було побудовано веб застосунок на базі .Net Core Framework.

Слід зазначити, що виходячи з реального навантаження на застосунок можливо матиме сенс додати аутентифікацію до застосунку, аби унеможливити сторонню DDoS атаку.

4. РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ЗАСТОСУНКУ

4.1 Побудова діаграми залежностей класів

Як зазначалось у розділі 3, реалізуємо нашу серверну частину на основі фреймворку ASP.NET Core. Основним контролером серверного застосунку є клас SpeedServerController, що приймає http запит, перевіряє його на коректність, після чого модифікує за допомогою веб-сервісів Google Roads API [18] та Google Elevation API [19], записує дані запиту до серверної бази даних та повертає модифікований запит з вилученими відомостями про якість стільникового покриття користувачу за допомогою http запиту. Діаграма залежності класів застосунку зображена на рис.4.1.1

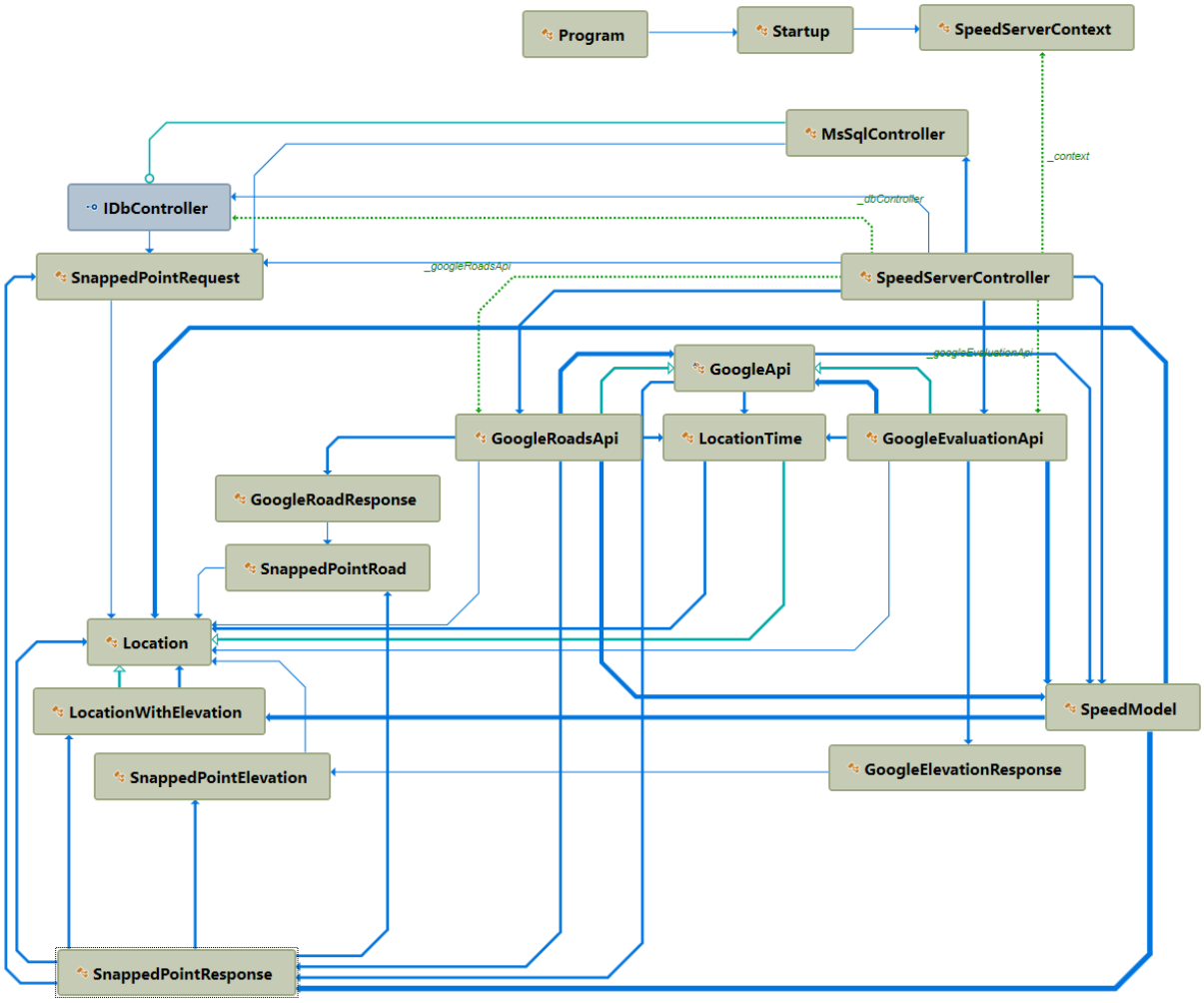


Рис. 4.1.1 Type Dependencies діаграма.

Як видно з наведеної вище діаграми, є SnappedPointRequest (міститься у запиті мобільного додатку до back-end частини застосунку) та SnapperPointResponse (міститься у відповіді back-end частини застосунку). У табл. 4.1 наведено порівняння полів, що містяться у цих моделях.

Таблиця 4.1 – Порівняння моделей SnappedPointRequest та SnapperPointResponse.

		Назва моделі	
Назва поля	Тип поля	SnappedPointRequest	SnapperPointResponse
Location.latitude	double	+	+
Location.longitude	double	+	+
LocationWithElevation.elevation	double		+
time	DateTime	+	+
originalIndex	int		+
placeId	string		+
cellData	string	+	

Як видно з наведеної таблиці 4.1, SnappedPointRequest містить унікальне поле «cellData». Це поле відповідає за будь-який обраний нами параметр якості стільникового зв’язку, відповідно ці дані ми отримуємо від мобільного додатку, зберігаємо у БД, що розміщена на сервері та не повертаємо у SnapperPointResponse, оскільки ці дані не потрібні мобільному клієнту для реалізації основного функціоналу програми. У той же час модель SnapperPointResponse містить унікальні поля LocationWithElevation.elevation (отримане з Google Elevation API), placeId та originalIndex (отримані з Google Roads API).

Слід зазначити, що поле elevation є частиною класу LocationWithElevation, що розширює клас Location. Повні ієрархічні залежності реалізованих класів відображені також на рис. 4.1.

4.2 Реалізація збільшення точності даних GPS шляхом прив’язування позиції до дороги

Задля збільшення точності отриманих даних про позицію клієнтського смартфону у момент запису даних про якість стільникового зв’язку використаємо службу Google Roads Api.

Roads API використовує до 100 точок GPS, зібраних по маршруту, і повертає аналогічний набір даних, причому точки у відповіді служби прив'язані до найбільш вірогідних доріг, за якими подорожував автомобіль. При бажанні ми можемо запросити інтерполяцію точок, в результаті чого переданий набір точок буде апроксимований додатковими локаціями.

Як і усі служби Google API, Roads API приймає запит у вигляді сконфігурованого URL, що відповідає виду «https://roads.googleapis.com/v1/snapToRoads?path= -35.27801,149.12958|-35.28032,149.12907&interpolate=true&key= msRMH1AbW7qABM», де key - ключ API нашого застосунку, а path – масив географічних точок розділених символом «|».

Слід зазначити, що реалізований клас GoogleRoadsApi підтримує розділення запиту від мобільного додатку на декілька запитів до служби Google API, задля дотримання вимог про максимальну довжину запиту.

4.3 Реалізація збільшення точності швидкості мобільного пристрою через використання служби Google Elevation API

Задля збільшення точності користувацьких даних про швидкість необхідно враховувати різницю висоти між кожними двома послідовно записаними точками маршруту. Звісно, будь-який модуль GPS повертає також дані про висоту пристрою, однак ми можемо збільшити точність цих даних, використавши Google Elevation API.

API Elevation забезпечує дані про висоти для всіх місць на поверхні Землі, включаючи розташовані на глибині світового океану.

Google Elevation API приймає запит у вигляді сконфігурованого URL, що відповідає виду «https://maps.googleapis.com/maps/api/elevation/json?locations=39.7391536,-104.9847034&key= msRMH1AbW7qABM», де key - ключ API нашого застосунку, а locations – масив географічних точок розділених символом «|».

Реалізований клас Google Elevation Api підтримує розділення запиту від мобільного додатку на декілька запитів до служби Google API, задля дотримання вимог про максимальну довжину запиту у 512 точок.

4.4 Публікація серверної частини додатку у мережі інтернет

Після завершення роботи над додатком, його можна опублікувати, щоб він став доступний з будь-якого пристрою у будь-якій мережі, а не тільки з localhost.

Традиційно веб-сервер IIS (Internet Information Services) застосовується для розгортання веб-додатків. Для розміщення веб-додатків ASP.NET Core також може застосовуватися IIS [20], тільки на відміну від попередніх версій ASP.NET, тепер його роль буде зводитися до проксі-сервера. Цей спосіб є для нас не самим оптимальним, оскільки нам потрібна мережа зі статичним IP та постійно увімкнений комп’ютер у цій мережі.

Набагато оптимальнішим виходом є розміщення нашого додатку на одному із зовнішніх хостингів. Традиційно найбільшою популярністю користуються Amazon [21], Google Cloud [22] та Azure [23]. Задля економії фінансів на початковому етапі життєвого циклу нашого застосунку використаємо хостинг SmarterASP.NET [24]. На рис 4.4.1 відображені тарифи на використання даного хостингу, оберемо тариф «ADVANCE» [25].

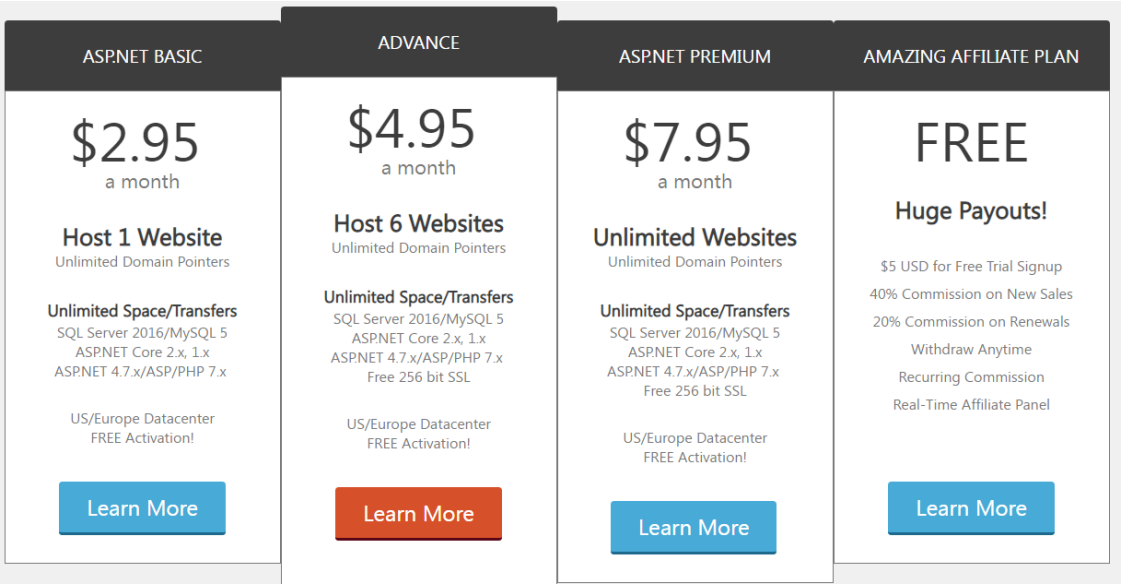


Рис. 4.4.1 Тарифи SmarterASP.NET.

Після публікації без урахування БД та логування наш додаток займає 42 Mb (рис.4.4.2).

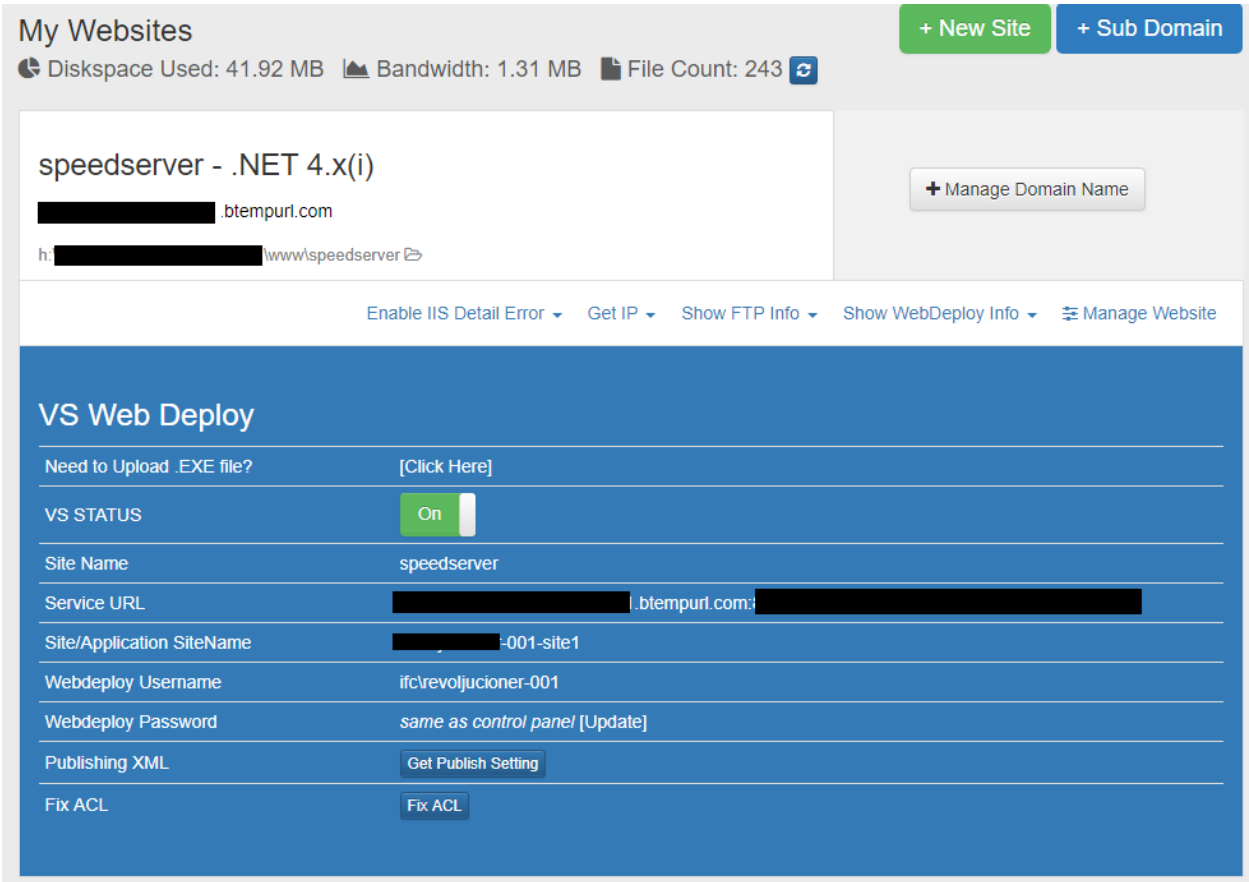


Рис. 4.4.2 Контрольна панель обраного хостингу.

Висновок

Можна зробити висновок, що запропоноване рішення щодо реалізації серверного застосунку є оптимальним, адже затримки на виконання запиту від мобільного додатку є мінімальними і у більшій мірі визначаються тривалістю відповіді сервісів Google.

Серед існуючих недоліків запропонованого рішення можна назвати лише ненадійний хостинг. Експериментально виявлено, що перший запит після відсутності запитів до серверного застосунку впродовж довгого часу (близько доби) виконується з додатковою затримкою, близько 15-20 секунд. Це можливо вирішити обравши у перспективі більш дорогий, але надійний хостинг.

5. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

5.1 Побудова діаграми залежностей класів

Як зазначалось у розділі 3, реалізуємо нашу мобільну частину додатку на основі фреймворку Xamarin. Діаграма залежностей класів зображена на рис.5.1.1.

Як видно з наведеної на рис.5.1.1 діаграми, основні класи для спілкування мобільного додатку та серверного застосунку мають ті ж назви, що моделі, наведені у розділі 4. Це пояснюється тим, що ці моделі були опубліковані у вигляді Dll (Dynamic Link Library) та опубліковані на Artifactory [26] – менеджері бінарних сховищ. Саме цю сформовану бібліотеку було використану у обох частинах розроблюваного додатку, що допомогло знатною мірою зменшити кількість програмного коду.

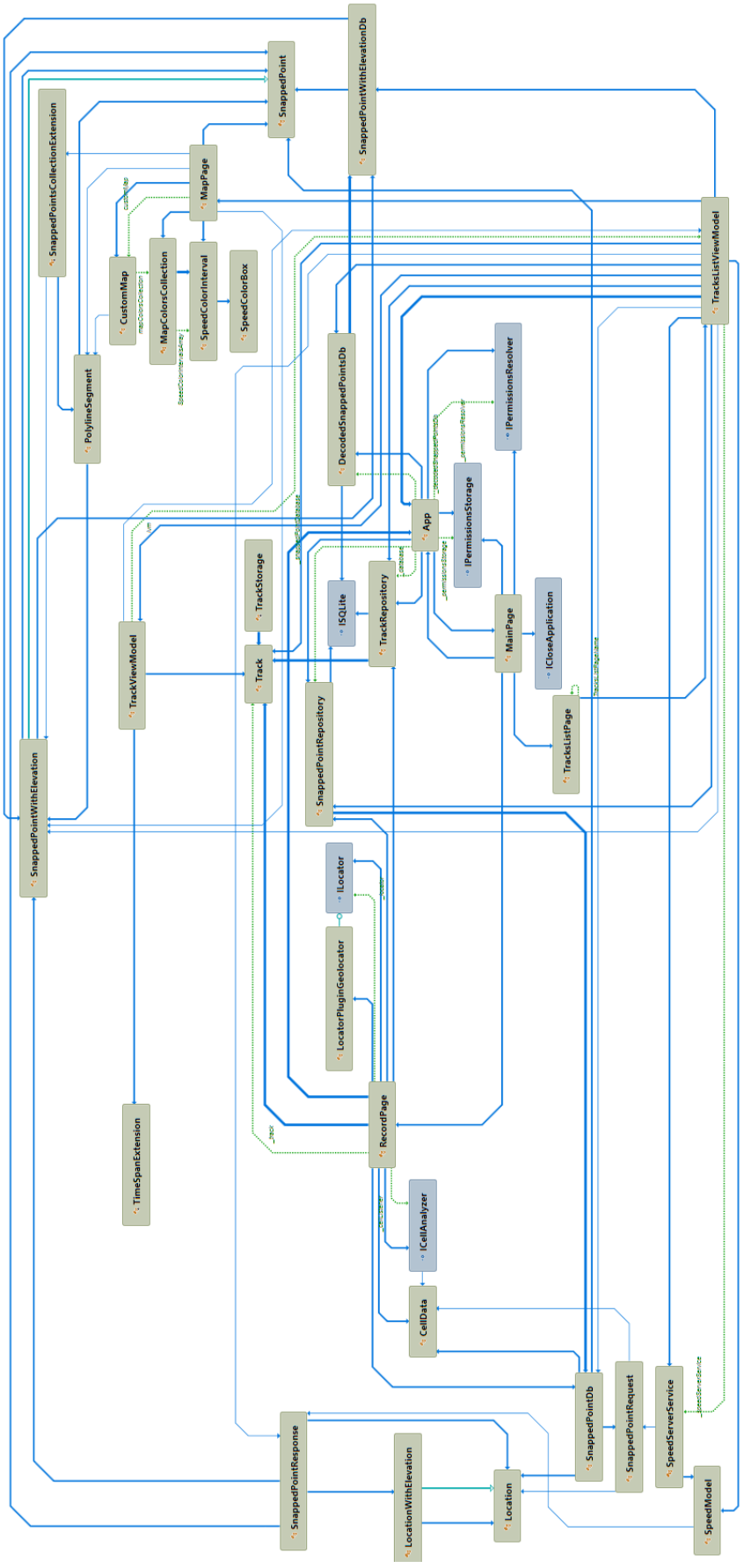


Рис. 5.1.1 Діаграма залежностей класів мобільного додатку.

5.2 Розробка бази даних мобільного додатку

Для збереження на смартфоні користувача даних про його переміщення необхідно реалізувати на мобільному апараті локальну базу даних. Найбільш оптимальним варіантом для інтеграції с фреймворком Xamarin є SQLite [27].

SQLite - компактна вбудована СУБД (система управління базами даних). Вихідний код бібліотеки переданий в суспільне надбання.

Слово «вбудована» означає, що SQLite не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремо працюючим процесом, з яким взаємодіє програма, а являє собою бібліотеку, з якої програма компонується, і рушій стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виповнюється програма. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції записи весь файл, який зберігає базу даних, блокується; ACID-функції досягаються в тому числі за рахунок створення файлу журналу.

Кілька процесів або потоків можуть одночасно, без будь-яких проблем, читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, якщо ніяких інших запитів в даний момент не обслуговується; в іншому випадку спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу.

У комплекті поставки SQLite йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина є кросплатформеною утилітою командного рядка.

Завдяки архітектурі рушія можливо використовувати SQLite як на вбудованих системах, так і на виділених машинах з гігабайтними масивами даних.

Також перевагою SQLite є наявність безлічі відкритих бібліотек для взаємодії з нею через .Net Framework.

Основним недоліком SQLite є неможливість назначати тільки поля типу «int» первинним ключем таблиці. Це пояснюється тим, що будь-який первинний ключ у таблиці SQLite пов’язується зі значенням у скритому від користувача полі «rowid». Таким чином лише при використані змінної типу «int» у якості первинного ключа СУБД записує це саме значення і у поле «rowid», що значно пришвидшує подальшу роботу з таблицею.

Окрім того позначити поле як «AutoIncrement», аби первинний ключ сам генерувався можливо лише для поля типу «int», що унеможливлює використання первинного ключа типу «string», згенерованого у вигляді GUID (Globally Unique Identifier).

Створимо у нашій базі даних на смартфоні 3 таблиці:

1. TrackRepository.

Містить записи типу Track. Даній таблиці відповідає клас TrackRepository. Ця таблиця відповідає за збереження основних відомостей про запису подорожей користувача.

2. SnappedPointRepository

Містить записи типу SnappedPointDb. Даній таблиці відповідає клас SnappedPointRepository. Ця таблиця відповідає за збереження даних про позицію користувача під час подорожі, з зазначенням часу під час якого ці дані були отримані, рівнем сигналу базової станції, ідентифікатором базової станції та ідентифікатором запису подорожі (Track) під час якої ці дані були записані.

3. DecodedSnappedPointsRepository

Містить записи типу SnappedPointWithElevationDb. Даній таблиці відповідає клас DecodedSnappedPointsDb. Ця таблиця відповідає за збереження модифікованих за допомогою серверного застосунку даних про позицію користувача під час подорожі, з зазначенням часу під час якого ці дані були отримані, висотою над рівнем моря, та ідентифікатором запису подорожі (Track) під час якої первинні дані були записані.

У табл. 5.2.1 а структура моделі Track.

Таблиця 5.2.1 – Структура моделі Track.

Назва поля	Тип поля
Id	int
StartDateTime	int
EndDateTime	double
StatusActive	bool
Decoded	bool

Як видно з табл. 5.2.1 модель Track також має поля StatusActive та Decoded типу «bool», що призначені для ідентифікації чи є записана подорож репрезентативною та чи пройшла вона модифікування через серверний застосунок відповідно.

У табл. 5.2.2 наведено порівняння полів, що містяться у зазначених моделях.

Таблиця 5.2.2 – Порівняння моделей SnappedPointDb та SnappedPointWithElevationDb.

		Назва моделі	
Назва поля	Тип поля	SnappedPointDb	SnappedPointWithElevationDb
Id	int	+	+
TrackId	int	+	+
Latitude	double	+	+
Longitude	double	+	+
Elevation	double		+
Time	DateTime	+	+
Cid	int	+	
CellSignalStrength	string	+	

Як видно з табл. 5.2.2, моделі SnappedPointDb та SnappedPointWithElevationDb по суті є модифікованими для збереження у базі даних моделями SnappedPointRequest та SnapperPointResponse, що були описані у пункті 4.1.

5.3 Розробка головного меню

Виконаємо головне меню у вигляді списку з чотирьох пунктів для переходу до сторінок Запису подорожі, Списку записаних подорожей, сторінки налаштувань та для виходу з додатку. У коді даній сторінці відповідає клас «Log.Pages.MainPage».

Стандартний інтерфейс додатків на базі фреймворку Xamarin мають синій колір. Збережемо цей існуючий фірмовий стиль Xamarin вибравши для основних елементів управлінь на головній сторінці кольори DeepSkyBlue, DodgerBlue, RoyalBlue та Blue. Вигляд головного меню зображено на рис. 5.3.1.

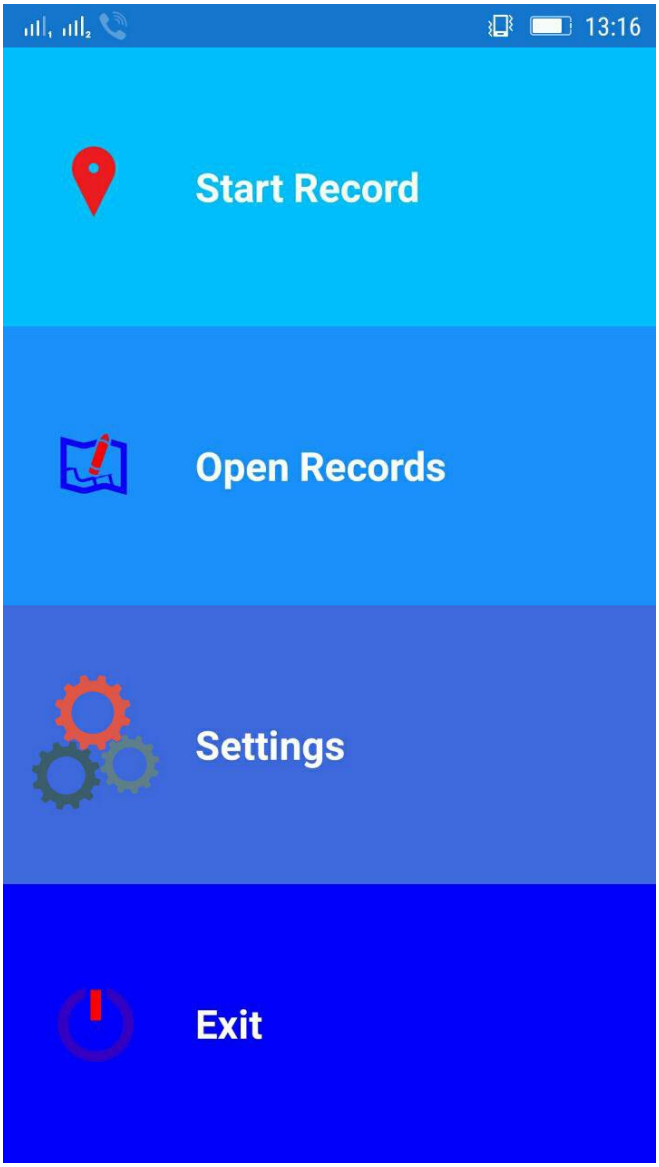
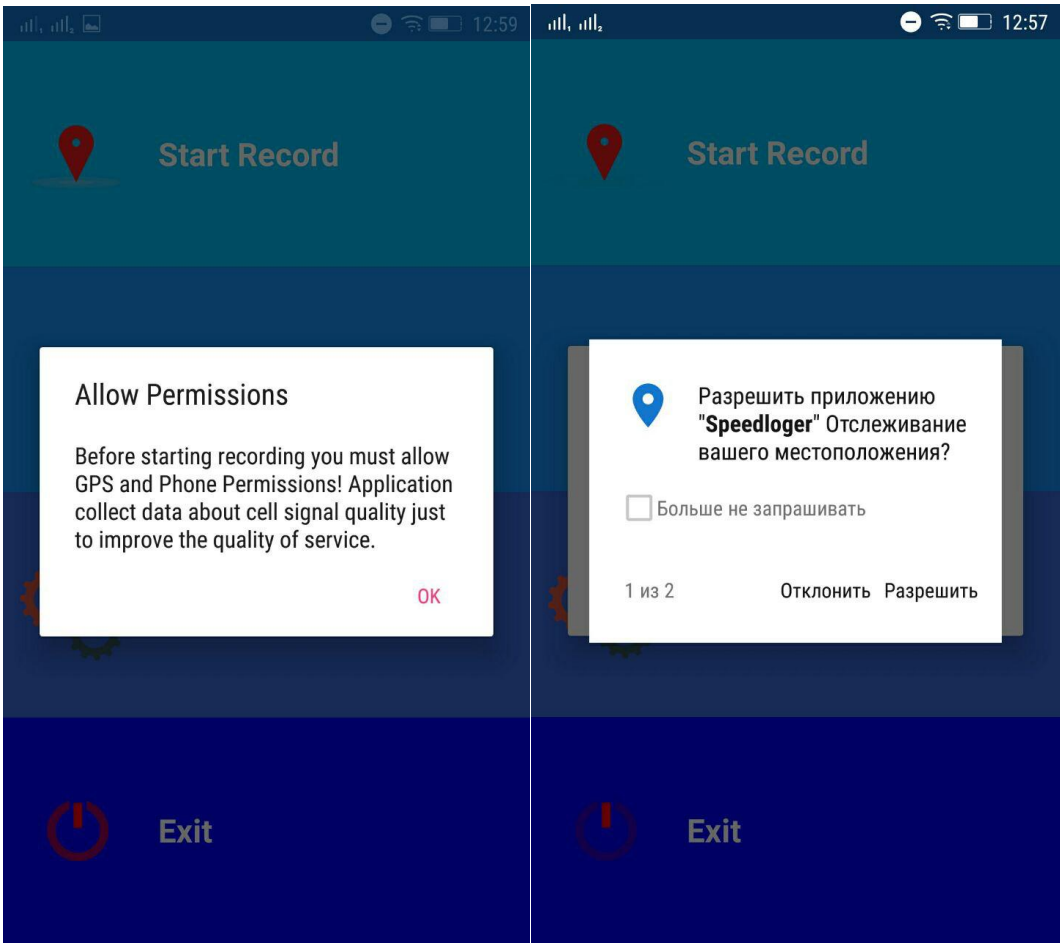


Рис. 5.3.1 Вигляд головного меню розроблюваного додатку.

Як видно з рисунку 5.3.1 написи виконані у білому кольорі, це зроблено для більшої контрастності.

Для кращого сприйняття розроблюваної сторінки кожен пункт меню має анімовану анімацію, що завантажена у проект у форматі Json та додана на сторінку як елемент «forms:AnimationView». Завантажити та відреагувати подібні анімації для кросплатформених додатків можна за допомогою безлічі бібліотек, нами серед них було обрано «Lottie» [29].

Натискання на пункт меню «Start Record» має викликати перевірку наявності необхідних дозволів від операційної системи та при необхідності або їх запросити (рис. 5.3.2), або розпочати запис шляху та відкрити сторінку «Запис» (RecordPage).



а) б)

Рис. 5.3.2 Вигляд інформаційного повідомлення про необхідність надати додатку запитувані дозволи (а), та безпосередньо запиту на отримання необхідних дозволів від операційної системи Android (б).

Натискання на пункт меню «Open Records» відкриває сторінку списку існуючих записів.

Натискання на пункт меню «Settings» відкриває сторінку налаштувань, де користувач може змінити такі налаштування як, наприклад, одиниця виміру швидкості.

Натискання на пункт меню «Exit» закриває мобільний додаток.

5.4 Розробка сторінки запису шляху

Сторінка «Запис» відображає поточну швидкість смартфону, тривалість поточного запису. Також реалізована кнопка «Стоп», що дозволяє зупинити поточний запис шляху. У коді даній сторінці відповідає клас «Log.Pages. RecordPage».

Данна сторінка є допоміжною, вона не відображає критично потрібної користувачу інформації та має бути замінена у майбутньому повідомленням у панелі повідомлень. Вигляд сторінки «Запис» зображено на рис. 5.4.1.

Рис. 5.4.1 Вигляд сторінки «Запис» розроблюваного додатку.

Слід зазначити, що вся логіка щодо отримання даних з датчику GPS (Global Positioning System) та запис їх у базу даних зосереджена безпосередньо у цьому класі

Отже після відкриття даної сторінки одразу в базу даних TrackRepository, що була описана у розділі 5.2 записується новий екземпляр класу Track з автоматично згенерованим полем id. Поля Decoded та StatusActive мають значення false. При надходженні від ILocator даних про зміну поточної позиції користувача більше ніж на 10 метрів (але не частіше, ніж раз на 500 мілісекунд) у базу даних

SnappedPointRepository записуються дані про поточні географічні координати (широту та довготу), рівень сигналу базової станції та її унікальний ідентифікатор, id записаної у таблицю TrackRepository подорожі.

Слід зазначити, що дані про географічне розташування апарату та рівень стільникового сигналу отримуються залежно від операційної системи смартфону і тому вони викликаються через зміні типу інтерфейсів ILocator там ICellAnalyzer. Значення цих полів присвоюються під час ініціалізації сторінки завдяки вбудованому методу Get<T>() класу DependencyService, що працює завдяки рефлексії та шукає у платформо залежному проекті класи, що реалізують шуканий інтерфейс.

Відповідно програмний інтерфейс ILocator зобов’язує класи, що його реалізують реалізувати метод StartListening та StopListening, що обидва приймають параметр типу EventHandler<PositionEventArgs> та повертають зміну типу Task.

У той же час програмний інтерфейс ILocator зобов’язує класи, що його реалізують реалізувати метод GetCellData, що не має вхідних параметрів та повертають зміну типу CellData, що у свою чергу містить унікальний ідентифікатор базової станції та поточний рівень прийому стільникового сигналу.

При натисненні кнопки «Stop» відбувається перевірка кількості записаних точок маршруту користувача, при недостатній довжині маршруту користувачу видається повідомлення про помилку (рис. 5.4.2). Така перевірка аби у подальшому не враховувати записи подорожей, під час яких користувач знаходився на одному місці.

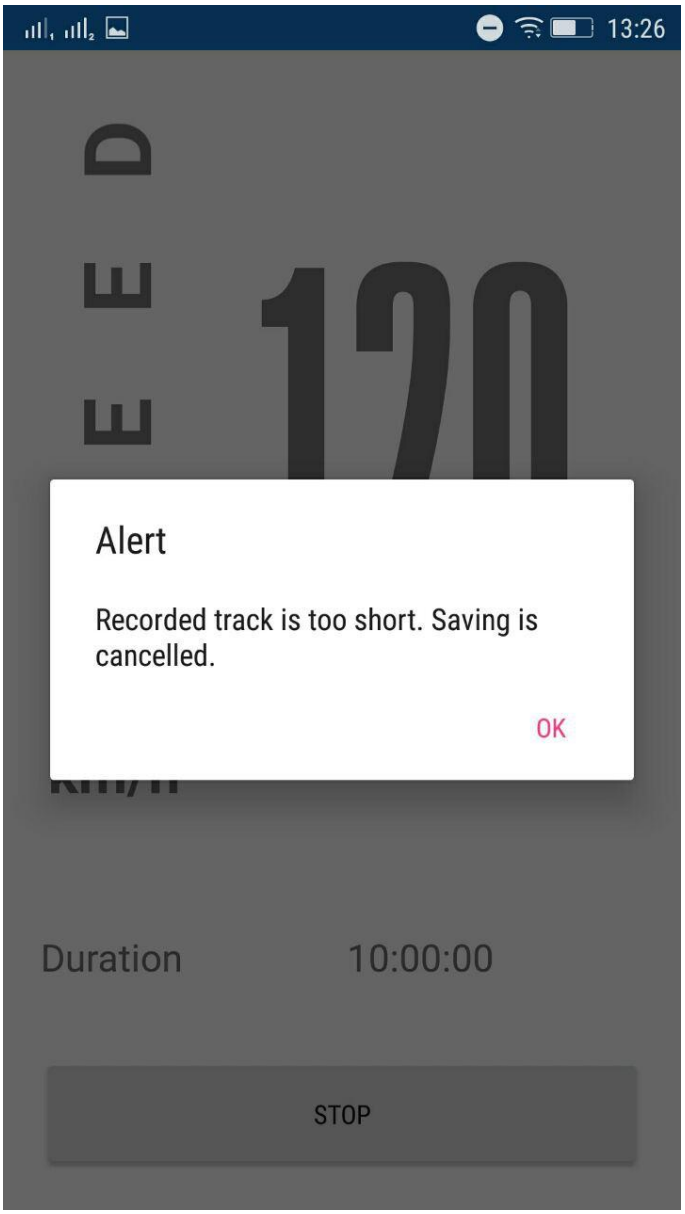


Рис. 5.4.2 Одне з можливих повідомлень про помилку при спробі зберегти записаний маршрут.

У разі якщо записані дані пройшли перевірку на репрезентативність, то запис про дану подорож у таблиці TrackRepository модифікуєтеся, поле StatusActive набуває значення true. Після закриття транзакції до бази даних користувач перенаправляється на головну сторінку додатку.

5.5 Розробка сторінки існуючих записів

Сторінка «Список записів» відображає всі існуючі у локальній базі даних записи маршрутів користувача, тобто всі елементи Track з бази даних

TrackRepository зі значенням поля StatusActive true. Виконана сторінка за допомогою стандартного елементу ListView. Оскільки дана сторінка використана за допомогою описаного у пункті 3.2 даної роботи патерну проектування MVVM, то їй відповідають класи TracksListPage, TrackViewModel та TracksListViewModel. Зовнішній вигляд сторінки «Список записів» зображено на рис. 5.5.1

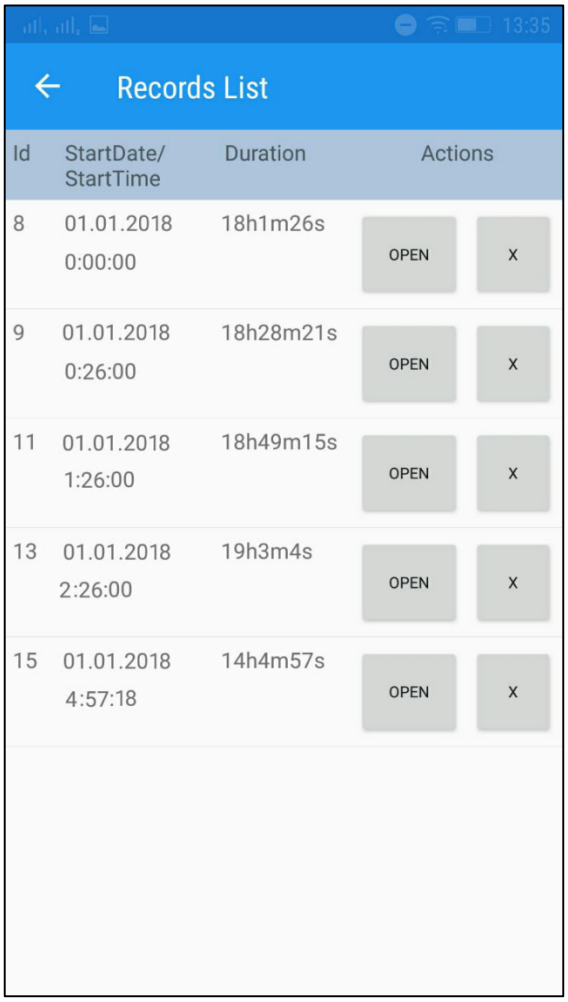


Рис. 5.5.1 Вигляд сторінки «Список записів».

Як видно на рис. 5.4.1, кожен елемент списку існуючих записів містить поля id, StartDate/StartTime, що отримуються безпосередньо з моделі Track, та кнопки Open, Delete. Поле Duration вираховується у класі TrackViewModel у властивість даного класу, форматування

Взаємодія з кнопками Open та Delete викликає методи OpenTrackCommand та DeleteTrackCommand відповідно з класу TracksListViewModel.

Також, я видно з рис 5.5.1 ця сторінка на відміну від описаних раніше має у «шапці» кнопку «Назад», що повертає користувача до головного меню додатку.

При взаємодії з кнопкою «Open» перевіряється значення поля Decoded відповідної моделі Track. Якщо Decoded має значення true, то викликається команда відкриття сторінки «Map», куди в якості параметру передається зазначене поле id моделі Track. Якщо Decoded має значення false, то через http запит викликається сервіс SpeedServerGate нашого серверного застосунку, параметром передається сериалізований за допомогою класу JsonConvert з вільної програмної бібліотеки Newtonsoft.Json масив об’єктів типу SnappedPointRequest, отриманих з таблиці SnappedPointRepository, куди параметром передається поле id моделі Track. Відповідь від серверного застосунку, як було описано у розділі 4 представляє собою модель SpeedModel, що містить у собі масив об’єктів типу SnappedPointResponse. Ці об’єкти за допомогою Extension методу перетворюються у об’єкти типу SnappedPointWithElevationDb (записавши у поле trackId id запису подорожі, що ми відкриваємо на сторінці «Список записів») та записуються у таблицю DecodedSnappedPointsDb мобільної бази даних. Після цього, лише у випадку успішних транзакцій до серверу та подальшого запису даних у БД поле Decoded відповідної подорожі у БД змінює значення на true та викликається команда відкриття сторінки «Map», куди в якості параметру передається зазначене поле id моделі Track.

5.6 Розробка сторінки відображення збереженої подорожі

Сторінка «Карта» безпосередньо відображає на карті записану подорож користувача у вигляді ламаної, кожний сегмент якої має колір відповідний до миттєвої швидкості користувача на даному відрізці подорожі. Виконана сторінка за допомогою стандартного елементу Xamarin Forms, що поставляється у бібліотеці Xamarin.Forms.Maps.

Використання таких функцій класу Map як додавання на карту зображень кривих є платформозалежним функціоналом. Тому унаслідуємо стандартний клас Map написавши наш новий клас CustomMap, що розміщується у спільному для всіх операційних систем класі. Саме цей клас містить масив кривих, що мають бути відображені на карті та колекцію MapColorsCollection, що містить масив об’єктів

SpeedColorInterval, які визначають відображені на карті елементи SpeedColorBox. А функціонал саме щодо додавання цих ламаних на карту перенесений у відповідальність класу CustomMapRenderer, що є платформозалежним та знаходиться у проекті Droid.

Для відображення швидкості користувача на мапі різними кольорами було обрані кольори, що наведені у табл. 5.6.1.

Таблиця 5.6.1 – Структура моделі Track.

Мінімальна швидкість	Мінімальна швидкість	Назва кольору	Color hex
0	3	Black	000000
3	10	Dark Red	550000
10	30	Red	ff0000
30	50	Orange	ffa500
50	70	Yellow	ffe500
70	90	Green 3	00cd00
90	110	Green	008000
110	130	Deepskyblue	00bfff
130	150	Blue	0000ff
150	170	Purple	8000ff
170	∞	Dark purple	2a0059

Величини у табл. 5.6.1 наведені без розмірностей. Це пояснюється тим, що інтервали швидкості є однаковими при використанні у якості одиниці вимірювання кілометрів на годину чи миль на годину.

Зовнішній вигляд сторінки «Карта» зображено на рис. 5.6.1

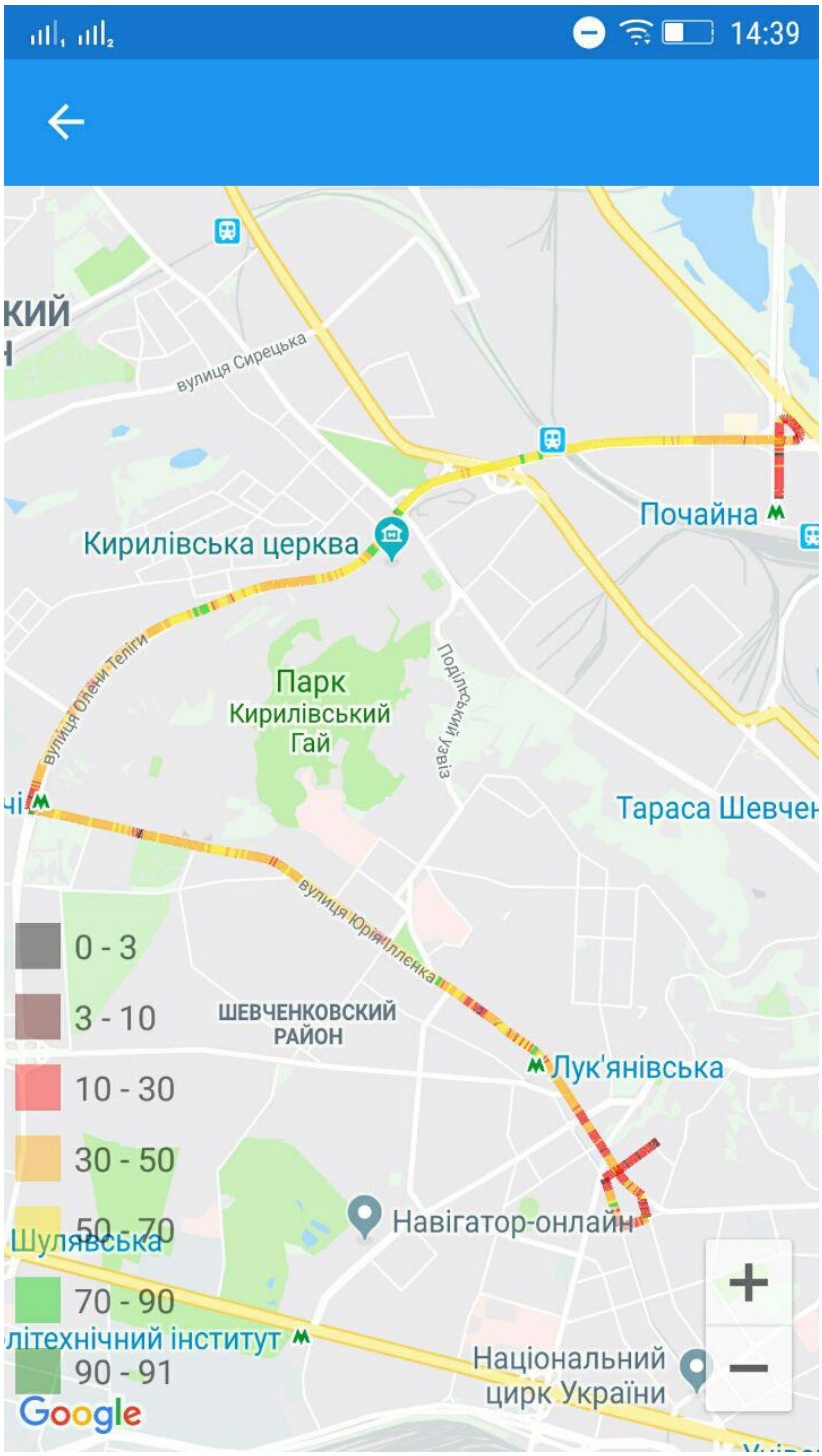


Рис. 5.6.1 Вигляд сторінки «Карта».

При відкритті сторінка «Карта» фокусується саме на географічному розташуванні потрібної подорожі, масштаб вибирається так аби весь маршрут вмістився на користувацькому екрані. Логіка такого фокусування та масштабування зосереджена у методі MoveToRegion класу MapPage.

Висновок

У даному розділі було розроблено мобільний додаток на базі мобільного фреймворку Xamarin. Даний додаток підтримує смартфони на базі операційної системи Android версії 6.0 та вище. Це пов’язане з тим, що функціонал, що дозволяє отримати дані про рівень сигналу базової станції стільникового зв’язку доступна лише в останніх версіях цієї ОС. Одночасно з цим починаючи саме з версії 6.0 ОС накладує обмеження на отримання доступу до різних можливостях платформи і окрім вказівки дозволів у маніфесті додатку необхідно також додатково запросити дозвіл у користувача шляхом спливаючого вікна. Це може негативно вплинути на довіру користувачів до додатку на зменшити потенційну аудиторію.

Проаналізувавши використання розробленого додатку на великій аудиторії користувачів можливо матиме сенс ввести аутентифікацію користувача у мобільний додаток за допомогою профілю користувача смартфону (профіль користувача Google Play у випадку ОС Android), задля введення обмеження кількості викликів веб-додатку одним користувачем впродовж певного періоду.

6. ЕКОНОМІЧНИЙ АНАЛІЗ

Основні витрати на підтримку розробленого додатку випадають на використання платних сервісів компанії Google.

Використання сервісу Google Static Maps API коштує $2.00 за тисячу запитів при кількості запитів менше 100000 на місяць, та $1.60 за тисячу запитів при перевищені даного ліміту.

Використання сервісу Google Roads (Roads - Route Traveled) API коштує $10.00 за тисячу запитів при кількості запитів менше 100000 на місяць, та $8.00 за тисячу запитів при перевищені даного ліміту.

Використання сервісу Google Elevation API коштує $5.00 за тисячу запитів при кількості запитів менше 100000 на місяць, та $4.00 за тисячу запитів при перевищені даного ліміту.

Таким чином, виходячи з вказаних тарифів, плата за використання сервісів Google впродовж одного місяця одним користувачем при збережені в середньому однієї подорожі вдень довжиною 2км склала 17,09 доларів за використання Elevation API, 5,66 доларів за використання Roads API та 2,31 долара за використання Static Maps API.

Співвідношення кількості викликів на кожен з зазначених сервісів Google показано на рис. 6.1. Помаранчевим кольором на даному рисунку позначені виклики Google Roads API (сумарно 96 викликів), зеленим кольором позначені виклики Google Static Maps API (сумарно 56 викликів), синім кольором позначені виклики Google Elevation API (сумарно 35 викликів).

Слід зазначити, що розроблений додаток кешує дані отримані за допомогою Google Static Maps API. Саме тому експериментально визначена середньомісячна вартість використання даного сервісу одним користувачем є статистично неточною, оскільки частина користувачів можуть активно використовувати додаток у різних містах завантажуючи карту кожний раз наново.

Використання технологій Xamarin та .Net Core є безкоштовним, використання Microsoft Visual Studio, що підтримує обидва ці фреймворки коштує 45 доларів на місяць.

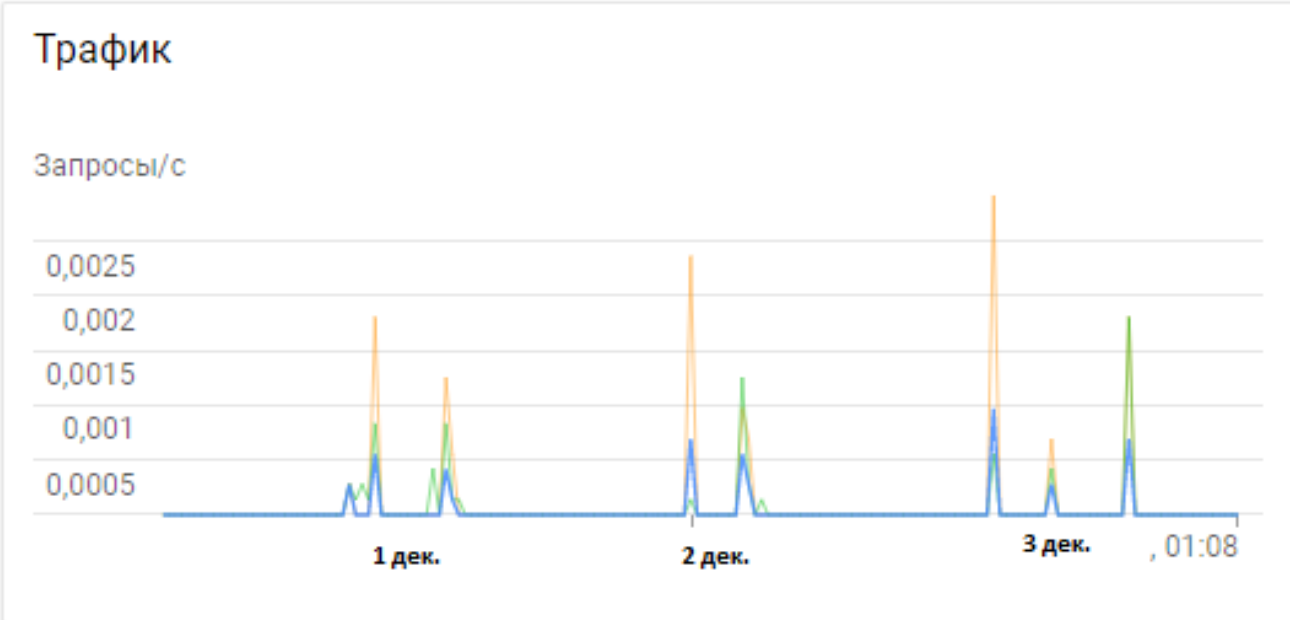


Рис. 6.1 Трафік використання сервісів Google API.

Також слід зазначити вже описані у розділі 4.4 витрати у розмірі 2,55 доларів на місяць на підтримку хостингу для публікації розробленого веб-застосунку.

Висновок

Таким чином, сумарна вартість підтримки розробленого додатку складає 47,55 доларів в місяць незалежно від кількості користувачів та 25,06 долара на кожного активного користувача.

ВИСНОВКИ

В даній роботі вирішувалась задача розробки мобільного додатку для збору даних про якість стільникового покриття.

Були отримані такі результати:

- Були проаналізовані сучасні рішення оцінки якості стільникового зв’язку, визначено ї переваги та недоліки.
- Були проаналізовані сучасні рішення оцінки якості стільникового зв’язку шляхом проведення драйв-тестів.
- Був розроблений веб-додаток для збільшення точності даних про географічне розташування та збереження даних про рівень сигналу базової станції стільникового зв’язку.
- Був розроблений мобільний додаток на базі фреймворку Xamarin для запису та візуалізації даних користувача про його переміщення.
- Був розроблений алгоритм збору даних.

СПИСОК ПОСИЛАНЬ

1. Про встановлення рівнів якості послуг рухомого (мобільного) зв'язку: наказ Міністерства транспорту та зв’язку України від 19 березня 2010 р. № 147
2. QoS [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/QoS.
3. Буранова М. А. Исследование влияния статистических свойств мультимедийного IP-трафика на характеристики качества обслуживания Научная библиотека диссертаций и авторефератов disserCat http://www.dissercat.com/content/issledovanie-vliyaniya-statisticheskikh-svoistv-multi : дис. канд. техн. наук / Буранова Марина Анатольевна.
4. Показатели качества услуг мобильной связи всех операторов Украины за I полугодие 2017 года [Електронний ресурс] – Режим доступу до ресурсу: https://itc.ua/news/nkrsi-opublikovala-pokazateli-kachestva-uslug-mobilnoy-svyazi-vseh-operatorov-ukrainyi-za-i-polugodie-2017-goda/
5. НКРЗІ опрацювання звітів операторів про якість телекомунікаційних послуг за 2017 рік [Електронний ресурс] – Режим доступу до ресурсу: http://spz.nkrzi.gov.ua/golovna/yakist-poslug/dani-pro-yakist/
6. RFBENCHMARK [Електронний ресурс] – Режим доступу до ресурсу: https://www.rfbenchmark.ru
7. API [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/API
8. Xamarin App Development [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/xamarin/
9. Плюсы и минусы разработки на Xamarin / Хабр [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/343098/
10. Паттерн Model-View-ViewModel [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/xamarin/4.2.php/

11. Android by example : MVVM +Data Binding [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@husayn.hakeem/android-by-example-mvvm-data-binding-introduction-part-1-6a7a5f388bf7
12. Плюсы и минусы разработки на Xamarin / Хабр [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/343098/
13. .Net Core Framework [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/
14. ASP.NET Core | Введение в MVC [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/3.1.php/
15. Структура ASP.NET MVC. вызов контроллера из вида [Електронний ресурс] – Режим доступу до ресурсу: http://qaru.site/questions/2372428/aspnet-mvc-structure-calling-a-controller-from-a-view/
16. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/HTTP
17. REST [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/REST/
18. Snap to Roads [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/roads/snap
19. Elevation API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/elevation/start
20. ASP.NET Core | Публикация на IIS [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet5/20.1.php
21. Official Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/
22. Google Cloud Platform | Современные инструменты [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com
23. Microsoft Azure Cloud Computing Platform & Services [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/
24. Хостинг ASP.NET [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/articles/mvc/6.php

25. SmarterASP.NET | Easy Plan Upgrade/Downgrade [Електронний ресурс] – Режим доступу до ресурсу: https://www.smarterasp.net/hosting_plans
26. Artifactory - Universal Artifact Repository Manager - JFrog [Електронний ресурс] – Режим доступу до ресурсу: https://jfrog.com/artifactory/
27. SQLite Home Page [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlite.org/index.html
28. Stackoverflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/7591492/is-it-possible-to-apply-primary-key-on-the-text-fields-in-android-database
29. Bring Stunning Animations to Your Apps with Lottie | Xamarin Blog [Електронний ресурс] – Режим доступу до ресурсу: https://blog.xamarin.com/bring-stunning-animations-to-your-apps-with-lottie/

ДОДАТОК А ЛІСТИНГ СЕРВЕРНОГО ЗАСТОСУНКУ

```
using System;
using System.Net;
using System.Net.Http;
using Microsoft.AspNetCore.Mvc;
using SpeedServerApi.Models;
using Speed_Server.Models;

namespace Speed_Server.Controllers
{
    [Produces("application/json")]
    [Route("api/SpeedServer")]
    public class SpeedServerController : Controller
    {
        private readonly SpeedServerContext _context;
        private static HttpClient client = new HttpClient();
        private readonly GoogleRoadsApi _googleRoadsApi;
        private readonly GoogleEvaluationApi _googleEvaluationApi;
        private readonly IDbController _dbController;
        public SpeedServerController(SpeedServerContext context)
        {
            _googleRoadsApi = new GoogleRoadsApi();
            _googleEvaluationApi = new GoogleEvaluationApi();
            _dbController = new MsSqlController();
        }

        [HttpPost]
        public IActionResult Create([FromBody] SnappedPointRequest[] snappedPointsRequests)
        {
            if (snappedPointsRequests == null)
            {
                return BadRequest("Track is empty");
            }
            SpeedModel speedModel = new SpeedModel(snappedPointsRequests);
            SpeedModel speedModelWithRoadsAndEvaluations;

            try
            {
                SpeedModel speedModelWithRoads = _googleRoadsApi.FillSpeedModel(speedModel,
false);
                speedModelWithRoadsAndEvaluations =
_googleEvaluationApi.FillSpeedModel(speedModelWithRoads);
            }
            catch (Exception e)
            {
                return BadRequest("Remote server error, please try again later.");
            }
            return new ObjectResult(speedModelWithRoadsAndEvaluations);
            try
            {
                _dbController.SaveSnappedPointRequestsToDb(snappedPointsRequests);
            }
            catch (Exception e)
            {
            }
        }
    }
}

using System.IO;
```

```
using System.Linq;
using System.Net;
using Speed_Server.Models;

namespace Speed_Server.Controllers
{
    public abstract class GoogleApi
    {
        protected int limitPointPerQuery { get; set; }
        protected string GoogleApiKey { get; set; }
        protected string urlApi { get; set; }

        protected string ExecuteQuery(string urlRequest)
        {
            WebRequest request = WebRequest.Create(urlRequest);
            WebResponse response = request.GetResponse();
            Stream data = response.GetResponseStream();
            StreamReader reader = new StreamReader(data);

            string responseFromServer = reader.ReadToEnd();

            response.Close();

            return responseFromServer;
        }

        protected IGrouping<int, LocationTime>[] GroupLocationTimeArrayByQuery(LocationTime[]
locations)
        {
            var groupedLocationByQuery = locations.Select((i, index) => new
            {
                i,
                index
            }).GroupBy(grouped => grouped.index / limitPointPerQuery, location =>
location.i).ToArray();
            return groupedLocationByQuery;
        }

        protected LocationTime[] PullLocationTimeArrayFromSpeedModel(SpeedModel speedModel)
        {
            LocationTime[] locationTimeArray = speedModel.snappedPoints.Select(x => new
LocationTime(x.Location, x.time)).ToArray();
            return locationTimeArray;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using Newtonsoft.Json;
using Speed_Server.Extensions;
using Speed_Server.Models;

namespace Speed_Server.Controllers
{
    public class GoogleEvaluationApi : GoogleApi
    {
        public GoogleEvaluationApi()
        {
            limitPointPerQuery = 400;
            GoogleApiKey = "GOOGLE_API_KEY ";
            urlApi =
"https://maps.googleapis.com/maps/api/elevation/json?locations={0}&key={1}";
        }
```

```
        public GoogleEvaluationApi(int limitPointPerQuery)
        {
            this.limitPointPerQuery = limitPointPerQuery;
            GoogleApiKey = "AIzaSyC2FZl6lpRYmLqLPB6py-fd_Q9Q6C6AIiQ";
            urlApi =
"https://maps.googleapis.com/maps/api/elevation/json?locations={0}&key={1}";
        }

        public SpeedModel FillSpeedModel(SpeedModel speedModel)
        {
            LocationTime[] locationTimeArray =
PullLocationTimeArrayFromSpeedModel(speedModel);
            locationTimeArray.ForEach(i=>i.RoundCoordinats(5));

            var groupedLocationTimeByRequest =
GroupLocationTimeArrayByQuery(locationTimeArray);

            List<SpeedModel> speedModelList = ExecuteAllRequest(groupedLocationTimeByRequest);

            var elevationSpeedModel = new SpeedModel(speedModelList);
            var compliteSpeedModel = new SpeedModel(speedModel, elevationSpeedModel);

            return compliteSpeedModel;
        }

        private string MadeUrlRequest(IGrouping<int, Location> locations)
        {
            string locationsString = String.Join("|", locations);
            string urlRequest = String.Format(urlApi, locationsString, GoogleApiKey);
            return urlRequest;
        }

        private List<SpeedModel> ExecuteAllRequest(IGrouping<int, LocationTime>[]
groupedLocationTimeByRequest)
        {
            List<SpeedModel> speedModelList = new List<SpeedModel>();

            for (var i = 0; i < groupedLocationTimeByRequest.Length; i++)
            {
                var urlRequest = MadeUrlRequest(groupedLocationTimeByRequest[i]);
                var responseFromServer = ExecuteQuery(urlRequest);
                var responseFromServerModelFrom = responseFromServer.Replace("lat",
"latitude").Replace("lng", "longitude");
                GoogleElevationResponse googleElevationResponse =
JsonConvert.DeserializeObject<GoogleElevationResponse>(responseFromServerModelFrom);
                var speedModel = new SpeedModel(googleElevationResponse.results);
                speedModelList.Add(speedModel);
            }
            return speedModelList;
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using Newtonsoft.Json;
using Speed_Server.Models;

namespace Speed_Server.Controllers
{
    public class GoogleRoadsApi : GoogleApi
```

```
    {
        public GoogleRoadsApi()
        {
            limitPointPerQuery = 100;
            GoogleApiKey = "GOOGLE_API_KEY";
            urlApi =
"https://roads.googleapis.com/v1/snapToRoads?path={0}&interpolate={1}&key={2}";
        }

        public GoogleRoadsApi(int limitPointPerQuery)
        {
            this.limitPointPerQuery = limitPointPerQuery;
            GoogleApiKey = " GOOGLE_API_KEY ";
            urlApi =
"https://roads.googleapis.com/v1/snapToRoads?path={0}&interpolate={1}&key={2}";
        }

        public SpeedModel FillSpeedModel(SpeedModel speedModel, bool interpolate)
        {
            LocationTime[] locationTimeArray =
PullLocationTimeArrayFromSpeedModel(speedModel);

            var groupedLocationTimeByRequest =
GroupLocationTimeArrayByQuery(locationTimeArray);

            List<SpeedModel> speedModelList = ExecuteAllRequest(groupedLocationTimeByRequest,
interpolate);

            var compliteSpeedModel = new SpeedModel(speedModelList);

            return compliteSpeedModel;
        }

        private string MadeUrlRequest(IGrouping<int, Location> locations, bool interpolate)
        {
            string locationsString = String.Join("|", locations);
            string urlRequest = String.Format(urlApi, locationsString, interpolate,
GoogleApiKey);
            return urlRequest;
        }

        private List<SpeedModel> ExecuteAllRequest(IGrouping<int, LocationTime>[]
groupedLocationTimeByRequest, bool interpolate)
        {
            List<SpeedModel> speedModelList = new List<SpeedModel>();

            for (var i = 0; i < groupedLocationTimeByRequest.Length; i++)
            {
                var urlRequest = MadeUrlRequest(groupedLocationTimeByRequest[i], interpolate);
                var responseFromServer = ExecuteQuery(urlRequest);
                GoogleRoadResponse googleRoadResponse =
JsonConvert.DeserializeObject<GoogleRoadResponse>(responseFromServer);

                SpeedModel speedModel = new SpeedModel(googleRoadResponse.snappedPoints);

                int previousOriginalElementIndex = 0;

                var groupOfTimeArray = groupedLocationTimeByRequest[i].ToArray();
                speedModel.snappedPoints[0].time = groupOfTimeArray[0].time;

                for (var j = 1; j < speedModel.snappedPoints.Length; j++)
                {
                    if (speedModel.snappedPoints[j].originalIndex != 0)
                    {
```

```
                        var indexDifferenceBetweenOriginalElements = j -
previousOriginalElementIndex;
                        var timeDifferenceBetweenOriginalElements =
                            groupOfTimeArray[speedModel.snappedPoints[j].originalIndex].time -
groupOfTimeArray[speedModel.snappedPoints[previousOriginalElementIndex].originalIndex].time;
                        var timeDifferenceBetweenNeighborElements =
timeDifferenceBetweenOriginalElements.TotalMilliseconds /
indexDifferenceBetweenOriginalElements;
                        var nextOriginalElementTime =
groupOfTimeArray[speedModel.snappedPoints[j].originalIndex].time;
                        int schetchik = 0;

                        for (var k = j; k > previousOriginalElementIndex; k--)
                        {
                            speedModel.snappedPoints[k].time =
                                nextOriginalElementTime.AddMilliseconds(-1 * schetchik *
timeDifferenceBetweenNeighborElements);
                            schetchik = schetchik + 1;
                        }
                        previousOriginalElementIndex = j;
                    }
                }
                speedModelList.Add(speedModel);
            }
            return speedModelList;
        }
    }
}


namespace Speed_Server.Models
{
    public class GoogleElevationResponse
    {
        public SnappedPointElevation[] results { get; set; }
    }
}

using System;
using System.Collections.Generic;

namespace Speed_Server.Models
{
    public class SpeedModel
    {
        public SnappedPointResponse[] snappedPoints { get; set; }

        public SpeedModel()
        {
        }

        public SpeedModel(SpeedModel standartModel, SpeedModel additionModle)
        {
            List<SnappedPointResponse> listPoints=new List<SnappedPointResponse>();

            var standartSnappedPoints = standartModel.snappedPoints;
            var additionSnappedPoints = additionModle.snappedPoints;

            if (standartSnappedPoints.Length != additionSnappedPoints.Length)
            {
                throw new Exception("разная длинна дополняемых моделей");
            }

            for (var i=0;i<standartSnappedPoints.Length;i++)
```

```
            {
                var standartPoint = standartSnappedPoints[i];
                var additionPoint = additionSnappedPoints[i];

                var newPoint = new SnappedPointResponse();
                var newLocation = new LocationWithElevation();
                // долгота
                if (standartPoint.Location.longitude != additionPoint.Location.longitude)
                {
                    if ((standartPoint.Location.longitude != 0) &&
(additionPoint.Location.longitude == 0))
                    {
                        newLocation.longitude = standartPoint.Location.longitude;
                    }
                    else if ((standartPoint.Location.longitude == 0) &&
(additionPoint.Location.longitude != 0))
                    {
                        newLocation.longitude = additionPoint.Location.longitude;
                    }
                    else
                    {
                        newLocation.longitude = additionPoint.Location.longitude;
                    }
                }
                else
                {
                    newLocation.longitude = standartPoint.Location.longitude;
                }
                // широта
                if (standartPoint.Location.latitude != additionPoint.Location.latitude)
                {
                    if ((standartPoint.Location.latitude != 0) &&
(additionPoint.Location.latitude == 0))
                    {
                        newLocation.latitude = standartPoint.Location.latitude;
                    }
                    else if ((standartPoint.Location.latitude == 0) &&
(additionPoint.Location.latitude != 0))
                    {
                        newLocation.latitude = additionPoint.Location.latitude;
                    }
                    else
                    {
                        newLocation.latitude = additionPoint.Location.latitude;
                    }
                }
                else
                {
                    newLocation.latitude = standartPoint.Location.latitude;
                }
                // высота

                if (standartPoint.Location.elevation != additionPoint.Location.elevation)
                {
                    if ((standartPoint.Location.elevation != 0) &&
(additionPoint.Location.elevation == 0))
                    {
                        newLocation.elevation = standartPoint.Location.elevation;
                    }
                    else if ((standartPoint.Location.elevation == 0) &&
(additionPoint.Location.elevation != 0))
                    {
                        newLocation.elevation = additionPoint.Location.elevation;
                    }
```

```
                    else
                    {
                        throw new Exception("Обе модели не могут иметь заполненное
значение?");
                    }
                }
                else
                {
                    newLocation.elevation = standartPoint.Location.elevation;
                }


                // время

                var nullDateTime = DateTime.Parse("1/1/0001 12:00:00 AM");

                if (standartPoint.time!= additionPoint.time)
                {
                    if ((standartPoint.time != nullDateTime) &&(additionPoint.time==
nullDateTime))
                    {
                        newPoint.time = standartPoint.time;
                    }
                    else if ((standartPoint.time == nullDateTime) && (additionPoint.time !=
nullDateTime))
                    {
                        newPoint.time = additionPoint.time;
                    }
                    else
                    {
                        throw new Exception("Обе модели не могут иметь заполненное время?");
                    }
                }
                else
                {
                    newPoint.time = standartPoint.time;
                }

                //placeID
                if (standartPoint.placeId != additionPoint.placeId)
                {
                    if ((standartPoint.placeId != null) && (additionPoint.placeId == null))
                    {
                        newPoint.placeId = standartPoint.placeId;
                    }
                    else if ((standartPoint.placeId == null) && (additionPoint.placeId !=
null))
                    {
                        newPoint.placeId = additionPoint.placeId;
                    }
                    else
                    {
                        throw new Exception("Обе модели не могут иметь заполненное время?");
                    }
                }
                else
                {
                    newPoint.placeId = standartPoint.placeId;
                }

                // назначим точке эту локацию

                newPoint.Location = newLocation;
                //добавим точку в лист
```

```
                listPoints.Add(newPoint);
            }

            this.snappedPoints = listPoints.ToArray();
        }

        public SpeedModel(SnappedPointRequest[] snappedPointsRequests)
        {
            List<SnappedPointResponse> snappedPointResponseList = new
List<SnappedPointResponse>();

            foreach (var snappedPointRequest in snappedPointsRequests)
            {
                snappedPointResponseList.Add(new SnappedPointResponse(snappedPointRequest));
            }

            this.snappedPoints = snappedPointResponseList.ToArray();
        }


        public SpeedModel(SnappedPointRoad[] snappedPointsRoad)
        {
            List<SnappedPointResponse> snappedPointsRoadList = new
List<SnappedPointResponse>();

            foreach (var snappedPointRoad in snappedPointsRoad)
            {
                snappedPointsRoadList.Add(new SnappedPointResponse(snappedPointRoad));
            }

            this.snappedPoints = snappedPointsRoadList.ToArray();
        }

        public SpeedModel(SnappedPointElevation[] snappedPointsElevation)
        {
            List<SnappedPointResponse> snappedPointsRoadList = new
List<SnappedPointResponse>();

            foreach (var snappedPointElevation in snappedPointsElevation)
            {
                snappedPointsRoadList.Add(new SnappedPointResponse(snappedPointElevation));
            }

            this.snappedPoints = snappedPointsRoadList.ToArray();
        }


        public SpeedModel(List<SpeedModel> speedModelList)
        {
            List<SnappedPointResponse> snappedPointsList = new List<SnappedPointResponse>();

            foreach (var speedModel in speedModelList)
            {
                snappedPointsList.AddRange(speedModel.snappedPoints);
            }

            this.snappedPoints = snappedPointsList.ToArray();
        }

        private double DistanceBetweenPoints(Location Point1, Location Point2)
        {
            var Point1Point2Distance = Math.Acos(Math.Sin(Point1.latitude) *
Math.Sin(Point2.latitude) + Math.Cos(Point1.latitude) * Math.Cos(Point2.latitude) *
Math.Cos(Point1.longitude - Point2.longitude));
```

```
            return Point1Point2Distance;
        }
    }
}

namespace Speed_Server.Models
{
    public class SnappedPointRoad
    {
        public Location Location { get; set; }
        public int originalIndex { get; set; }
        public string placeId { get; set; }

        public SnappedPointRoad(Location location)
        {
            this.Location = location;
        }
    }
}



using System;

namespace Speed_Server.Models
{
    public class SnappedPointResponse
    {
        public LocationWithElevation Location { get; set; }
        public DateTime time { get; set; }
        public int originalIndex { get; set; }
        public string placeId { get; set; }

        public SnappedPointResponse()
        {
        }

        public SnappedPointResponse(SnappedPointRequest snappedPointRequest)
        {
            this.Location = new LocationWithElevation(snappedPointRequest.Location);
            this.time = snappedPointRequest.time;
        }

        public SnappedPointResponse(SnappedPointRoad snappedPointRoad)
        {
            this.Location = new LocationWithElevation(snappedPointRoad.Location);
            this.originalIndex = snappedPointRoad.originalIndex;
            this.placeId = snappedPointRoad.placeId;

        }
        public SnappedPointResponse(SnappedPointElevation snappedPointElevation)
        {
            LocationWithElevation locationElevation = new LocationWithElevation{ latitude =
snappedPointElevation.Location.latitude, longitude = snappedPointElevation.Location.longitude
, elevation = snappedPointElevation.Elevation};
            this.Location = locationElevation;
        }
    }
}


using System;

namespace Speed_Server.Models
```

```
{
    public class SnappedPointRequest
    {
        public Location Location { get; set; }
        public DateTime time { get; set; }

        public SnappedPointRequest()
        { }

        public SnappedPointRequest(SnappedPointRequest snappedPointRequest)
        {
            this.Location = snappedPointRequest.Location;
        }
    }
}


namespace Speed_Server.Models
{
    public class SnappedPointElevation
    {
        public Location Location { get; set; }
        public double Elevation { get; set; }
    }
}


namespace Speed_Server.Models
{
    public class LocationWithElevation: Location
    {
        public double elevation { get; set; }

        public LocationWithElevation()
        { }

        public LocationWithElevation(Location location)
        {
            this.latitude = location.latitude;
            this.longitude = location.longitude;
        }
    }
}


using System;

namespace Speed_Server.Models
{
    public class LocationTime: Location
    {
        public DateTime time { get; set; }

        public LocationTime(Location location, DateTime time)
        {
            this.latitude = location.latitude;
            this.longitude = location.longitude;
            this.time = time;
        }
    }
}

using System;
```

```
namespace Speed_Server.Models
{
    public class Location
    {
        public double latitude { get; set; }
        public double longitude { get; set; }

        public override string ToString()
        {
            string str = latitude + "," + longitude;
            return str;
        }

        public void RoundCoordinats(int decimals)
        {
            latitude = Math.Round(latitude, decimals);
            longitude = Math.Round(longitude, decimals);
        }
    }
}

using System;
using System.Collections.Generic;

namespace Speed_Server.Extensions
{
    public static class LinqExtensions
    {
        public static void ForEach<T>(this IEnumerable<T> enumeration, Action<T> action)
        {
            foreach (T item in enumeration)
            {
                action(item);
            }
        }
    }
}
```

ДОДАТОК В ЛІСТИНГ МОБІЛЬНОГО ДОДАТКУ

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
                        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                        xmlns:forms="clr-namespace:Lottie.Forms;assembly=Lottie.Forms"
                        x:Class="Log.Pages.MainPage"
                NavigationPage.HasNavigationBar="false">
        <StackLayout>
                <Grid VerticalOptions="FillAndExpand" RowSpacing="0">
                        <Grid.RowDefinitions>
                                <RowDefinition Height="1*" />
                                <RowDefinition Height="1*" />
                                <RowDefinition Height="1*" />
                                <RowDefinition Height="1*" />
                        </Grid.RowDefinitions>
                        <StackLayout x:Name="StackLayoutStartRecord" Grid.Row="0"
Orientation="Horizontal" BackgroundColor="DeepSkyBlue">
                                <forms:AnimationView
                                        x:Name="AnimationViewLocation"
                                        Animation="location.json"
                                        AutoPlay="True"
                                        Loop="True"
                                        WidthRequest = "100"
                                        HeightRequest = "100"
                                        VerticalOptions="Center"
                                        OnClick ="StackLayoutStartRecord_Clicked"/>
                                <Label Text="Start Record" TextColor="White"
VerticalOptions="Center" FontSize="Large" FontAttributes="Bold"/>
                        </StackLayout>
                        <StackLayout x:Name="StackLayoutOpenRecordslistPage" Grid.Row="1"
Orientation="Horizontal" BackgroundColor="DodgerBlue">
                                <forms:AnimationView
                                        x:Name="AnimationViewUdatingMap"
                                        Animation="updating_map(blue-red).json"
                                        AutoPlay="True"
                                        Loop="True"
                                        WidthRequest = "100"
                                        HeightRequest = "100"
                                        VerticalOptions="Center"
                                        OnClick ="StackLayoutOpenRecordslistPage_Clicked"/>
                                <Label Text="Open Records" TextColor="White"
HorizontalOptions="Center" VerticalOptions="Center" FontSize="Large" FontAttributes="Bold"/>
                        </StackLayout>
                        <StackLayout x:Name="StackLayoutStartSettings" Grid.Row="2"
Orientation="Horizontal" BackgroundColor="RoyalBlue">
                                <forms:AnimationView
                                        x:Name="AnimationView"
                                        Animation="gears.json"
                                        AutoPlay="True"
                                        Loop="True"
                                        WidthRequest = "100"
                                        HeightRequest = "100"
                                        VerticalOptions="Center"
                                        OnClick ="StackLayoutStartSettings_Clicked"/>
                                <Label Text="Settings" TextColor="White" HorizontalOptions="Center"
VerticalOptions="Center" FontSize="Large" FontAttributes="Bold"/>
                        </StackLayout>
                        <StackLayout x:Name="StackLayoutExit" Grid.Row="3"
Orientation="Horizontal" BackgroundColor="Blue">
                                <forms:AnimationView
                                        x:Name="AnimationViewExit"
```

```
                                    Animation="power(red).json"
                                    AutoPlay="True"
                                    Loop="True"
                                    WidthRequest = "100"
                                    HeightRequest = "100"
                                    VerticalOptions="Center"
                                    OnClick ="StackLayoutExit_Clicked"/>
                            <Label Text="Exit" TextColor="White" HorizontalOptions="Center"
VerticalOptions="Center" FontSize="Large" FontAttributes="Bold"/>
                        </StackLayout>
                </Grid >
        </StackLayout>
</ContentPage>


using System;
using System.Linq;
using Log.DependenciesOS;
using Log.Views;
using Xamarin.Forms;

namespace Log.Pages
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();

            AddGestureRecognizer(StackLayoutStartRecord, StackLayoutStartRecord_Clicked);
            AddGestureRecognizer(StackLayoutOpenRecordslistPage,
StackLayoutOpenRecordslistPage_Clicked);
            AddGestureRecognizer(StackLayoutStartSettings, StackLayoutStartSettings_Clicked);
            AddGestureRecognizer(StackLayoutExit, StackLayoutExit_Clicked);
        }

        private void AddGestureRecognizer(StackLayout stackLayout, Action action)
        {
            stackLayout.GestureRecognizers.Add(
                new TapGestureRecognizer()
                {
                    Command = new Command(action)
                });
        }

        private async void StackLayoutStartRecord_Clicked(object sender, EventArgs e)
        {
            StackLayoutStartRecord_Clicked();
        }

        private async void StackLayoutOpenRecordslistPage_Clicked(object sender, EventArgs e)
        {
            StackLayoutOpenRecordslistPage_Clicked();

        }

        private async void StackLayoutStartSettings_Clicked(object sender, EventArgs e)
        {
            StackLayoutStartSettings_Clicked();

        }

        private async void StackLayoutExit_Clicked(object sender, EventArgs e)
        {
```

```
            StackLayoutExit_Clicked();

        }

        private async void StackLayoutStartRecord_Clicked()
        {
            var allPermissions = App.PermissionsStorage.PermissionsLocation.ToList();
            allPermissions.AddRange(App.PermissionsStorage.PermissionsPhone.ToList());
            var allPermissionsArray = allPermissions.ToArray();

            if (!App.PermissionsResolver.IsAllPermissionsChecked(allPermissionsArray))
            {
                App.PermissionsResolver.SetPermissions(allPermissionsArray, true);
                await DisplayAlert("Allow Permissions", "Before starting recording you must
allow GPS and Phone Permissions! Application collect data about cell signal quality just to
improve the quality of service.", "OK");
            }
            else
            {
                await Navigation.PushAsync(new RecordPage(), true);
            }
        }

        private async void StackLayoutOpenRecordslistPage_Clicked()
        {
            await Navigation.PushAsync(new TracksListPage(), true);
        }

        private async void StackLayoutStartSettings_Clicked()
        {
            // TODO:
        }

        private async void StackLayoutExit_Clicked()
        {
            DependencyService.Get<ICloseApplication>().TerminateApplication();
        }
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:log="clr-namespace:Log;assembly=Log"
             xmlns:elements="clr-namespace:Log.Elements;assembly=Log"
             x:Class="Log.Pages.MapPage">
       <ContentPage.Content>
              <Grid>
                     <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="*" />
                     </Grid.ColumnDefinitions>
                     <Grid.RowDefinitions>
                            <RowDefinition Height="*" />
                     </Grid.RowDefinitions>
                     <StackLayout Grid.Row="0" Grid.Column="0">
                            <elements:CustomMap x:Name="customMap" MapType="Street" />
                     </StackLayout>
                     <StackLayout x:Name="SpeedColorBoxesLayout" Grid.Row="0" Grid.Column="0"
HorizontalOptions="Start"  VerticalOptions="End" Margin ="0,0,0,20">
                     </StackLayout >
              </Grid>
       </ContentPage.Content>
</ContentPage>
```

```
using System;
using System.Collections.Generic;
using System.Device.Location;
using System.Linq;
using Log.Extensions;
using Log.Models;
using Xamarin.Forms;
using Xamarin.Forms.Maps;

namespace Log.Pages
{
    public partial class MapPage : ContentPage
    {
        private double borderCoeficient = 0.8;
        private List<SnappedPointWithElevation> _snappedPointsWithElevationList;

        public MapPage(List<SnappedPointWithElevation> snappedPointsList)
        {
            InitializeComponent();
            _snappedPointsWithElevationList = snappedPointsList;

            customMap.PolylineSegmentList =
_snappedPointsWithElevationList.ToPolylineSegmentList();
            customMap.mapColorsCollection = new MapColorsCollection();

            DrawSpeedColorBoxesLayout(10, 65);
            SizeChanged += MoveToRegion;
        }

        private void DrawSpeedColorBoxesLayout(double minSpeed, double maxSpeed)
        {
            var speeds = customMap.PolylineSegmentList.Select(i=>i.SpeedBetweenPoints());
            var speedColorIntervals =
customMap.mapColorsCollection.SpeedColorIntervalsArray.Where(sci => speeds.Any(speed=>speed<=
sci.RightSpeedBorder&&speed>sci.LeftSpeedBorder));

            speedColorIntervals.First().LeftSpeedBorder = Math.Floor(speeds.Min());
            speedColorIntervals.Last().RightSpeedBorder = Math.Ceiling(speeds.Max());

            foreach (var speedColorInterval in speedColorIntervals)
            {
                var speedColorBox = speedColorInterval.ToSpeedColorBox();
                SpeedColorBoxesLayout.Children.Add(speedColorBox);
            }
        }
        private void MoveToRegion(object sender, EventArgs e)
        {
            var latitudeMin = _snappedPointsWithElevationList.Select(i =>
i.Position.Latitude).Min();
            var latitudeMax = _snappedPointsWithElevationList.Select(i =>
i.Position.Latitude).Max();

            var longitudeMin = _snappedPointsWithElevationList.Select(i =>
i.Position.Longitude).Min();
            var longitudeMax = _snappedPointsWithElevationList.Select(i =>
i.Position.Longitude).Max();

            var avgLatitude = (latitudeMin + latitudeMax) / 2;
            var avgLongitude = (longitudeMin + longitudeMax) / 2;

            var trackWidth = new GeoCoordinate(avgLatitude, longitudeMin).GetDistanceTo(new
GeoCoordinate(avgLatitude, longitudeMax));
```

```
            var trackHeight = new GeoCoordinate(latitudeMin, avgLongitude).GetDistanceTo(new
GeoCoordinate(latitudeMax, avgLongitude));

            var trackDimension = trackHeight / trackWidth;
            var pageDimension = Height / Width;

            var centerPosition = new Position(avgLatitude, avgLongitude);

            var diameterMeters =
                pageDimension >= trackDimension
                    ? trackHeight
                    : trackWidth * trackDimension / pageDimension / 2;

            customMap.MoveToRegion(MapSpan.FromCenterAndRadius(centerPosition,
Distance.FromMeters((1 / borderCoeficient) * diameterMeters)));
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Log"
             xmlns:gauge="clr-
namespace:Syncfusion.SfGauge.XForms;assembly=Syncfusion.SfGauge.XForms"
             x:Class="Log.Pages.RecordPage"
             NavigationPage.HasNavigationBar="false">

       <Grid ColumnSpacing="5" RowSpacing="5" Padding="20">
              <Grid.RowDefinitions>
                     <RowDefinition Height="5*" />
                     <RowDefinition Height="1.5*" />
                     <RowDefinition Height="1.5*" />
                     <RowDefinition Height="1*" />
              </Grid.RowDefinitions>
              <Grid.ColumnDefinitions>
                     <ColumnDefinition Width="*" />
                     <ColumnDefinition Width="3*" />
              </Grid.ColumnDefinitions>
              <StackLayout Orientation="Vertical">
                     <Label Text="D" FontSize="50"
                            LineBreakMode="NoWrap"
                            HorizontalOptions="Center"
                            Rotation="-90"
                            VerticalOptions="Center"
                            FontAttributes="Bold"
                            FontFamily=""/>
                     <Label Text="E" FontSize="50"
                            LineBreakMode="NoWrap"
                            HorizontalOptions="Center"
                            Rotation="-90"
                            VerticalOptions="Center"
                            FontAttributes="Bold"/>
                     <Label Text="E" FontSize="50"
                            LineBreakMode="NoWrap"
                            HorizontalOptions="Center"
                            Rotation="-90"
                            VerticalOptions="Center"
                            FontAttributes="Bold"/>
                     <Label Text="P" FontSize="50"
                            LineBreakMode="NoWrap"
                            HorizontalOptions="Center"
```

```
                        Rotation="-90"
                        VerticalOptions="Center"
                        FontAttributes="Bold"/>
                    <Label Text="S" FontSize="50"
                        LineBreakMode="NoWrap"
                        HorizontalOptions="Center"
                        Rotation="-90"
                        VerticalOptions="Center"
                        FontAttributes="Bold"/>
            </StackLayout>
            <Label x:Name="SpeedLabel" FontSize="250" Text="120" Grid.Row="0" Grid.Column="1"
Grid.RowSpan="2"  VerticalOptions="CenterAndExpand" HorizontalOptions="CenterAndExpand">
                    <Label.FontFamily>
                        <OnPlatform x:TypeArguments="x:String">
                            <On Platform="Android" Value="Droid.ttf#Droid-Regular" />
                        </OnPlatform>
                    </Label.FontFamily>
            </Label>
            <Label x:Name="MeasurementUnit" FontSize="30" FontAttributes="Bold"
HorizontalOptions="Center" VerticalOptions="Center" Text="km/h" Grid.Row="1" Grid.Column="0"
/>
            <Label x:Name="DurationTextLabel" FontSize="20"  HorizontalOptions="Start"
VerticalOptions="Center" Text="Duration" Grid.Row="2" Grid.Column="0" />
            <Label x:Name="DurationLabel" HorizontalOptions="Center" VerticalOptions="Center"
FontSize="20" Text="10:00:00" Grid.Row="2" Grid.Column="1" />
            <Button x:Name="buttonStop" Text="stop" Grid.Row="3" Grid.ColumnSpan="2"
Clicked="ButtonStop_Clicked" />

        </Grid>

</ContentPage>
```

```
using System;
using Log.DependenciesOS;
using Log.Locators;
using Log.Models;
using Plugin.Geolocator.Abstractions;
using Xamarin.Forms;

namespace Log.Pages
{
    public partial class RecordPage : ContentPage
    {
        private readonly DateTime _startTimeConst;
        private Track _track = new Track();
        private int _snappedPointsCount = 0;
        // meters
        readonly ILocator _locator;
        private ICellAnalyzer _cellListener;
        private Position _previousPosition = new Position(0, 0);
        public string DurationString;

        public RecordPage()
        {чфьф
            InitializeComponent();

            _locator = new LocatorPluginGeolocator(minimumTime:
TimeSpan.FromMilliseconds(0.5), minimumDistance: 10);
            _locator.StartListening(CrossGeolocator_Current_PositionChanged);

            _cellListener = DependencyService.Get<ICellAnalyzer>();
```

```
            _track.StartDateTime = _startTimeConst;

            _track.Id = SaveTrackToDb(_track);
        }

        private int SaveTrackToDb(Track track)
        {
            return App.Database.SaveItem(track);
        }

        public void CrossGeolocator_Current_PositionChanged(object sender, PositionEventArgs
e)
        {
            DurationString = DateTime.UtcNow.Subtract(_startTimeConst).ToString();
            DurationLabel.Text = DurationString.ToString();

            Device.BeginInvokeOnMainThread(() =>
            {
                var positionGeolocator = e.Position;

                if ((positionGeolocator.Latitude != _previousPosition.Latitude) ||
(positionGeolocator.Longitude != _previousPosition.Longitude))
                {
                    var cellData = _cellListener.GetCellData();

                    var snappedPointDb =
                        new SnappedPointDb
                        {
                            TrackId = _track.Id,
                            Latitude = positionGeolocator.Latitude,
                            Longitude = positionGeolocator.Longitude,
                            Time = positionGeolocator.Timestamp.UtcDateTime,
                            Cid = cellData.Cid,
                            CellSignalStrength = cellData.CellSignalStrength
                        };
                    App.SnappedPointDatabase.SaveItem(snappedPointDb);
                    _snappedPointsCount += 1;
                    _previousPosition = positionGeolocator;
                }
            });
        }

        #region activities

        private async void ButtonStop_Clicked(object sender, EventArgs e)
        {
            await _locator.StopListening(CrossGeolocator_Current_PositionChanged);

            if (_snappedPointsCount > 1)
            {
                _track.EndDateTime = DateTime.UtcNow;
                _track.StatusActive = true;
                App.Database.Update(_track);
                await DisplayAlert("Successful", "Track is saved.", "OK");
            }
            else
            {
                App.Database.DeleteItem(_track.Id);
                App.SnappedPointDatabase.DeleteItemsByTrackId(_track.Id);
                await DisplayAlert("Alert", "Recorded track is too short. Saving is
cancelled.", "OK");
            }
            await Navigation.PopAsync();
```

```
        }

        #endregion
    }
}

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Input;
using Log.Pages;
using Log.Services.Controllers;
using Log.Services.Models;
using Xamarin.Forms;

namespace Log.ViewModels
{
    public class TracksListViewModel : INotifyPropertyChanged
    {
        public ObservableCollection<TrackViewModel> Tracks { get; set; }

        private SpeedServerService _speedServerService = new SpeedServerService();
        public event PropertyChangedEventHandler PropertyChanged;
        private readonly Page _page;

        public ICommand DeleteTrackCommand { protected set; get; }
        public ICommand BackCommand { protected set; get; }
        public ICommand OpenTrackCommand { protected set; get; }
        public INavigation Navigation { get; set; }

        public TracksListViewModel(Page page)
        {
            if (page==null)
                throw new Exception();
            _page = page;
            var tracks = App.Database.GetItems().Where(i => i.StatusActive);
            var trackViewModels = tracks.Select(i => new TrackViewModel(i));
            Tracks = new ObservableCollection<TrackViewModel>(trackViewModels);
            DeleteTrackCommand = new Command(DeleteTrackAsync);
            BackCommand = new Command(Back);
            OpenTrackCommand = new Command(OpenTrackAsync);
        }

        protected void OnPropertyChanged(string propName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));
        }

        private void Back()
        {
            Navigation.PopAsync();
        }

        private async void DeleteTrackAsync(object friendObject)
        {
            var track = friendObject as TrackViewModel;

            var trackId = track.Id;
            var action = await _page.DisplayActionSheet("Are you sure that you wanna delete
Track?", "Cancel", "Delete");
            if (action != "Delete")
                return;
```

```
            App.Database.DeleteItem(trackId);
            App.SnappedPointDatabase.DeleteItemsByTrackId(trackId);
            Tracks.Remove(track);
        }

        private async void OpenTrackAsync(object friendObject)
        {
            var track = friendObject as TrackViewModel;
            var trackId = track.Id;
            SpeedModel speedModel;
            if (!App.Database.GetItem(trackId).Decoded)
            {
                try
                {
                    speedModel = await DecodeTrack(trackId);
                }
                catch (Exception exception)
                {
                    await _page.DisplayAlert("Error", exception.Message, "OK");
                    return;
                }
                var snappedPointsWithElevation = speedModel.snappedPoints.Select(i =>
(i.ToSnappedPointWithElevation()));
                var snappedPointsWithElevationDb = snappedPointsWithElevation.Select(i =>
i.ToSnappedPointsWithElevationDb(trackId));
                App.DecodedSnappedPointsDatabase.SaveItems(snappedPointsWithElevationDb);
                App.Database.SetDecoded(trackId, true);
            }

            var snappedPointsWithElevationFromDb =
App.DecodedSnappedPointsDatabase.GetItemsByTrackId(trackId).OrderBy(i => i.Time);

            if (snappedPointsWithElevationFromDb.Count() < 2)
                await _page.DisplayAlert("Error", "This track is too short, please remove it",
"OK");
            var snappedPointsList = snappedPointsWithElevationFromDb.Select(i =>
i.ToSnappedPointWithElevation()).ToList();

            await Navigation.PushAsync(new MapPage(snappedPointsList), true);
        }

        private async Task<SpeedModel> DecodeTrack(int trackId)
        {
            var snappedPoints = App.SnappedPointDatabase.GetItemsByTrackId(trackId);
            var snappedPointRequestArray = snappedPoints.Select(i =>
i.ToSnappedPointRequest());
            var speedModelDecoded = await
_speedServerService.GetSnappedPointsArrayFromSpeedServer(snappedPointRequestArray);

            return speedModelDecoded;
        }
    }
}

using System;
using System.ComponentModel;
using Log.Extensions;
using Log.Models;

namespace Log.ViewModels
{
    public class TrackViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
```

```
        TracksListViewModel lvm;

        public Track Track { get; private set; }

        public TrackViewModel(Track track)
        {
            Track = track;
        }

        public TracksListViewModel ListViewModel
        {
            get { return lvm; }
            set
            {
                if (lvm != value)
                {
                    lvm = value;
                    OnPropertyChanged("ListViewModel");
                }
            }
        }
        public int Id => Track.Id;

        public string StartDate => Track.StartDateTime.ToShortDateString();
        public string StartTime => Track.StartDateTime.ToLongTimeString();

        public string Duration
        {
            get
            {
                var timeSpan = Track.EndDateTime.Subtract(Track.StartDateTime);
                var str = timeSpan.ToReadableString();
                return str;
            }
        }

        protected void OnPropertyChanged(string propName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Name="TracksListPageName"
             x:Class="Log.Views.TracksListPage" Title="Records List">
       <StackLayout>
             <ListView x:Name="TracksList" ItemsSource="{Binding Tracks}"
HasUnevenRows="True">
                   <ListView.Header>
                         <StackLayout Orientation="Horizontal" Padding="5" BackgroundColor =
"LightSteelBlue">
                               <Grid>
                                     <Grid.ColumnDefinitions >
                                           <ColumnDefinition Width ="1*"/>
                                           <ColumnDefinition Width ="3.5*"/>
                                           <ColumnDefinition Width ="3*"/>
                                           <ColumnDefinition Width ="4.5*"/>
                                     </Grid.ColumnDefinitions >
                                     <Label Text="Id" FontSize="Small" Grid.Column="0"/>
                                     <Label Text="StartDate/StartTime" FontSize="Small"
Grid.Column="1"/>
```

```
                                        <Label Text="Duration" FontSize="Small"
Grid.Column="2"/>
                                        <Label Text="Actions" HorizontalOptions="Center"
FontSize="Small" Grid.Column="3"/>
                                    </Grid >
                                </StackLayout>
                            </ListView.Header>
                            <ListView.ItemTemplate>
                                <DataTemplate>
                                    <ViewCell>
                                        <ViewCell.View>
                                            <StackLayout Orientation="Horizontal"
Padding="5">
                                                <Grid>
                                                    <Grid.ColumnDefinitions >
                                                        <ColumnDefinition Width
="1*"/>
                                                        <ColumnDefinition Width
="3.5*"/>
                                                        <ColumnDefinition Width
="3*"/>
                                                        <ColumnDefinition Width
="2.5*"/>
                                                        <ColumnDefinition Width
="2*"/>
                                                    </Grid.ColumnDefinitions >

                                                    <Label Text="{Binding Id}"
FontSize="Small" Grid.Column="0"/>
                                                    <StackLayout
Orientation="Vertical" Grid.Column="1">
                                                        <Label Text="{Binding
StartDate}" FontSize="Small" />
                                                        <Label Text="{Binding
StartTime}" FontSize="Small" />
                                                    </StackLayout>
                                                    <Label Text="{Binding Duration}"
FontSize="Small"  Grid.Column="2"/>
                                                    <Button Text="Open"
Grid.Column="3" FontSize="Micro"  CommandParameter="{Binding .}" Command="{Binding
Source={x:Reference TracksListPageName}, Path=BindingContext.OpenTrackCommand}"/>
                                                    <Button Text="X" Grid.Column="4"
FontSize="Micro"  CommandParameter="{Binding .}" Command="{Binding Source={x:Reference
TracksListPageName}, Path=BindingContext.DeleteTrackCommand}"/>
                                                </Grid >
                                            </StackLayout>
                                        </ViewCell.View>
                                    </ViewCell>
                                </DataTemplate>
                            </ListView.ItemTemplate>
                        </ListView>
                        <AbsoluteLayout HorizontalOptions="FillAndExpand"
                                        VerticalOptions="FillAndExpand">
                            <ActivityIndicator x:Name="activityIndicator" />
                        </AbsoluteLayout>
                    </StackLayout>
                </ContentPage>

                using Log.ViewModels;
                using Xamarin.Forms;

                namespace Log.Views
                {
                    public partial class TracksListPage : ContentPage
```

```
    {
        public TracksListPage()
        {
            InitializeComponent();
            BindingContext = new TracksListViewModel(this) { Navigation = this.Navigation };
        }
    }
}

using Log.DB;
using Log.DependenciesOS;
using Log.Pages;
using Xamarin.Forms;

namespace Log
{
       public partial class App : Application
       {
        public const string TracksUncode = "tracksUncode.db";
           public const string SnappedPointDatabasePath = "snappedPoints.db";
           public const string DecodedSnappedPointsDatabasePath = "decodedSnappedPoints.db";
           private static TrackRepository _database;
           private static SnappedPointRepository _snappedPointDatabase;
           private static DecodedSnappedPointsDb _decodedSnappedPointsDb;
           private static IPermissionsResolver _permissionsResolver;
           private static IPermissionsStorage _permissionsStorage;

        public static TrackRepository Database
        {
            get
            {
                if (_database == null)
                {
                    _database = new TrackRepository(TracksUncode);
                }
                return _database;
            }
        }

           public static SnappedPointRepository SnappedPointDatabase
        {
               get
               {
                   if (_snappedPointDatabase == null)
                   {
                       _snappedPointDatabase = new
SnappedPointRepository(SnappedPointDatabasePath);
                   }
                   return _snappedPointDatabase;
               }
           }

           public static DecodedSnappedPointsDb DecodedSnappedPointsDatabase
        {
               get
               {
                   if (_decodedSnappedPointsDb == null)
                   {
                       _decodedSnappedPointsDb = new
DecodedSnappedPointsDb(DecodedSnappedPointsDatabasePath);
                   }
                   return _decodedSnappedPointsDb;
               }
           }
```

```
        public static IPermissionsResolver PermissionsResolver
        {
               get
               {
                   if (_permissionsResolver == null)
                   {
                       _permissionsResolver = DependencyService.Get<IPermissionsResolver>();
                   }
                   return _permissionsResolver;
               }
           }

        public static IPermissionsStorage PermissionsStorage
        {
               get
               {
                   if (_permissionsStorage == null)
                   {
                       _permissionsStorage = DependencyService.Get<IPermissionsStorage>();
                   }
                   return _permissionsStorage;
               }
           }

        public App ()
             {
            MainPage = new NavigationPage(new MainPage());
        }
       }
}



using System.Linq;
using Android.Content;
using Android.Telephony;
using Log.DependenciesOS;
using Log.Droid;
using Log.Models;
using Xamarin.Forms;

[assembly: Xamarin.Forms.Dependency(typeof(CellListener))]
namespace Log.Droid
{
    public class CellListener : ICellAnalyzer
    {
        private readonly TelephonyManager _telephonyManager;

        public CellListener()
        {
            _telephonyManager =
(TelephonyManager)Forms.Context.GetSystemService(Context.TelephonyService);
        }

        public CellData GetCellData()
        {
            var currentCellInfo = _telephonyManager.AllCellInfo.First();
            var cellDataEntity = new CellData();

            if (currentCellInfo is CellInfoWcdma)
            {
                var a = (CellInfoWcdma) currentCellInfo;
                cellDataEntity.Cid = a.CellIdentity.Cid;
```

```
                cellDataEntity.CellSignalStrength = a.CellSignalStrength.Dbm;
            }
            else if (currentCellInfo is CellInfoGsm)
            {
                var a = (CellInfoGsm) currentCellInfo;
                cellDataEntity.Cid = a.CellIdentity.Cid;
                cellDataEntity.CellSignalStrength = a.CellSignalStrength.Dbm;
            }

            return cellDataEntity;
        }
    }
}
```

```
using Android.App;
using Log.DependenciesOS;
using Log.Droid;
using Xamarin.Forms;

[assembly: Xamarin.Forms.Dependency(typeof(CloseApplication))]
namespace Log.Droid
{
    public class CloseApplication : ICloseApplication
    {
        public void TerminateApplication()
        {
            var activity = (Activity)Forms.Context;
            activity.FinishAffinity();
        }
    }
}
```

```
using System.Collections.Generic;
using Android.Content;
using Android.Gms.Maps.Model;
using Log.Droid;
using Log.Elements;
using Log.Models;
using Xamarin.Forms;
using Xamarin.Forms.Maps;
using Xamarin.Forms.Maps.Android;

[assembly: ExportRenderer(typeof(CustomMap), typeof(CustomMapRenderer))]
namespace Log.Droid
{
    public class CustomMapRenderer : MapRenderer
    {
        public List<PolylineSegment> PolylineSegmentList { get; set; }
        public MapColorsCollection mapColorsCollection;

        public CustomMapRenderer(Context context) : base(context)
        {
        }

        protected override void
OnElementChanged(Xamarin.Forms.Platform.Android.ElementChangedEventArgs<Map> e)
        {
            base.OnElementChanged(e);
```

```
            if (e.OldElement != null)
            {
                // Unsubscribe
            }

            if (e.NewElement != null)
            {
                var formsMap = (CustomMap)e.NewElement;
                PolylineSegmentList = formsMap.PolylineSegmentList;
                mapColorsCollection = formsMap.mapColorsCollection;
                Control.GetMapAsync(this);
            }
        }

        protected override void OnMapReady(Android.Gms.Maps.GoogleMap map)
        {
            base.OnMapReady(map);
            DrawPolylineSegments(PolylineSegmentList);
        }

        private void DrawPolylineSegments(IEnumerable<PolylineSegment>
polylineSegmentCollection)
        {
            foreach (var polylineSegment in PolylineSegmentList)
            {
                var colorInt =
mapColorsCollection.GetColorForSpeed(polylineSegment.SpeedBetweenPoints());
                DrawPolylineSegment(polylineSegment, colorInt);
            }
        }

        private void DrawPolylineSegment(PolylineSegment polylineSegment, int colorInt)
        {
            var polylineOptions = new PolylineOptions();
            polylineOptions.InvokeColor(colorInt);

            polylineOptions.Add(new
LatLng(polylineSegment.SnappedPointStart.Position.Latitude,
polylineSegment.SnappedPointStart.Position.Longitude));
            polylineOptions.Add(new LatLng(polylineSegment.SnappedPointEnd.Position.Latitude,
polylineSegment.SnappedPointEnd.Position.Longitude));

            NativeMap.AddPolyline(polylineOptions);
        }
    }
}
```

```
using System;
using Log.Droid;
using System.IO;
using Xamarin.Forms;
using Log.DependenciesOS;

[assembly: Dependency(typeof(SQLite_Android))]
namespace Log.Droid
{
    public class SQLite_Android : ISQLite
    {
        public SQLite_Android() { }
        public string GetDatabasePath(string sqliteFilename)
```

```
        {
            string documentsPath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentsPath, sqliteFilename);
            return path;
        }
    }
}
```

```
using System.Linq;

namespace Log
{
    public class MapColorsCollection
    {
        // 0x66 + RGB hex
        public static int[] ColorArray = new[]
        {
            0x66000000, // черный               // 0
            0x66550000, // темно-красный        // 1
            0x66FF0000, // Red                  // 2
            0x66FFA500, // Orange               // 3
            0x66ffe500, // желтый               // 4
            0x6600cd00, // салатовый            // 7
            0x66008000, // зеленый              // 6
            0x6600BFFF, // deepskyblue          // 8
            0x660000FF, // синий                // 9
            0x668000FF, // purple               // 11
            0x662a0059  // dark purple          // 12
        };

        public SpeedColorInterval[] SpeedColorIntervalsArray = new[]
        {
            new SpeedColorInterval(0,3,ColorArray[0]),
            new SpeedColorInterval(3,10,ColorArray[1]),
            new SpeedColorInterval(10,30,ColorArray[2]),
            new SpeedColorInterval(30,50,ColorArray[3]),
            new SpeedColorInterval(50,70,ColorArray[4]),
            new SpeedColorInterval(70,90,ColorArray[5]),
            new SpeedColorInterval(90,110,ColorArray[6]),
            new SpeedColorInterval(110,130,ColorArray[7]),
            new SpeedColorInterval(130,150,ColorArray[8]),
            new SpeedColorInterval(150,170,ColorArray[9]),
            new SpeedColorInterval(170,int.MaxValue,ColorArray[10]),
        };

        public int GetColorForSpeed(double speed)
        {
            var color = SpeedColorIntervalsArray.First(i => i.LeftSpeedBorder <= speed &&
i.RightSpeedBorder > speed).SpeedColor;
            return color;
        }
    }
}
```

```
using Log.Elements;
using Xamarin.Forms;

namespace Log
```

```
{
    public class SpeedColorInterval
    {
        public SpeedColorInterval(double leftSpeedBorder, double rightSpeedBorder, int color)
        {
            LeftSpeedBorder = leftSpeedBorder;
            RightSpeedBorder = rightSpeedBorder;
            SpeedColor = color;
        }

        public int SpeedColor;
        public double LeftSpeedBorder;
        public double RightSpeedBorder;

        public SpeedColorBox ToSpeedColorBox()
        {
            var color = Color.FromUint((uint)SpeedColor);

            var speedString = $"{LeftSpeedBorder} - {RightSpeedBorder}";
            var speedColorBox = new SpeedColorBox(color, speedString);
            return speedColorBox;
        }
    }
}
```

```
using System.Collections.Generic;
using Log.Models;
using Xamarin.Forms.Maps;

namespace Log.Elements
{
    public class CustomMap : Map
    {
        public List<PolylineSegment> PolylineSegmentList { get; set; }
        public MapColorsCollection mapColorsCollection;
    }
}
```

```
using Xamarin.Forms;

namespace Log.Elements
{
    public class SpeedColorBox : StackLayout
    {
        public SpeedColorBox(Color color, string text)
        {
            Orientation = StackOrientation.Horizontal;
            Padding = new Thickness (5, 0, 0, 5);

            var colorBox = new BoxView
            {
                Color = color,
                WidthRequest = 20,
                HeightRequest = 20
            };

            var speedLabel = new Label {FontSize = 15, Text = text};

            Children.Add(colorBox);
```

```
            Children.Add(speedLabel);
        }
    }
}
```

```
using Android.Content.PM;

namespace Log.Extensions
{
    public static class PermissionExtension
    {
        //TODO:
        // change to generic to allow work with ios permissions
        // and then rename to 'ToNativePermission'
        public static Permission ToAndroidPermission(this bool value)
        {
            return value ? Permission.Granted : Permission.Denied;
        }
    }
}
```

```
using System;

namespace Log.Extensions
{
    public static class TimeSpanExtension
    {
        public static string ToReadableString(this TimeSpan span)
        {
            string formatted = string.Format("{0}{1}{2}",
                span.Duration().Hours > 0 ? string.Format("{0:0}h", span.Hours) :
string.Empty,
                span.Duration().Minutes > 0 ? string.Format("{0:0}m", span.Minutes) :
string.Empty,
                span.Duration().Seconds > 0 ? string.Format("{0:0}s", span.Seconds) :
string.Empty);
            if (string.IsNullOrEmpty(formatted)) formatted = "0 s";
            return formatted;
        }
    }
}
```

```
using System;
using System.Threading.Tasks;
using Plugin.Geolocator;
using Plugin.Geolocator.Abstractions;

namespace Log.Locators
{
    public class LocatorPluginGeolocator : ILocator
    {
        private IGeolocator geolocator;
        private TimeSpan _minimumTime;
        private double _minimumDistance;


        public LocatorPluginGeolocator(TimeSpan minimumTime, double minimumDistance)
        {
            _minimumTime = minimumTime;
            _minimumDistance = minimumDistance;
```

```
        }

        public async Task StartListening(EventHandler<PositionEventArgs> eventMethod)
        {
            if (CrossGeolocator.Current.IsListening)
                return;
            await CrossGeolocator.Current.StartListeningAsync(_minimumTime, _minimumDistance,
false);

            CrossGeolocator.Current.PositionChanged += eventMethod;
        }

        public async Task StopListening(EventHandler<PositionEventArgs> eventMethod)
        {
            if (!CrossGeolocator.Current.IsListening)
                return;
            await CrossGeolocator.Current.StopListeningAsync();
            CrossGeolocator.Current.PositionChanged -= eventMethod;
        }
    }

}



namespace Log.Models
{
    public class CellData
    {
        public int Cid;
        public int CellSignalStrength;
    }
}


using System.Device.Location;

namespace Log.Models
{
    public class PolylineSegment
    {
        public SnappedPointWithElevation SnappedPointStart { get; set; }
        public SnappedPointWithElevation SnappedPointEnd { get; set; }
        public PolylineSegment(SnappedPointWithElevation snappedPointStart,
SnappedPointWithElevation snappedPointEnd)
        {
            SnappedPointStart = snappedPointStart;
            SnappedPointEnd = snappedPointEnd;
        }

        public double SpeedBetweenPoints()
        {
            var geoCoordinateStart = new GeoCoordinate(SnappedPointStart.Position.Latitude,
SnappedPointStart.Position.Longitude);
            var geoCoordinateEnd = new GeoCoordinate(SnappedPointEnd.Position.Latitude,
SnappedPointEnd.Position.Longitude);

            // km
            var distance = geoCoordinateStart.GetDistanceTo(geoCoordinateEnd) / 1000;
            // h
            var hoursTime = (SnappedPointEnd.Time - SnappedPointStart.Time).TotalHours;
            // km/h
            var speed = distance / hoursTime;
```

```
            return speed;
        }
    }
}
```

```
using System;
using Log.Services.Models;
using SQLite;
using Xamarin.Forms.Maps;

namespace Log.Models
{
    [Table("SnappedPointDb")]
    public class SnappedPointDb
    {
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }
        public int TrackId { get; set; }
        public double Latitude { get; set; }
        public double Longitude { get; set; }
        public DateTime Time { get; set; }
        public int Cid { get; set; }
        public int CellSignalStrength { get; set; }

        public SnappedPointDb()
        {
        }

        public SnappedPointDb(SnappedPoint snappedPoint)
        {
            Latitude = snappedPoint.Position.Latitude;
            Longitude = snappedPoint.Position.Longitude;
            Time = snappedPoint.Time;
        }

        public SnappedPoint ToSnappedPoint()
        {
            return new SnappedPoint(new Position(Latitude, Longitude), Time);
        }

        public SnappedPointRequest ToSnappedPointRequest()
        {
            var location = new Location(Latitude, Longitude);
            var cellData = new CellData{Cid = Cid, CellSignalStrength = CellSignalStrength};
            var snappedPointRequest = new SnappedPointRequest { Location = location, time =
Time, CellData = cellData };
            return snappedPointRequest;
        }
    }
}
```

```
using System;
using SQLite;
using Xamarin.Forms.Maps;

namespace Log.Models
{
    [Table("DecodedSnappedPointsDb")]
    public class SnappedPointWithElevationDb
    {
```

```
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }
        public int TrackId { get; set; }
        public double Latitude { get; set; }
        public double Longitude { get; set; }
        public DateTime Time { get; set; }
        public double Elevation { get; set; }

        public SnappedPointWithElevation ToSnappedPointWithElevation()
        {
            var snappedPointWithElevation = new SnappedPointWithElevation {Position = new
Position(Latitude,Longitude),Time = Time, Elevation = Elevation};
            return snappedPointWithElevation;
        }
    }
}
```

ДОДАТОК С СЛАЙДИ ПРЕЗЕНТАЦІЇ

Тема:Розробка покращеного алгоритму оцінки якості стільникового покриття

Мета:розробка мобільного додатку для збору даних про якість стільникового покриття

Об'єкт дослідження:фреймворк для кросплатферменої мобільної розробки Xamarin

Предмет дослідження:використання фреймворку для кросплатферменої розробки Xamarin для збору даних про якість стільникового покриття

Роботу виконав: Михайлов Сергій Олександрович

Науковий керівник: доцент, ктн Міночкін Дмитро Анатолійович

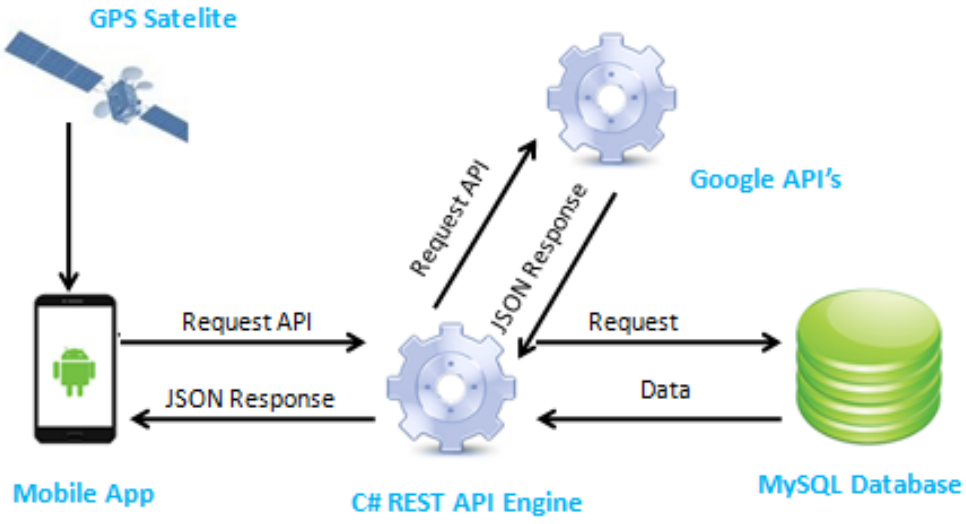
2018

Антени стільникових терміналів, розташованих у вимірювальному комплексі оператору Beeline

Комплекс вимірювання якості стільникового покриття компанії Beeline

ASCOM TEMS Automatic змонтований на базі автомобіля ГРЧЦ

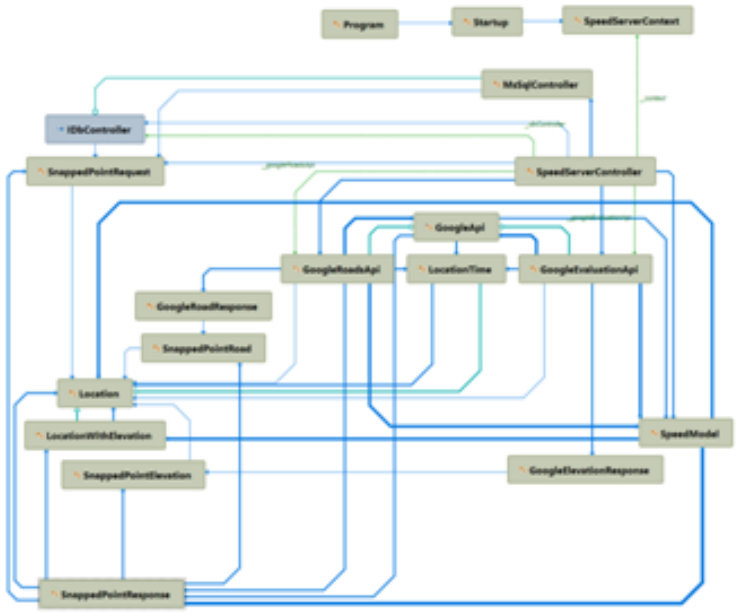
Комплекс вимірювання якості стільникового покриття компанії RFBENCHMARK



Взаємодія між мобільним додатком, серверним застосунком та сторонніми API

		Назва моделі	
Назва поля	Тип поля	SnappedPointRequest	SnapperPointResponse
Location.latitude	double	+	+
Location.longitude	double	+	+
LocationWithElevation.elevation	double		+
time	DateTime	+	+
originalIndex	int		+
placeId	string		+
cellData	string	+	

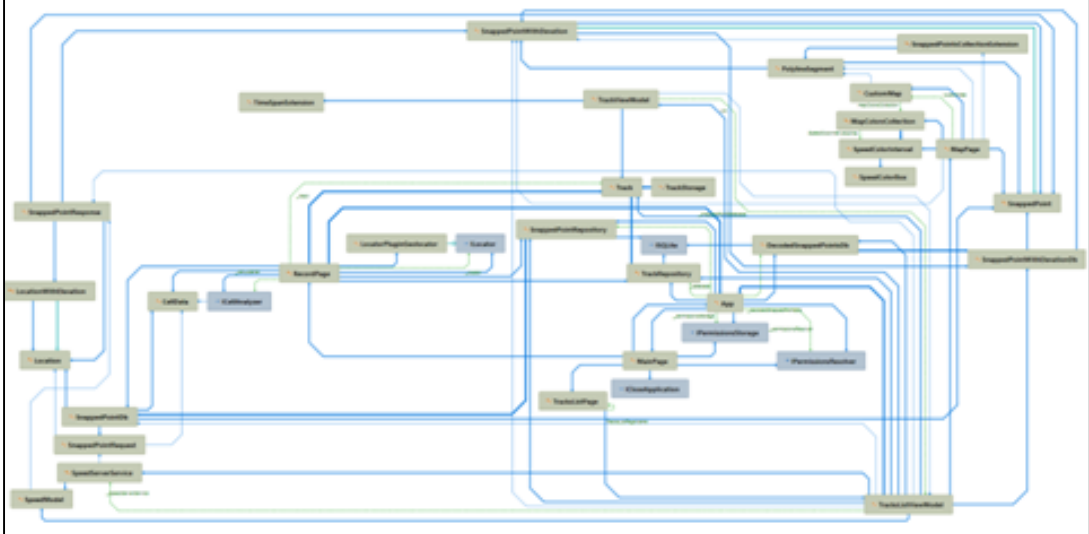
Порівняння моделей SnappedPointRequest та SnapperPointResponse



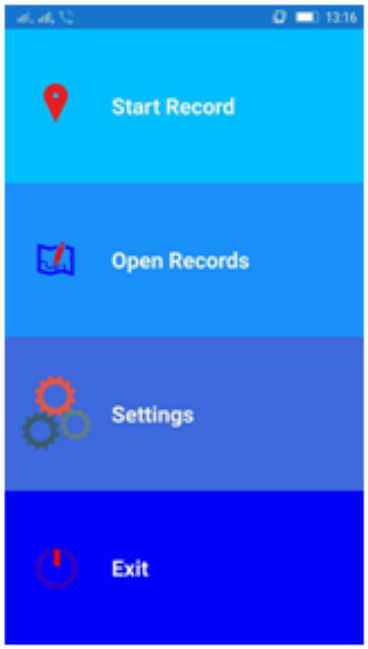
Type Dependencies діаграма серверного застосунку

		Назва моделі	
Назва поля	Тип поля	SnappedPointDb	SnappedPointWithElevationDb
Id	int	+	+
TrackId	int	+	+
Latitude	double	+	+
Longitude	double	+	+
Elevation	double		+
Time	DateTime	+	+
Cid	int	+	
CellSignalStrength	string	+	

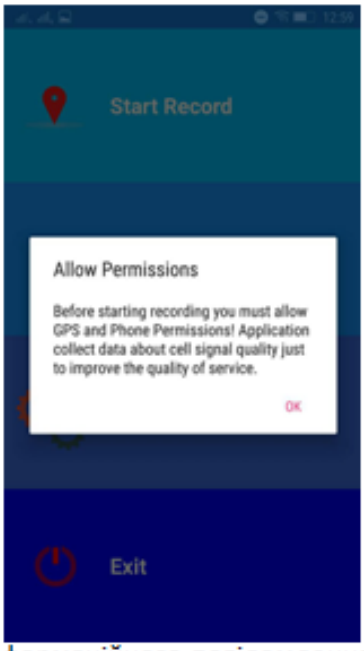
Порівняння моделей SnappedPointDb та SnappedPointWithElevationDb



Діаграма залежностей класів мобільного додатку



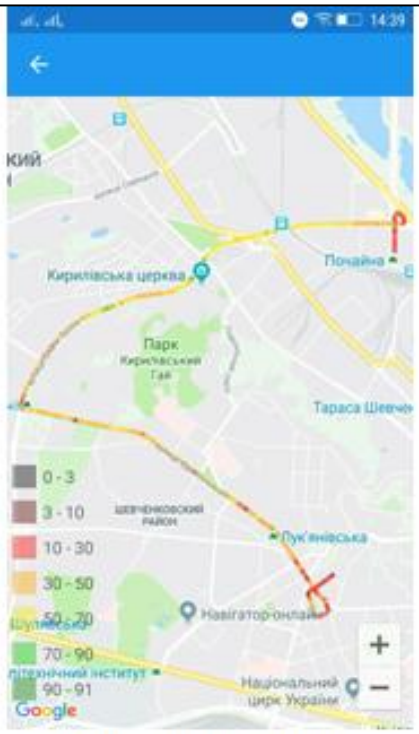
Вигляд головного меню розроблюваного додатку



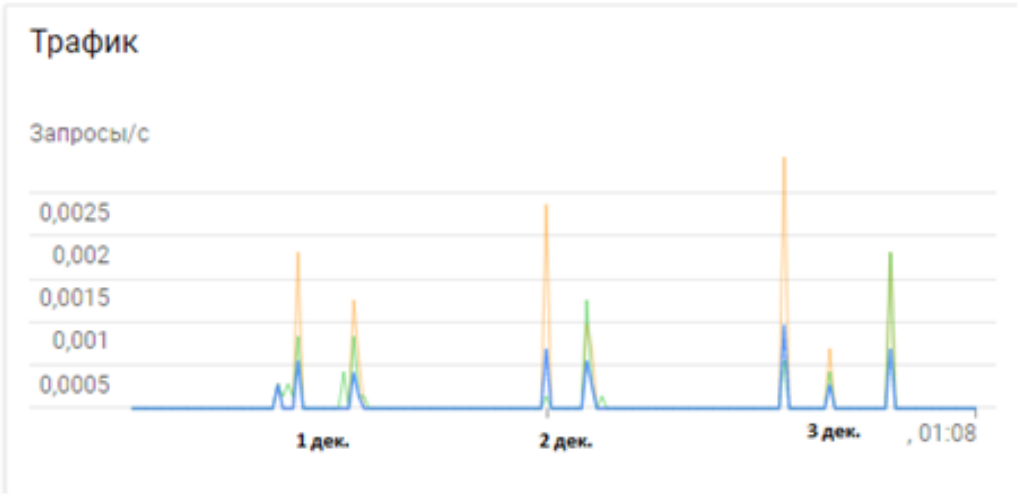
Вигляд інформаційного повідомлення про необхідність надати додатку запитувані дозволи

Вигляд сторінки «Запис» розроблюваного додатку

Вигляд сторінки «Список записів»



Вигляд сторінки «Карта»



Трафік використання сервісів Google API

Дякую за увагу!