КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ВЕЛОСИПЕДИСТІВ З GPS-ТРЕКЕРОМ

Виконав студент 4 курсу
Крук Сергій Васильович

Науковий керівник:
доцент, кандидат технічних наук
Ткаченко Олексій Миколайович

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«01» червня 2022 р., протокол № 10
Завідувач кафедри
М. С. Нікітченко _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 43 сторінки, 20 ілюстрацій, 23 джерел посилання.

ANDROID, GPS, FIREBASE, ВЕЛОСИПЕД, МОБІЛЬНИЙ ДОДАТОК.

Об’єктом дослідження є велосипедні поїздки, предметом дослідження є мобільний Android додаток «Cycler» для велосипедистів.

Мета роботи: проаналізувати основні способи інтеракції користувачів у спортивних додатках для велосипедистів та покращити вже існуючий мобільний додаток, розроблений під час курсової роботи, а саме: покращити інтерфейс та GPS-трекер, додати авторизацію користувачів, забезпечити збереження даних на сервері та розширити функціонал додатку задля інтеракції користувачів між собою – надати велосипедистам можливості організовувати групові поїздки та публікувати їх.

Методи розробки: розробка програмного продукту на основі аналізу існуючих засобів для трекінгу велосипедних поїздок, комп’ютерне моделювання. Інструменти розробки: мови програмування Java, Java Script, XML markup language.

Результати кваліфікаційної роботи: Проаналізовано основні способи інтеракції користувачів у спортивних додатках для велосипедистів, покращено вже існуючий мобільний застосунок для велосипедистів.

Сфера застосування: кінцевий мобільний додаток може використовуватись як продукт у галузі охорони здоров’я.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

XML - Extensible Markup Language

API – Application Programming Interface

GPS - Global Positioning System

Wi-Fi – Wireless Fidality

БД – База Даних

URL - Uniform Resource Locator

URI - Uniform Resource Identifier

SQL - Structured Query Language

SDK – Software Development Kit

ПДР – Правила Дорожнього Руху

JSON - JavaScript Object Notation

ВСТУП

Актуальність роботи та підстави для її виконання. Разом з еко-трендами і приростом популярності велосипедів, зростає і попит на спортивні додатки для велосипедистів, зокрема на ті, що забезпечили б користувачам організовувати групові велосипедні поїздки або приєднуватись до вже існуючих.

Спільні велосипедні поїздки об’єднують людей і усувають соціально-економічні відмінності. Це дає платформу для спілкування з однодумцями та дозволяє новачкам отримати переваги від своїх досвідчених колег. Групова їзда на велосипеді має багато переваг над соло поїздками:

а) Більшість міст світу не мають спеціальної велосипедної інфраструктури. Таким чином, ми в кінцевому підсумку ділимо дорогу з моторизованими транспортними засобами. Кожен водій автомобіля може бути недостатньо ввічливим, щоб поступитися дорогою велосипедисту, але якщо ми в цифрах, у нього немає іншого вибору, окрім як пригальмувати та зупинитися. Таким чином, велогрупа може тонким чином претендувати на право на простір та бути в безпеці, коли їх багато.

б) Катання з людьми відкриває можливість поділитися легким моментом зі своїми колегами. Сміх — найкращі ліки, тому, коли ви ділитеся хвилиною сміху з друзями, це піднімає настрій і вивільняє хороші гормони в організмі. Посмішка допоможе вам пройти ще багато миль.

в) Слово команда означає «разом кожен досягає більшого». Мета стає легшою, якщо її досягати разом. Коли у вас є завдання подолати певну дистанцію, ви можете допомагати один одному підтримувати певний темп і рухатися з інерцією. Велосипедисти можуть мотивувати один одного робити все можливе та показувати новаторські результати.

Мета й завдання роботи. Метою кваліфікаційної роботи є покращення існуючого мобільного додатку для велосипедистів шляхом вдосконалення функціоналу GPS-трекера та реалізації можливості організації групових велосипедних поїздок користувачів. Для досягнення цієї мети було поставлено наступні завдання:

1) Дослідити існуючі програмні рішення для організації спільних групових поїздок
2) Систематизувати та поглибити теоретичні і практичні знання у сфері мобільної розробки
3) Проаналізувати доступні бази даних для розробки мобільних додатків
4) Спроектувати та розробити базу даних для якісного збереження та представлення інформції
5) Вдосконалити GPS-трекер
6) Проаналізувати доступні сервіси для авторизації користувачів мобільних додатків
7) Додати реєстрацію та авторизацію до мобільного додатку для персоналізації користувачів
8) Покращити дизайн та навігацію мобільного додатку «Cycler»

Об’єкт, предмет, методи й засоби розробки. Об’єктом дослідження є групові велосипедні поїздки, предметом дослідження є мобільний додаток «Cycler» що забезпечує можливість їх організації.

Інструменти розроблення: мова програмування Java, Java Script, XML. Для збереження даних було обрано Cloud Firestore та Firestore Storage, для авторизації користувачів – Firebase Authentication, для надсилання сповіщень – Cloud Messaging.

Сфера застосування. Додаток можна використовувати як продукт у сфері охорони здоров’я. Розроблений мобільний додаток має наступний функціонал:

1) GPS-tracking велосипедної поїздки
2) Збереження поїздок
3) Можливість переглянути маршрут конкретної поїздки на карті та детальний аналіз її показників
4) Перегляд основних показників за певні часові періоди(день/тиждень/загалом)
5) Організація групових поїздок, можливість приєднання до них
6) Комунікація між учасниками групової поїздки

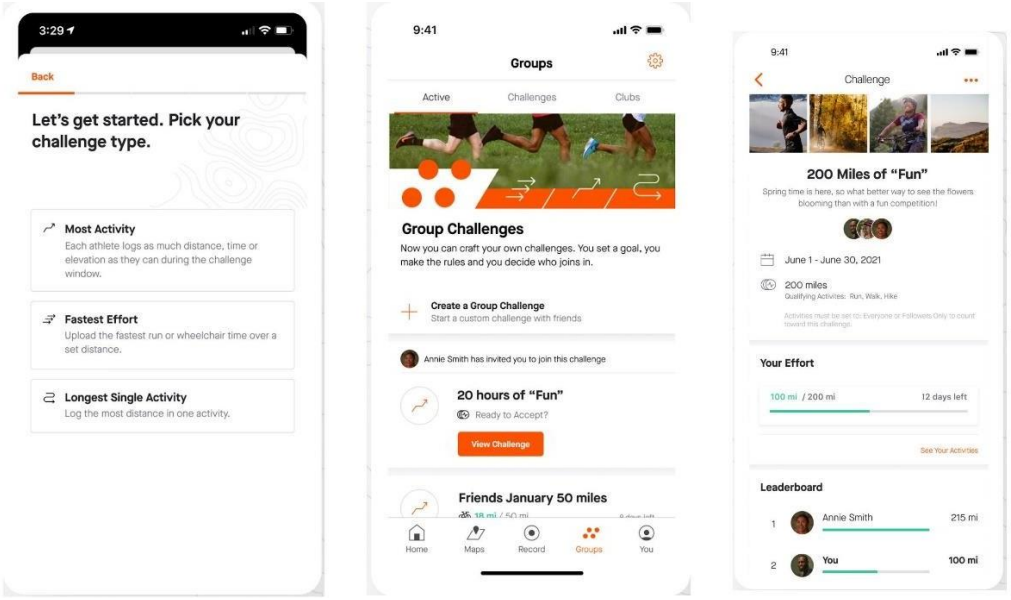
Додаток можна використовувати як продукт у сфері охорони здоров’я.

РОЗДІЛ 1 ПОПЕРЕДНІЙ ОПИС ДОДАТКУ

1.1 Попередній опис додатку

Зараз спортивні додатки, особливо для велосипедистів, користуються чималою популярністю, тому були проаналізовані ті, функціонал яких забезпечує інтеракцію між користувачами.

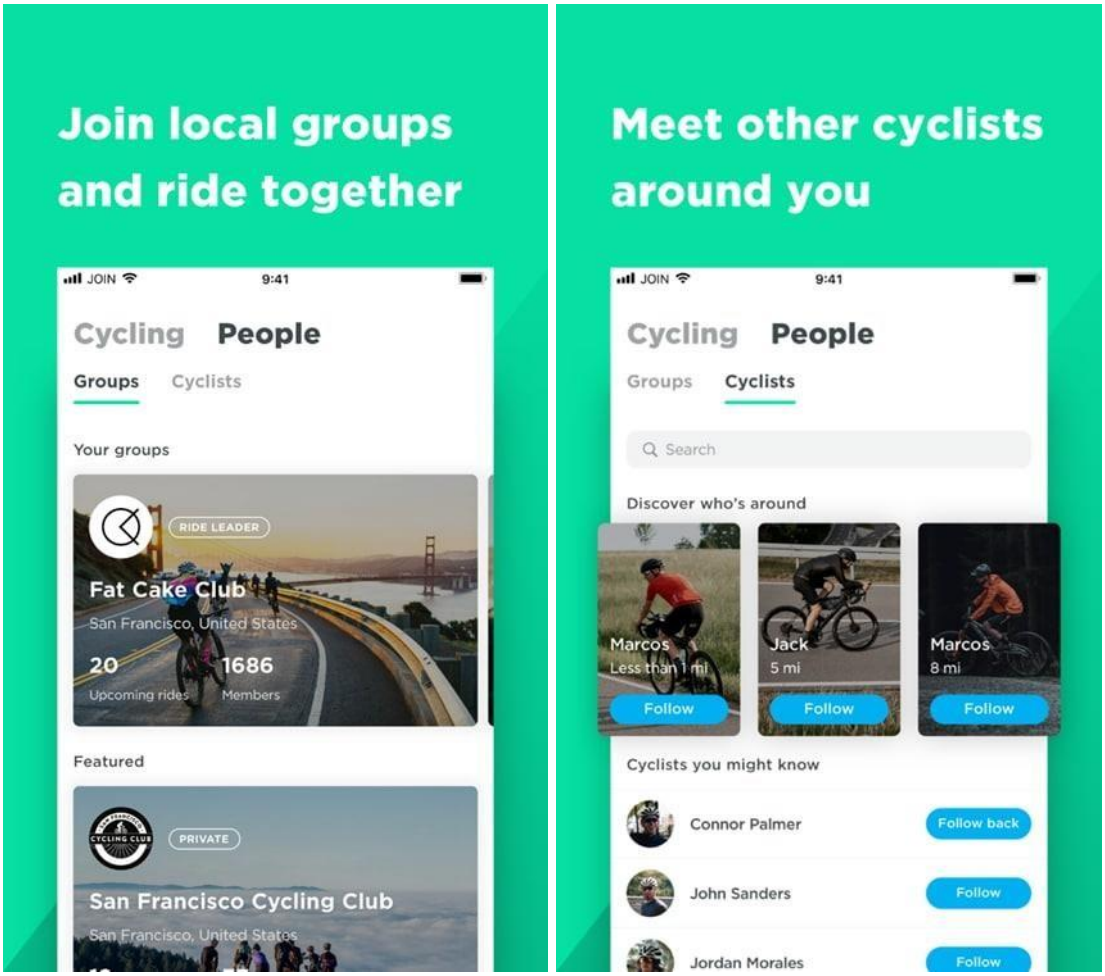
У додатку **Strava[1]** доступні так звані «Групові задачі». Вони дозволяють встановити певні цілі і часові рамки, змагатися з людьми на їх досягнення. В інтерфейсі групових задач можна відслідковувати прогрес, бачити положення спортсмена у задачі і переглядати список учасників. Групові задачі відкриті лише для тих користувачів, які знаходяться у конкретній групі. Із недоліків – є обмеження у кількості створених групових задач одним користувачем.



«Рисунок 1.1 - Додаток «Strava»

Також у користувачів є можливість створювати окремі «Клуби», всередині яких користувачі можуть комунікувати між собою, створювати пости, вести облік спортивних досягнень тощо.

JOIN[2] – це новий додаток, завдяки якому неймовірно легко створювати групові поїздки, ділитися маршрутом або знаходити його в новій місцевості під час подорожі. JOIN – це безкоштовна програма, яка акцентує увагу на соціальній природі катання. У програмі ви можете запросити інших приєднатися до вашої поїздки, поділитися маршрутом і навіть створити маршрут, просто натиснувши точки шляху та збереживши його.



«Рисунок 1.2 - Додаток JOIN»

Ви також можете переглядати інші групові поїздки, прокручувати результати та натискати, щоб приєднатися. Навіть якщо наразі немає групових поїздок, ви можете переглядати маршрути та слідувати їм, переглядаючи оцінки інших гонщиків за ці поїздки.

На основі проаналізованих вище додатків, було прийнято реалізувати у мобільному додатку систему організації групових поїздок, де організатор зможе обрати час та місце старту збору учасників. Користувачі мають змогу переглядати всі пропозиції та подати заявку до конкретної групової поїздки. Після приєднання користувачі зможуть комунікувати між собою у межах чату.

1.2 Огляд використаних технологій

Мобільний проект було розроблено в інтегрованому середовищі розробки мобільного програмного забезпечення Android Studio мовою програмування **Java[3]** для платформи Android. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains.

Для збереження даних додатку було використано **Cloud Firestore[4]** – хмарну NoSQL базу даних від Google, котра вважається покращенням Firestore Realtime Database. Cloud Firestore дозволяє зберігати дані на віддаленому сервері, легко отримувати доступ до них та слідкувати за змінами у режимі реального часу. Cloud Firestore забезпечує офлайн підтримку для різних мобільних ОС, швидші та складніші запити до БД порівняно з Firebase Realtime Database. В Firestore для збереження даних використовуються колекції та документи. Документ – це запис, який зберігає певні поля. Документи об’єднуються у колекції. Документ також може зберігати вкладені колекції. Якщо проводити аналогію з SQL-базою, то колекція – це таблиця, а документ – це запис в цій таблиці. Одна колекція може зберігати документи з різним набором полів.

Для збереження фотографій профілю користувача було використано **Firestore Storage[5]** – хмарне сховище, створене для розробників додатків, яким потрібно зберігати та обслуговувати створений користувачами вміст, наприклад фотографії чи відео. Це потужна, проста й економічно

ефективна служба зберігання об’єктів, створена для масштабу Google. Пакет SDK Firebase для хмарного сховища забезпечує захист Google для завантаження файлів у ваші програми Firebase, незалежно від якості мережі.

Ви можете використовувати наші пакети SDK для зберігання зображень, аудіо, відео чи іншого вмісту, створеного користувачами. На сервері ви можете використовувати API Google Cloud Storage для доступу до тих самих файлів.

Firebase Authentication[6] має на меті полегшити створення безпечних систем аутентифікації, одночасно покращуючи вхід та підключення для кінцевих користувачів. Це система аутентифікації на основі токенів, яка забезпечує легку інтеграцію з більшістю платформ. Вона забезпечує наскрізне рішення ідентифікації, підтримуючи облікові записи електронної пошти та паролів, автентифікацію телефону, а також вхід у Google, Twitter, Facebook та GitHub тощо, що надає цьому сервісу значну перевагу над іншими.

XML[7] – досить популярний формат для обміну інформації, що використовується багатьма веб-сервісами. Розмітка екрану, ресурси та інші файли вашого додатку складаються з XML-файлів. Загальний принцип роботи схожий до HTML – множина тегів в кутових дужках, які певним чином структуризовані та впорядкові.

Node JS[8] — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript.

1.3 Вимоги до мобільного застосунку

Проаналізувавши предметну область та існуючі мобільні додатки можна сформулювати наступні вимоги до мобільного застосунку.

Мобільне додаток повинен бути зручним у використанні, мати інтуїтивно-зрозумілий користувачеві дизайн та функціонал, згаданий вище, забезпечувати безпеку особистим даним користувача, мати просту для розуміння навігацію.

Мобільний додаток повинен бути розроблений з англійським інтерфейсом.

РОЗДІЛ 2 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

2.1 Покращення GPS-трекера

Було виділено два основні недоліки вже існуючого GPS-трекера:

1) Додаток міг відстежувати ваше переміщення лише тоді, коли вкладка «Track» була відкритою та активною на вашому пристрої
2) Користувач не мав змоги поставити поїздку на паузу

Реалізація GPS-трекеру, що працює у фоновому режимі, тобто коли додаток активно не використовується, є великою перевагою, адже користувач під час поїздки мав би змогу користуватися іншими додатками – наприклад музичним плеєром, месенджером чи здійснювати телефонний дзвінок за потребою. Часто виникає потреба призупинити трекінг – інколи у велосипедиста можуть виникнути певні проблеми під час їзди, що змусять його зупинитись. Наприклад, потрібно переїхати на інший бік дороги, а відповідно до ПДР[9] велосипедисту потрібно дочекатися зеленого світла світлофора та переходити частину дороги через пішоходний перехід, тримаючи велосипед у руках. Такі необхідні дії спричинять тривалу паузу.

Для покращення роботи GPS-трекеру було реалізовано окремий bound-service[10] GPSTrackerService. Перевага bound-service у тому, що вони може працювати незалежно від інших компонент додатку та при потребі ми зможемо з’єднати його з Activity[11] і передати або отримати певні дані. Фактично наш сервіс та Activity працюють у форматі сервер-клієнт.

Для отримання місцезнаходження було використано Google’s Location Services API[11] — службу визначення місцезнаходження, що частиною Google Play Services APK. Google’s Location Services API надає

нам спеціальний FusedLocationProvider, що дуже грамотно поєднує різні сигнали, що б якнайшвидше і якнайточніше передати інформацію про місцезнаходження, необхідну програмі. Провайдер керує основними технологіями визначення розташування, такими як GPS, Wi-Fi та інші; надає простий API, що дозволяє нам легко та зручно керувати службою. Крім того, FusedLocationProvider працює значно швидше ніж стандартний, використаний у курсовій роботі Android Location API, оскільки ви отримуєте місцезнаходження від загальносистемної служби, яка постійно його оновлює. Сервіс також буде При ініціалізації сервіса ми створюємо об’єкти **LocationRequest**, **LocationCallback** та ініціалізуємо **track_id** - ідентифікатор поточної поїздки для запису у БД.

LocationRequest задає конфігурацію параметрів обслуговуваггя запитів до FusedLocationProvider.

```
LocationRequest locationRequest = new LocationRequest();
locationRequest.setInterval(UPDATE_INTERVAL_IN_MIL);
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
;
locationRequest.setSmallestDisplacement(SMALLEST_DISPLASEMENT);
locationRequest.setWaitForAccurateLocation(true);
```

Ініціалізація **track_id:**

```
track_id = firebaseFirestore
        .collection("users")
        .document(firebaseUser.getUid())
        .collection("walks")
        .document().getId();
```

LocationCallback використовується для отримання сповіщень від FusedLocationProvider, коли розташування пристрою змінилося або більше не може бути визначено:

```
LocationCallback locationCallback = new LocationCallback(){
   @Override
   public void onLocationResult(LocationResult locationResult){
      super.onLocationResult(locationResult);
      onNewLocation(locationResult.getLastLocation());
   }
};
```

Функція onNewLocation() буде отримувати нові локації, записуватиме їх у документи Cloud Firestore у режимі реального часу за адресою: **/users/<user_id>/tracks/<track_id>/locations.**

Крім того, у сервісі ми оновлюватимо статистичні дані(пройдена відстань, висота, максимальна швидкість). Якщо при отриманні нової локації сервіс працюватиме у foreground-режимі(тобто вікно додатку не активне), користувач отримає нотифікацію, яка покаже йому оновлені дані поїздки:

```
private void onNewLocation(Location lastLocation) {
    updateStats(lastLocation);
    saveLocationToFirestore(lastLocation);
    if (serviceIsRunningInForeground(this)) {
       NotificationManager.notify(NOTIFICATION_ID,  createNotification());
}
```

Сервіс GPSTracker працюватиме у трьох режимах – «Started», «Stopped», «Paused». На стороні клієнта(Activity) для відстеження з'єднання з сервісом був використаний об'єкт класу ServiceConnection, що імплементує методи onServiceConnected та onServiceDisconnected. Після

приєднання Activity до сервіса, якому в залежності від поточного режиму трекера буде побудовано відповідний інтерфейс для користувача. Якщо поточним режимом трекера є «Started» або «Paused», тоді нам потрібно буде відновити вже пройдений маршрут. Для цього у методі recreateRoute() ми дістанемо всі локації, які були вже записані раніше.

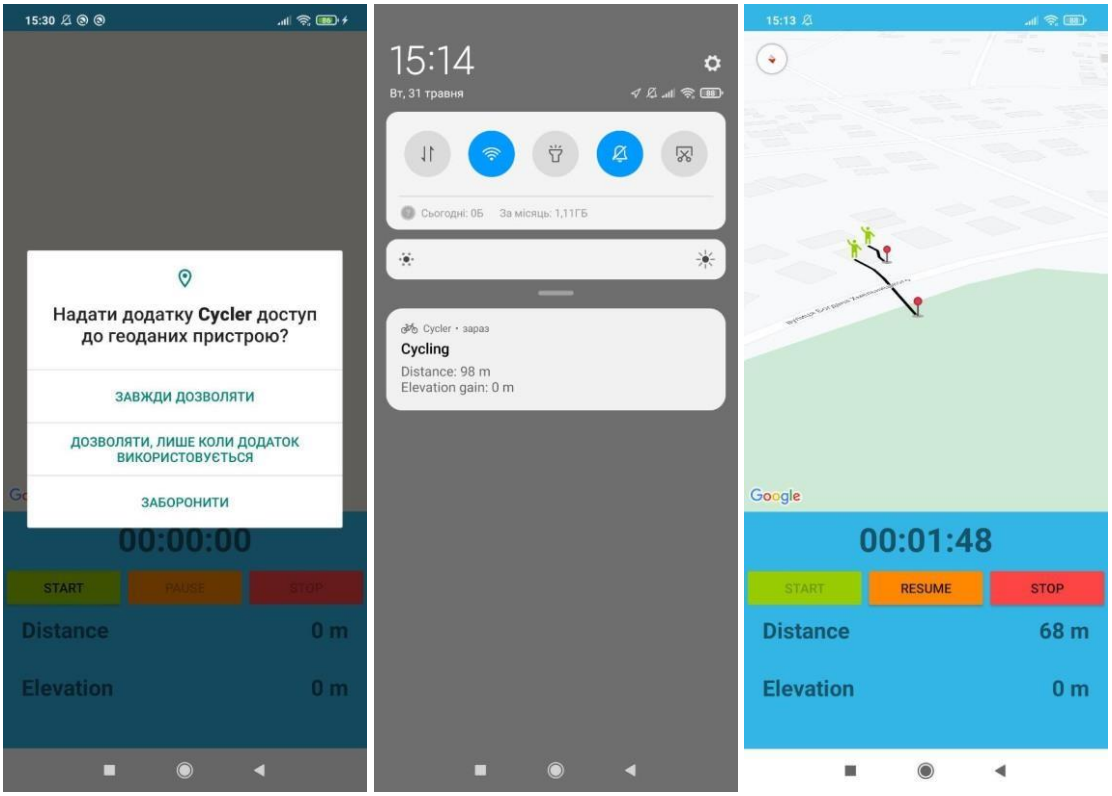
```
@Override
public void onServiceConnected(ComponentName className, IBinder service)
{
   GPSTrackerService.RunServiceBinder binder =
(GPSTrackerService.RunServiceBinder) service;
   GPSTrackerService = binder.getService();
   serviceBound = true;
   GPSTrackerService.background();
   int state = GPSTrackerService.isTrackerRunning();
   switch (state){
      case STARTED:
         recreateRoute();
         updateUIStart();
      case PAUSED:
         recreateRoute();
         updateUIPause;
      case STOPPED:
         updateUIStop();
   }
}
```

Якщо ми закриємо сторінку «Track» з увімкненим або у стані паузі сервісом, то ми продовжимо його роботу у foreground-режимі,

демонструючи дані поїздки для користувача у вікні сповіщень (див. рис. 2.1).

Починаючи з версії Android 8.1, для того що б програма мала змогу відстежувати місцезнаходження у фоновому режимі, користувачу потрібно надати дозвіл: **android.permission.ACCESS_BACKGROUND_LOCATION,** натиснувши на екрані запиту дозволів «Дозволити завжди» (див. рис. 2.1, а).

Локації, у яких користувач брав паузу будуть відображатися окремим маркером. Маршрути до та після паузи буде розділено на два окремих при відображенні(див. рис. 2.1, в).



«Рисунок 2.1 – Демонстрація GPS-трекера: а – запит дозволів у користувача, б – відображення пройденого маршруту, в – робота трекера у фоновому режимі»

2.2 Проектування бази даних для нової версії

У курсовій роботі вже було реалізоване збереження до локальної бази даних SQLite з наступною схемою:

«Рис 2.2 Збереження даних у SQLite у старій версії додатку»

У новій версії мобільного додатку нам потрібно буде також зберігати інформацію про всіх користувачів, прив’язати дані старих таблиці до конкретного користувача та організувати збереження інформації про організовані спільні поїздки.

Ми будемо зберігати наступну інформацію про кожного користувача:

- ID користувача
- Дата створення акаунту
- Електронна пошта(унікальне поле)
- Повне ім’я
- Нікнейм користувача(унікальне поле)
- URI фотографії профілю
- ID користувача
- Коротка біографія користувача

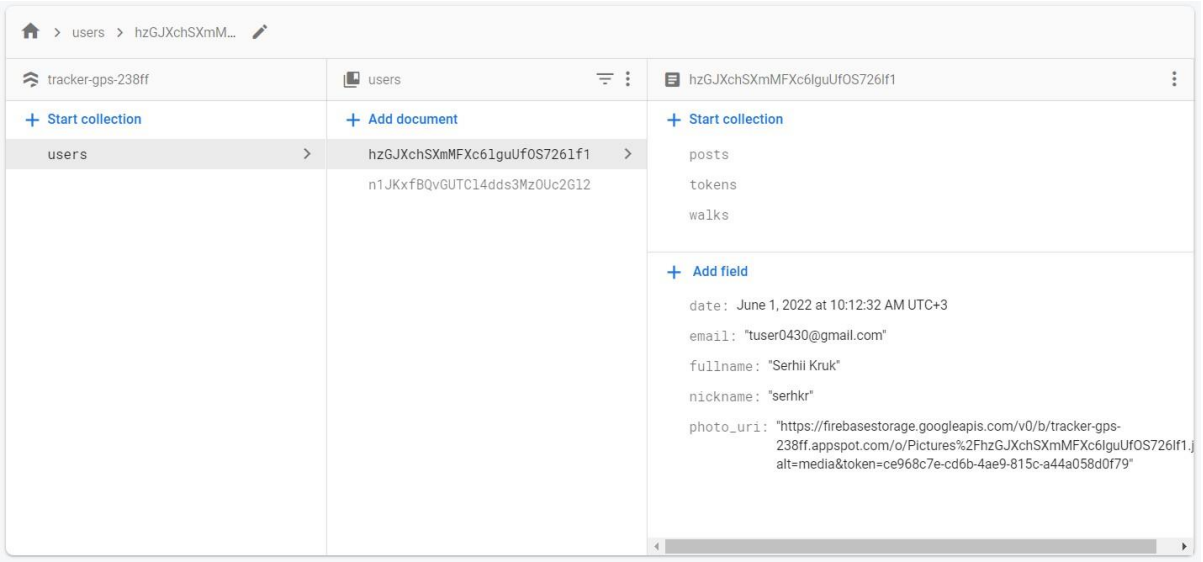
Кожна організована спільна ведосипедна поїздка буде містити наступну інформацію:

- Дата публікації
- Дата старту поїздки
- Опис поїздки
- Довгота точки старту поїздки
- Ширина точки старту поїздки
- Геохеш точки старту
- ID організатора
- Список людей, що приєдналась до поїздки(Їх ідентифікатори та час приєдання)

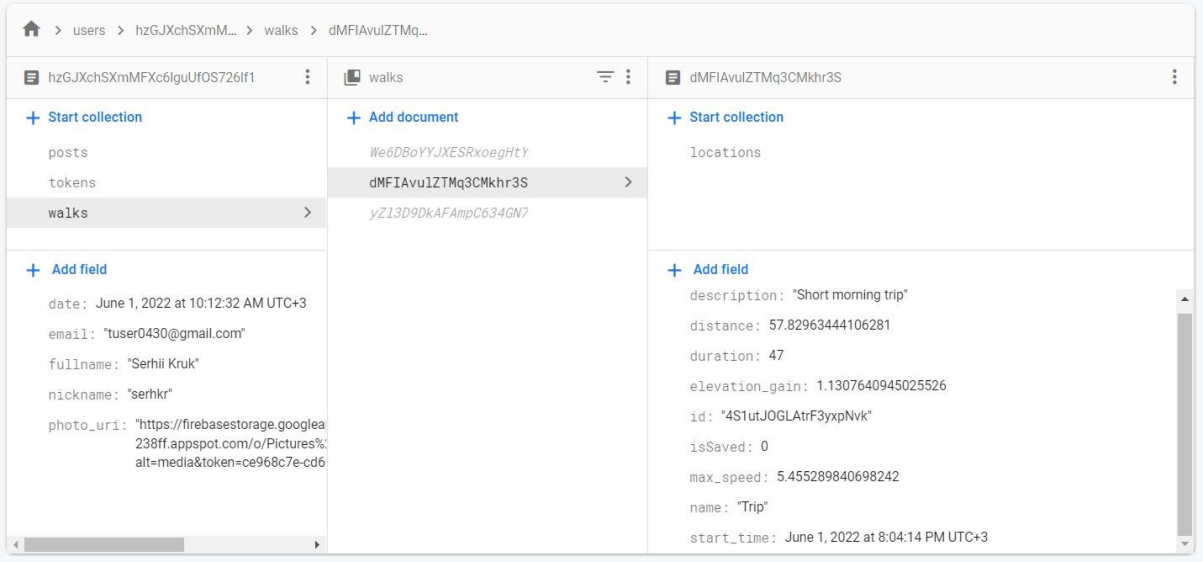
Дані, які будуть зберігатися для вже реалізованого функціоналу у курсовій роботі(збереження записаної поїздки) змінюватися не будуть. Збереження даних в Cloud Firestore було реалізовано наступним чином:

Для збереження інформації про користувачів була створена колекція «users»(див. рис. 2.3), в якій для кожного окремого користувача буде створюватись окремий документ, ID якого буде відповідати ID створеного користувача. Далі полями кожного документа будуть дата створення, біографія, електронна пошта, повне ім’я, нікнейм та URI фото. Крім того, є окрема колекція «tokens», що зберігатиме набір токенів користувача.

Кожна соло прогулянка користувача буде зберігатися як окремий документ з відповідним ID, у колекції «walks» (див. рис. 2.4) у документі користувача. Далі полями документа є опис, дистанція, тривалість поїздки у секундах, кумулятивна пройдена висота, максимальна швидкість у м/с. Поле isSaved вказує на те, чи зберіг користувач пройдений маршрут.

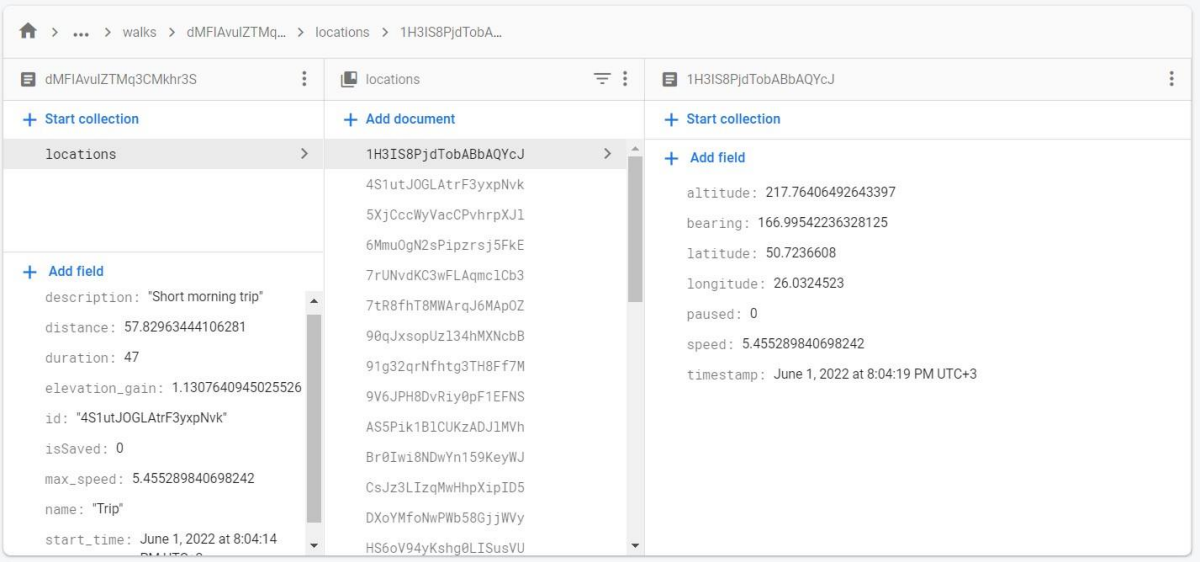


«Рисунок 2.3 – Збереження користувачів у БД»



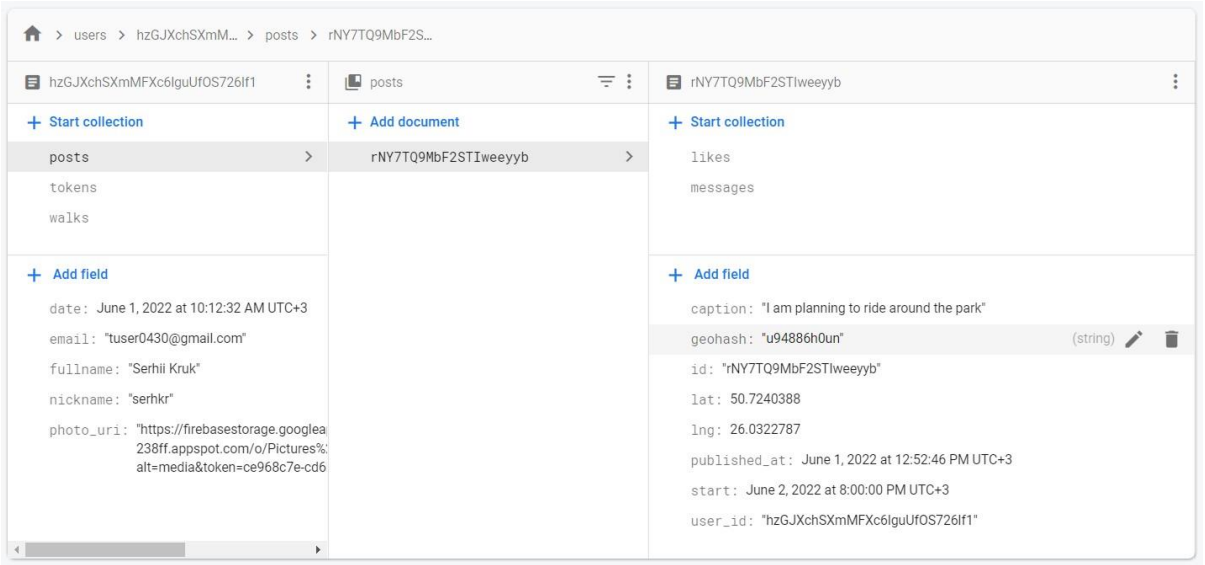
«Рисунок 2.4 – Збереження соло-поїздки у БД»

Пройдений маршрут поїздки буде зберігатися у вигляді наборів документів у колекції «locations» (див. рис. 2.5) всередині документу, де кожен документ відповідатиме конкретній точці маршруту, тобто зберігатиме координати, кут повороту, швидкість у поточний момент часу, час отримання локації та булеву змінну paused, що показуватиме, чи брав користувач паузу у цій локації.



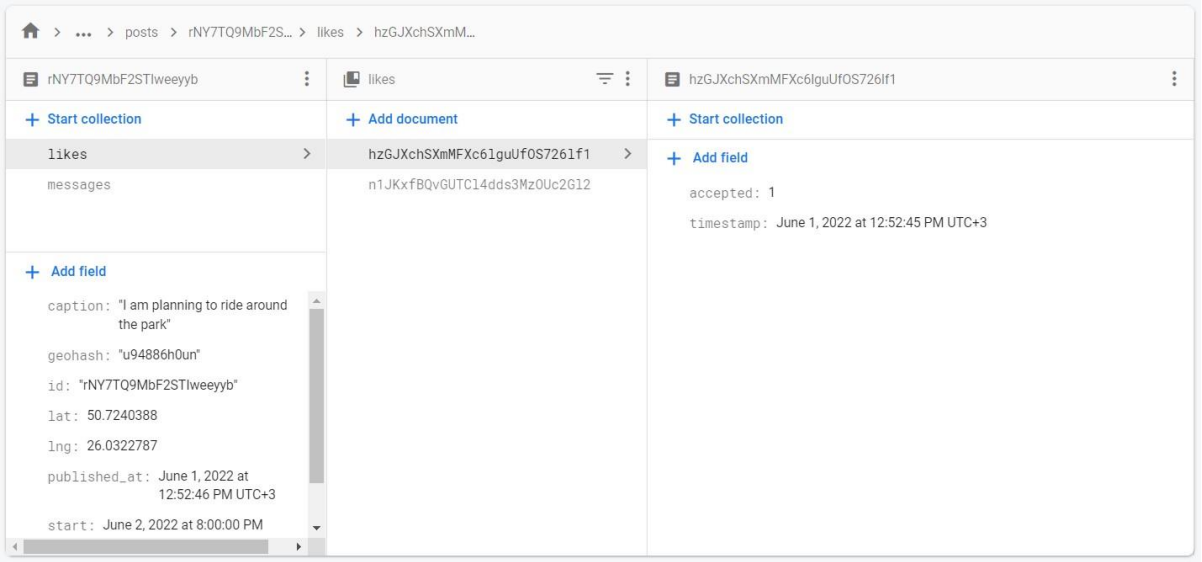
«Рис 2.5 – Збереження маршруту соло-поїздки у БД»

Кожна організована групова поїздка буде зберігатися у окремому документі у колекції «posts» (див. рис. 2.6), яка буде розміщена у документі відповідного користувача. Окремий документ матиме ID поїздки та зберігатиме опис, час старту, дату публікації, стартову точку прогулянки, її геохеш та ID організора.



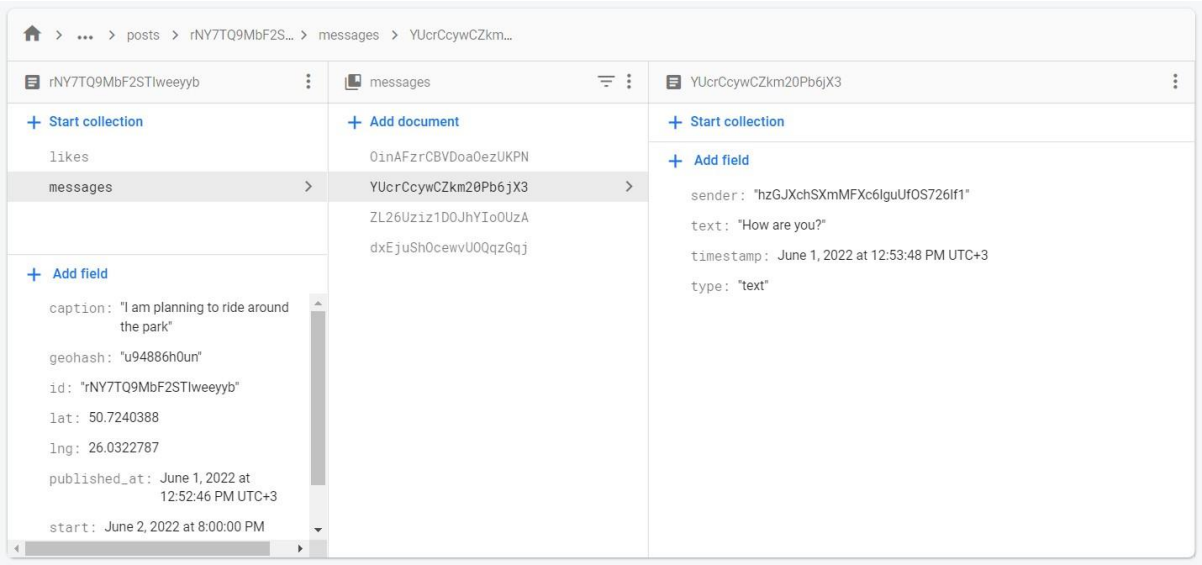
«Рис 2.6 – Збереження організованої групової поїздки у БД»

Список людей, що долучились до поїздки буде зберігатися у колекції «likes» (див. рис. 2.7), кожному приєднаному користувачу буде відповідати документ з ID користувача, полями якого будуть час запиту та його стан.



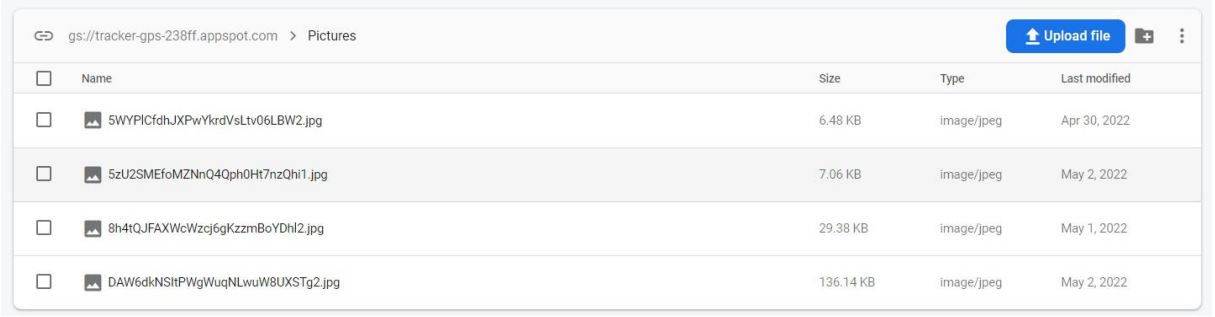
«Рис 2.7 – Збереження списку людей, що приєднались до поїздки»

Колекція «messages» (див. рис. 2.8) зберігатиме повідомлення у чаті. Повідомлення – окремий документ з полями: час надсилання, ID відправника та текст повідомлення.



«Рис 2.8 – Збереження повідомлень у чаті»

Фотографії профілів користувачів зберігаються у Firestore Storage наступним чином: у папці «Pictures» назва кожної фотографії відповідає ID користувача (див. рис. 2.8).



«Рис 2.9 – Збереження фотографій профілю користувачів у Firestore Storage»

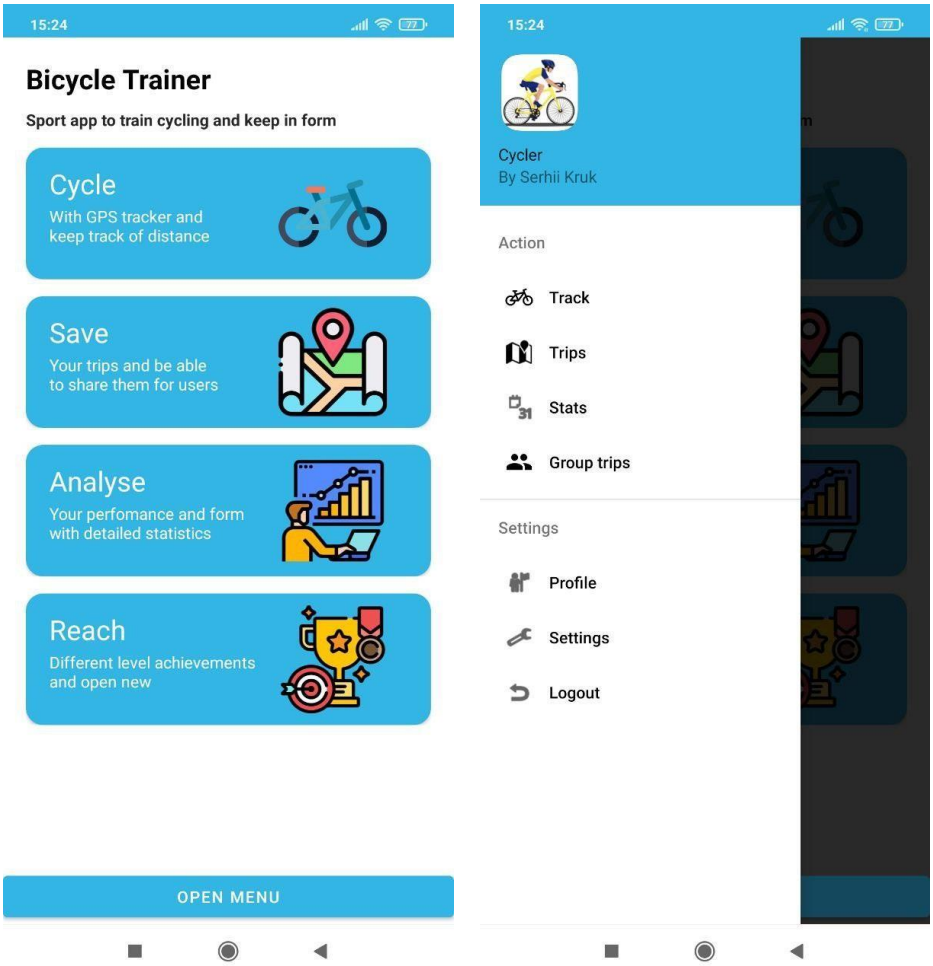
Cloud Firestore надає нам можливість зчитувати дані з БД прямо у об’єкти Java класів, що значно спрощує роботу. Великою перевагою є те, що для зберігання дат, часу, оперування ними Firebase надає спеціальний клас Timestamp. Так, наприклад для організованої групової поїздки було створено наступний клас GroupTrip з відповідними полями:

```
public class GroupTrip {
        private String id, organizator_id, caption, geohash;
        private Timestamp published_at, start;
        private Double lng, lat;
        private Double lat;
}
```

2.3 Дизайн мобільного додатку

Для розробки дизайну мобільного додатку було обрано шаблон Navigation Drawer[12], який надає користувачу дуже зручну, приємну у користуванні бокову панель навігації, яка допомагає перемикатися між різними блоками додатку. Панель навігації висувається ліворуч шляхом

слайду або натиском на кнопку і містить пункти призначення навігації для програми. Кожна вкладка – це окрема Activity у структурі додатку.



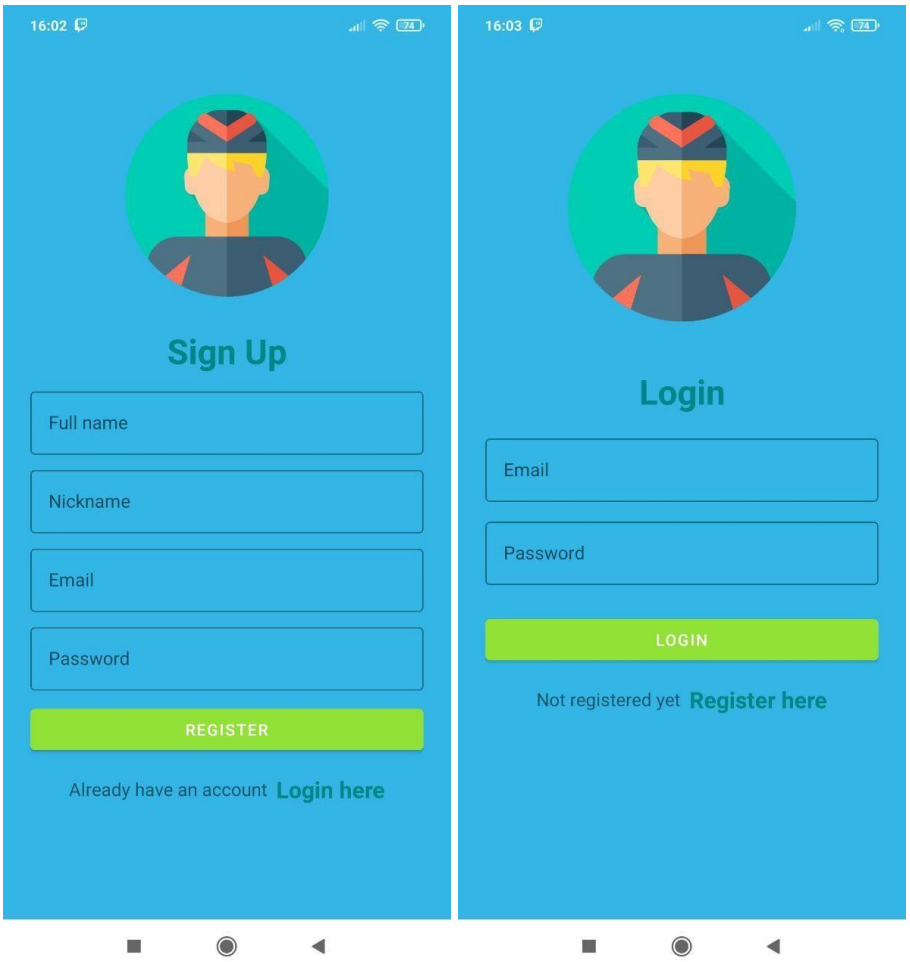
«Рисунок 2.11 – Головна сторінка додатку та бокова панель»

Були розроблені наступні вкладки:

- Track – для трекінгу поїздки
- Trips – для перегляду збережених поїздок
- Stats – перегляд вашої статистики за різні періоди часу
- Group trips – стрічка із організованих групових поїздок, до яких можна доєднатися
- Profile – сторінка вашого профілю
- Settings – сторінка нашаштувань (зміна паролю та електронної пошти)
- Logout – вихід з поточного акаунту

2.4. Авторизація користувачів

Для авторизації були розроблені наступні сторінки:



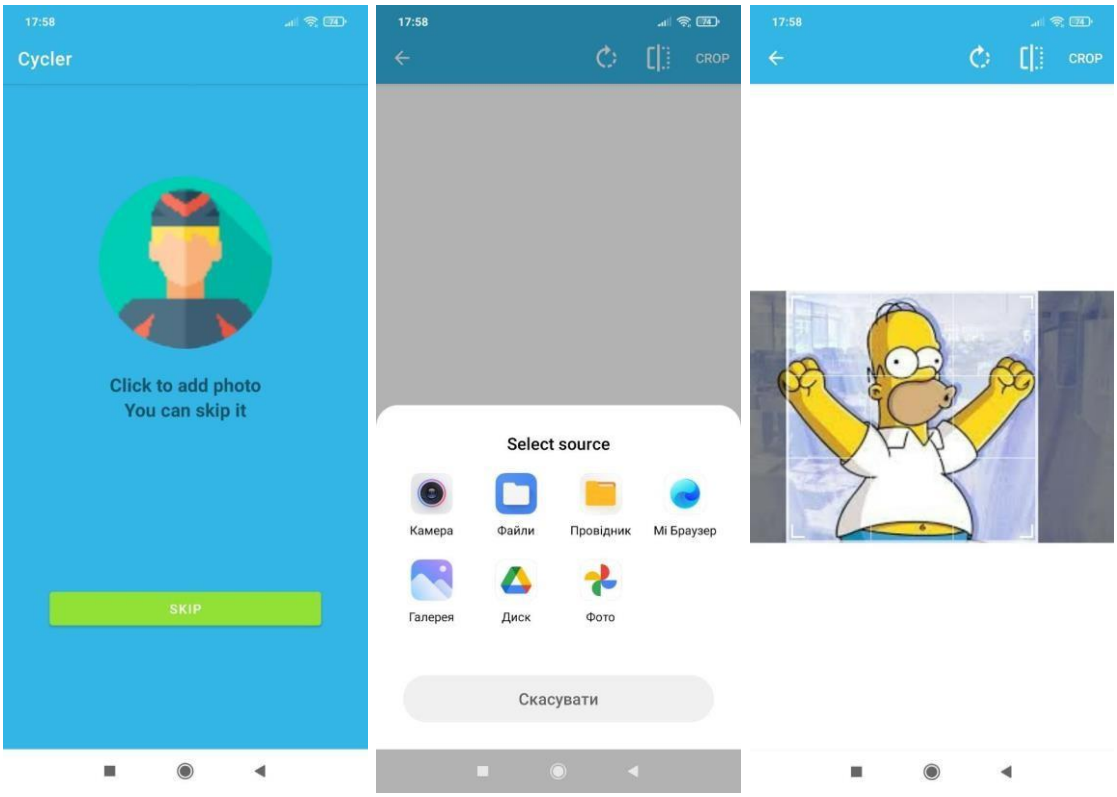
«Рисунок 2.12 – Сторінки для авторизації користувачів: а – сторінка логіну, б – сторінка реєстрації»

Для реєстрації нам знадобляться дані користувача: електронна адреса, пароль, нікнейм, повне ім’я. Після отримання цих даних від користувача ми передаємо їх у Firebase Authentication SDK. Сервіси Firebase оброблятимуть цю інформацію та створюватимуть користувачів у разі успіху.

Для вхідних даних поставлені наступні умови: нікнейми користувачів повинні бути унікальними, пароль повинен містити щойнайменше вісім символів, один спеціальний символ, одну цифру, одну велику букву. Зв’язок з сервером здійснюється за допомогою об’єкта класу

FirebaseAuth, створення акаунту за допомогою методів createUserWithEmailAndPassword, signInWithEmailAndPassword та асинхронного прослуховувача addOnCompleteListener, який очікуватиме на завершення операції.

Далі користувачеві буде запроповано вибрати фото для вашого профілю, у випадку згоди – користувач надає доступ до внутрішнього сховища телефона/камери та вибирає потрібну фотографію із джерела. Для відображення фотографії профілю були використані бібліотеки Android-Image-Cropper від ArthurHub[13], що надає можливість обрізки зображень та CircleImageView від hdodenhof[14], яка надає нам модифікований віджет ImageView, що відображає фотографію округлою.



«Рисунок 2.13 – Додавання фотографії: а – вигляд сторінки , б – вибір джерела фотографії, в – обрізання фотографії»

Для запуску Activity, яка дозволяє обрати та обрізати фотографію: CropImage.activity().setGuidelines(CropImageView.Guidelines.ON).setAspect Ratio(1, 1).start(this);

У XML файл для використання віджету потрібно додати наступний код елементу CircleImageView:

```
<de.hdodenhof.circleimageview.CircleImageView
    android:id="@+id/add_profile_pic"
    android:layout ="match_parent"
    android:src="@drawable/IC_CYCLICT "/>
```

2.5 Сторінки привітання, профілю та налаштувань

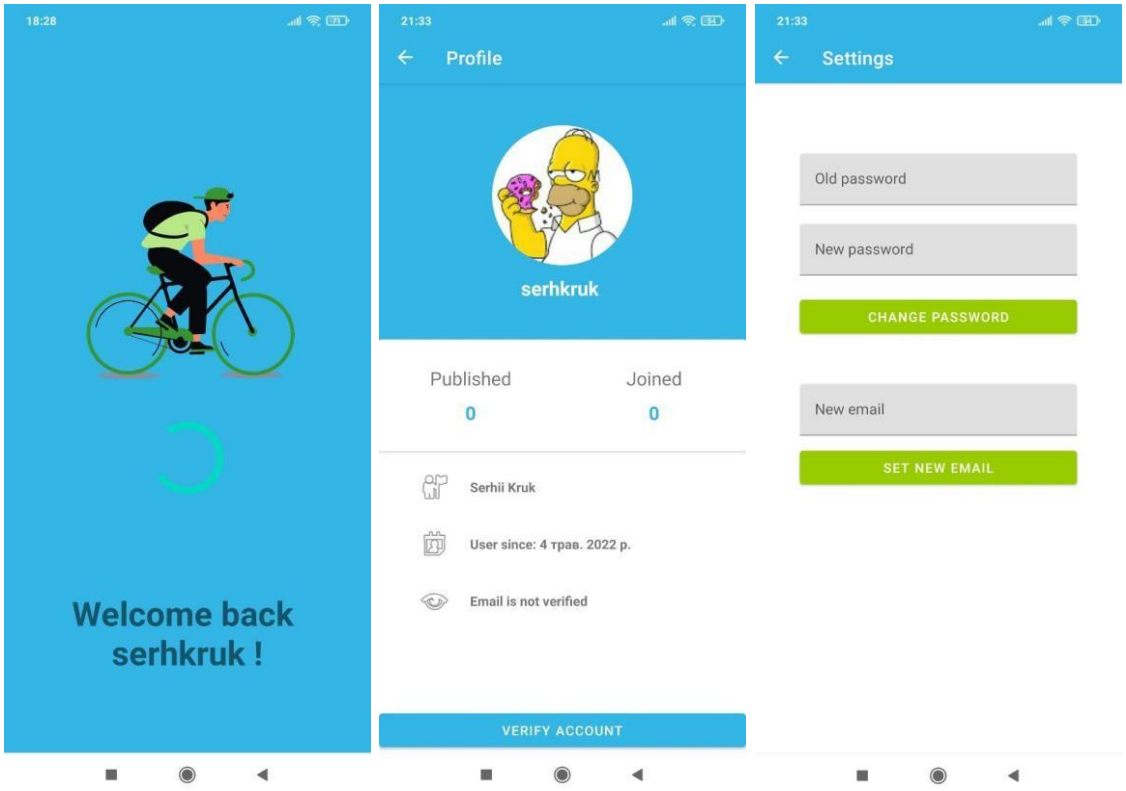
На сторінці профілю ми будемо відображати фотографію профілю, нікнейм, повне ім’я, дату створення, біографію, кількість організованих спільних поїздок та кількість поїздок, до яких користувач. Для підтвердження електронної пошти, потрібно натиснути на кнопку «Verify email», після чого на електронну пошту прийде смс з посиланням. На цій сторінці також можна редагувати вашу біографію та оновлювати фотографію. На сторінці налаштувань акаунту можна змінити пароль або електронну пошту, попередньо ввівши поточний пароль.

Якщо при вході у додаток ви вже були авторизовані, то замість сторінки для логіна, ви побачите сторінку з привітанням користувача та анімацією велосипедиста. Для відображення анімацій у мобільному додатку була використана бібліотека Lottie Animations[15]. Для того, щоб анімація працювала в автономному режимі, вона була включена у вихідні ресурси мобільного додатку у виді Json-файлу. Для демонстрації потрібно створити LottieAnimationView у XML файлі відповідної Activity.

```
<com.airbnb.lottie.LottieAnimationView
   android:id="@+id/splash_cycler_animation"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
```

app:lottie_autoPlay="true"

app:lottie_rawRes="@raw/intro_cycler"/>



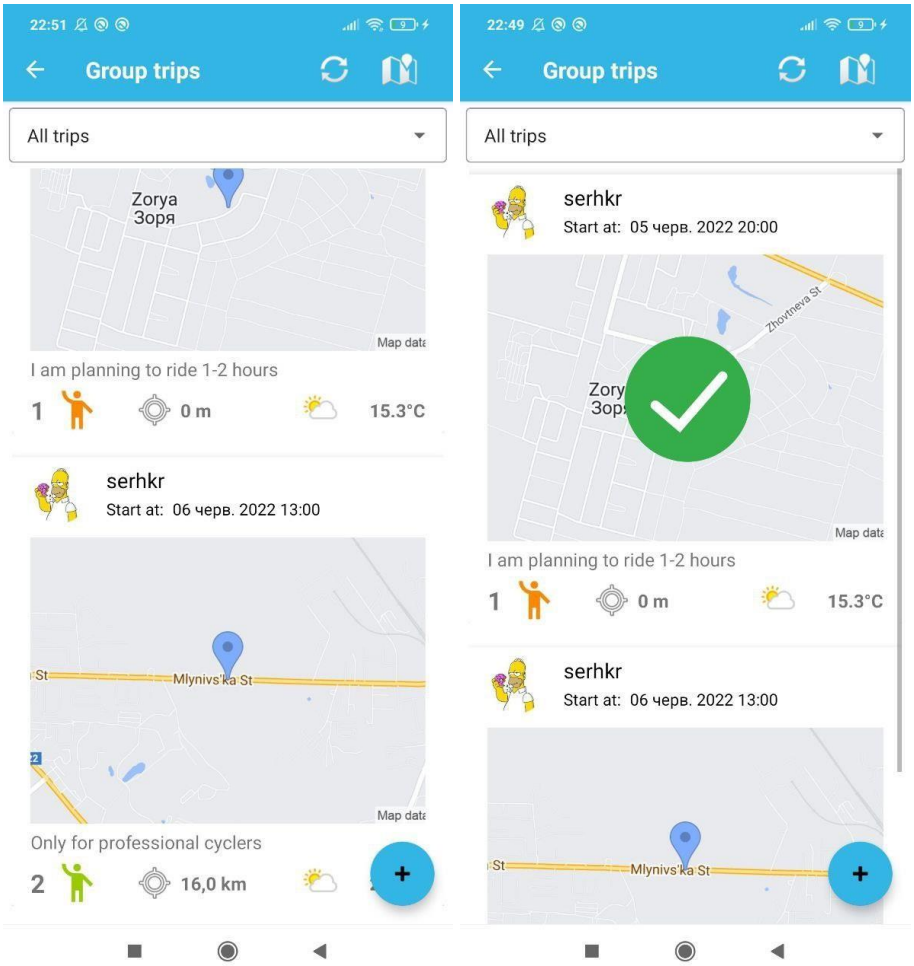
«Рисунок 2.14 – Сторінки додатку: а – сторінка привітання, б – сторінка профілю, в – сторінка налаштувань»

2.6 Сторінка «Group trips»

Групові поїздки, до яких можна приєднатись будуть відображатись у вигляді стрічки постів, де кожен пост міститиме інформацію про організатора(фото, нікнейм), дату старту поїздки, опис від організатора, мапу з відображеною точкою старту на ній. Додатково відображатиметься кількість людей, що приєднались до поїздки, відстань від вашого поточного місцезнаходження та прогноз погоди на час старту(температура та стан погоди у вигляді іконки).

Для відображення постів програмі потрібно дізнатися поточне місце користувача задля обчислення дистанції до старту. Пости будуть

сортуватись за датою початку подорожі. Є можливість переглянути список всіх поїздок, організованих вами поїздок та тих поїздок, до яких ви долучились за фільтром. Оскільки відображати для кожного посту фрагмент з повноцінною картою дуже ресурсо затратно, тому було прийняте рішення використовувати лише статичне зображення карти.



«Рисунок 2.15 – Сторінка «Group trips»: а –загальний вигляд, б – анімація приєдання після подвійного кліку»

Для цього було використано Google Maps Static API[16]. API дозволяє створювати окреме статичне зображення карти, створене зі стилю на основі Maps GL. Це означає, що ви можете додати карту на свій веб-сайт або мобільний додаток без JavaScript або інших плагінів. Google Maps Static API є Rest API, тому для отримання зображення нам потрібно відправляти

запити по певному URL, вказавши у параметрах ваш токен, точку центру, розмір зображення, зум, кут нахилу.

Код вивантаження статичного зображення карти:

```
String url = Uri.parse(googleDirectionsUrl)
        .buildUpon()
        .appendQueryParameter("origin", your_coords)
        .appendQueryParameter("dest",   start_coords)
        .appendQueryParameter("zoom", zoom)
        .appendQueryParameter("mode", "CYCLING")
        .appendQueryParameter("key", googleMapKey)
        .toString();
Glide
        .with(this)
        .load(url)
        .centerCrop()
        .into(map);
```

Для отримання прогнозу погоди у конкретній точці у конкретний час було використано WeatherAPI[17], що надає інформацію про прогноз погоди, історичну погоди, астрономічні дані тощо. Forecast Weather API дозволяє нам отримати прогноз погоди на наближчі два тижні погодинно. Для кожного типу погоди із запиту можна також отримати URL іконки. Для відправлення запиту в параметрах потрібно обов’язково вказати широту, довготу та дату у форматі dd-mm-yyyy.

Для відправлення запиту було використано бібліотеку OkHttp[18]. OkHttp — це стороння бібліотека, розроблена Square для надсилання та отримання мережевих запитів на основі HTTP. Вона побудований на основі бібліотеки Okio, яка намагається бути більш ефективною щодо читання та запису даних, ніж стандартні бібліотеки вводу/виводу Java, створюючи пул

спільної пам’яті. Параметри запиту до API – ширина та довгота точки та дата у форматі yyyy-mm-dd.

При збільшенні кількості доступних групових прогулянок, час на завантаження сторінки буде збільшуватись, тому було реалізована наступна логіка: пост буде завантажуватись лише тоді, коли користувач проскролить до його початку у стрічці. Для цього був використаний клас FirestorePagingAdapter[18], що надає нам зручне розбиття об’єктів на сторінки. FirestorePagingAdapter прив’язує об’єкт Query до елементу RecyclerView, завантажуючи дані на сторінках.

Отримаємо список всіх групових поїздок:

```
Query query = firebaseFirestore.collection("posts")
        .orderBy("start", Query.Direction.ASCENDING)
        .whereGreaterThan("start", Timestamp.now());
```

Потрібно задати об’єкт класу PagedList.Config, вказавши початкову кількість постів, що завантажаться з початку та розмір наступної «сторінки».

```
    PagedList.Config config = new PagedList.Config.Builder()
        .setInitialLoadSizeHint(5)
        .setPageSize(2)
        .build();
```

Далі потрібно створити об’єкт класу FirestorePagingOptions, що приймає на вхід список всіх об’єктів та конфігурацію пагінації.

```
FirestorePagingOptions<Post> options = new
FirestorePagingOptions.Builder<Post>()
        .setQuery(query, config, Post.class)
        .build();
```

Адаптер:

```
adapter = new FirestorePagingAdapter<Post, PostViewHolder>(options){
```

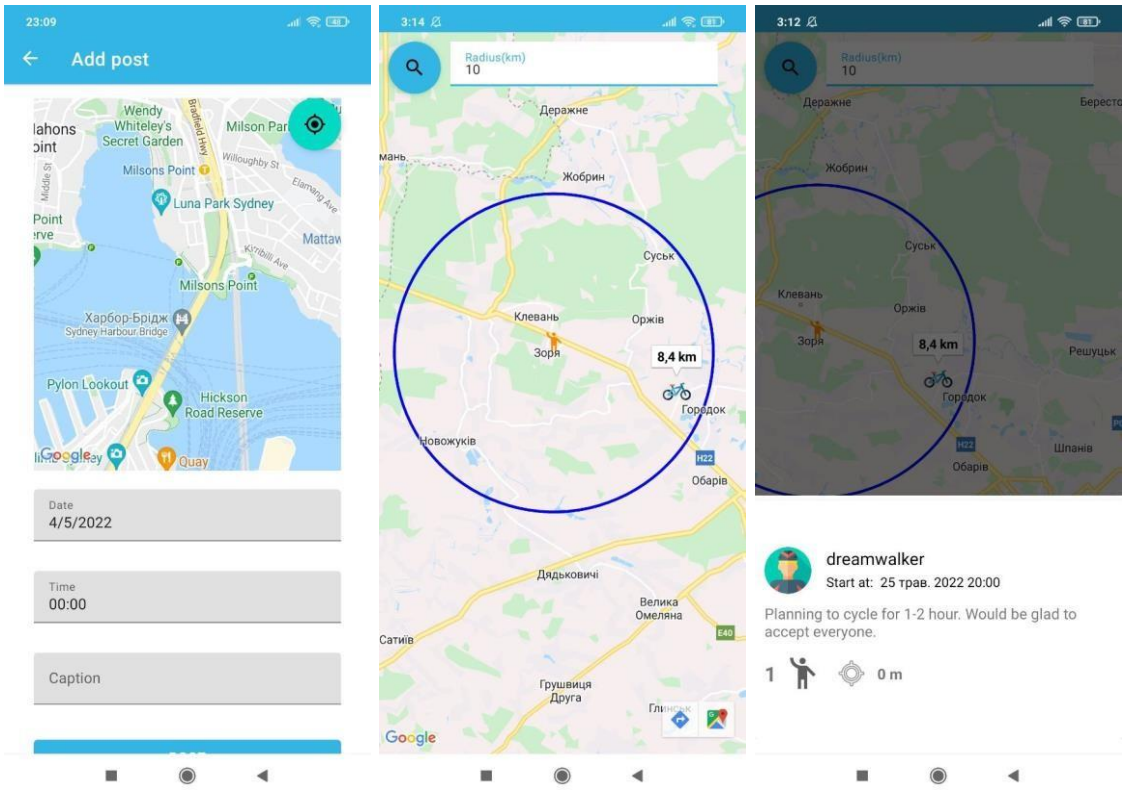
```
        @NonNull
        @Override
        public PostViewHolder onCreateViewHolder(@NonNull ViewGroup
        parent, int viewType) {
        View view =
        LayoutInflater.from(parent.getContext()).inflate(R.layout.each_post,
        parent, false);
        return new PostViewHolder(view);
    }
    @Override
    protected void onBindViewHolder(@NonNull PostViewHolder
        postViewHolder, int i, @NonNull Post post) {
    ...}
```

Після чого потрібно приєднати адаптер до елементу RecyclerView та почати прослуховування даних з Firestore:

```
recyclerView.setLayoutManager(new LinearLayoutManager(this));
recyclerView.setAdapter(adapter);
adapter.startListening();
```

Користувач, за умови верифікації електронної пошти, може сам організувати групову поїздку або приєднатись до вже створених, Для того, що б організувати нову групову поїздку, потрібно натиснути на кнопку «+» у правому нижньому куті, після чого у формі обрати час та дату старту, додати опис та вибрати на карті місце старту, можлива опція – вибрати поточне місцезнаходження, натиснувши кнопку у правому верхньому куті.

Крім цього, користувач може здійснювати пошук прогулянок на карті на окремій сторінці, вказавши радіус пошуку у кілометрах. Після чого на карті з’являться маркери велосипедів, натиснувши на які можна буде отримати інформацію про групову поїздку та приєднатися до неї.



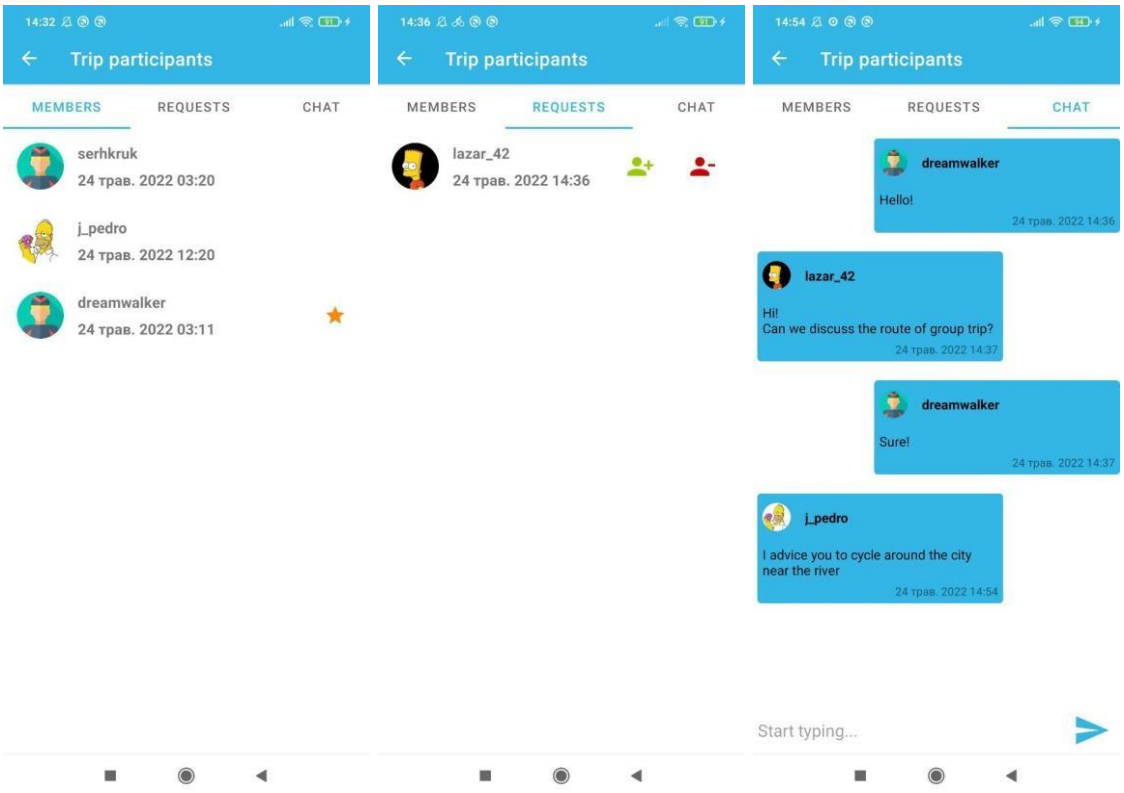
«Рисунок 2.17 – Сторінки додатку: а – додавання спільної поїздки, б-пошук доступних групових поїздок, в-вигляд поїздки»

Оскільки Cloud Firestore не дає змогу діставати дані за компонованими умовами(в нашому випадку, відстанню від нашого місцезнаходження до старту поїздки), тому для фільтрації поїздок було використано геохеш[19] – систему, яка кодує пару (широта, довгота) у об’єкт типу String. У системі геохешів карта світу поділена на прямокутну сітку. Кожен символ рядка геохешу визначає один з тридцяти двох розділів хешу префікса. Так наприклад, площина геохешу «abcd» повністю розташована у площині геохешу «abc». Таким чином, чим довший спільний префікс двох геохешів, чим ближче знаходяться дві локації. Тому використання геохешу для зберігання та запиту документів за позиціями в Cloud Firestore з достатньою ефективним, це значно прискорить фільтрацію даних, при цьому потрібно лише одне індексоване поле. Серед недоліків виділяють те, що використання геохешу дає не повністю точні результати,

тому нам потрібно буде відфільтрувати отриманий результат ще на стороні клієнту. Точність геохешу зменшується з наближенням точок до північного/південного полюсів, це пояснюється формою Землі.

2.7 Сторінка «Trip participants»

Для комунікації між користувачами, а саме учасниками групової поїздки була реалізована окрема сторінка «Trip participants» з трьома розділами «Members», «Requests», «Chat». На сторінці «Members» відображуються всі учасники групової поїздки та дати їх приєднання. Організоватор відмічений зірочкою. На сторінці «Requests» відображено користувачів, які надіслали заявку для приєднання, організатор може схвалили або відхилити конкретну заявку. Ця сторінка доступна лише організатору.



«Рисунок 2.18 – Сторінки додатку: а – додавання спільної поїздки, б-пошук доступних групових поїздок, в-вигляд поїздки»

Для того, що б отримати доступ до нього, необхідно, що б організатор схвалив вашу заявку. Сторінка «Chat» надає змогу учасникам надсилати повідомлення в межах створеної групи.

Чат працює наступним чином: при ініціалізації фрагменту "Chat" ми вмикаємо прослуховувач, який відслідковуватиме зміни в БД за адресою /users/<user_id>/trips/<trip_id>/messages. При отриманні нового повідомлення ми автоматично оновимо інтерфейс діалогу.

РОЗДІЛ 3 СИСТЕМА СПОВІЩЕНЬ КОРИСТУВАЧІВ

Сповіщення у додатку буде отримувати організатор поїздки, коли новий використувач надішле запит про приєднання. Аналогічно сповіщення отримає новий учасник групової поїздки, після того, як організатор підтвердить його заявку та додасть його у групу. Для надсилання сповіщень на конкретні девайси будуть використовуватися Firebase Messaging[19] токени. Токен користувача може змінюватися за декількох умов:

- Користувач оновив/перезавантажив додаток
- Користувач увійшов до акаунту з іншого смартфону
- Користувач очистив дані програми(кеш)
- Пройшов понад місяць часу використання токену

Оскільки у нашому випадку користувач може увійти до додатку з різних девайсів одночасно, тому кожному користувачеві можуть відповідати одночасно декілька токенів. У нашому додатку ми будемо оновлювати токени при реєстрації користувача, вході в акаунт та при простому відкритті додатку. Кожен токен зберігати час останнього використання, це дозволить нам видаляти з бази даних ті токени, які довго не використовувались користувачем – їх можна буде вважати застарілими. Для роботи з токенами та читанням вхідних повідомлень було створено окремий сервіс NotificationService, який унаслідує Notification Service - базовий клас для отримання повідомлень від Firebase Cloud Messaging. Розширення цього класу потрібне для того, щоб мати можливість обровляти вхідні повідомлення різних типів. Він також надає функціональні можливості для автоматичного відображення сповіщень і має методи, які викликаються для надання статусу повідомлень, що знаходяться у черзі. Перевагою є те, шо усі методи викликаються у фоновому потоці й можуть

бути викликані, коли програма працює у фоновому режимі чи навіть не відкрита. До маніфест файлу потрібно додати ініціалізацію сервісу:

```
<service
android:name=".YourFirebaseMessagingService"
android:exported="false">
<intent-filter>
<action android:name="com.google.firebase.MESSAGING_EVENT" />
</intent-filter>
</service>
```

Заміщений метод onNewToken() автоматично викликається при першому запуску додатку, коли токен для мобільного пристрою ще не був згенерований або тоді, коли термії дії старого токену досяг кінця і потрібно згенерувати новий. Отримати поточний токен можна за допомогою методу get_token() класу FirebaseMessaging:

```
@Override
public void onNewToken(@NonNull String mToken) {
        super.onNewToken(mToken);
        FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
        if(user != null)
        {
                save_token_to_db(user, mToken);
        }
}
```

Перевизначивши метод FirebaseMessagingService.onMessageReceived, ми можете виконувати дії на основі отриманого об’єкта RemoteMessage та отримати дані цього повідомлення. Сповіщення на Android представлені класом Notification. Об’єкт класу NotificationChannel дозволяє нам створити певний канал сповіщень з окремим ID та пріоритетністю відображення юзеру. Для відображення нотифікації у бекграунд-режимі було

використано системний сервіс NotificationManager. Notification.Builder надає інтерфейс конструктора для створення об’єкта Notification. Окремо можна ініціалізувати дію, яку слід виконати, коли користувач вибере сповіщення.

```
@Override
public void onMessageReceived(@NonNull RemoteMessage message) {
        String title = message.getNotification().getTitle();
        String body = message.getNotification().getBody();
        String CHANNEL_ID = "MESSAGE";
        getSystemService(NotificationManager.class)
        .createNotificationChannel(notificationChannel);
        Notification.Builder notification= new Notification.Builder(this,
                CHANNEL_ID)
                .setContentTitle(title)
                .setContentText(body)
                .setSmallIcon(R.drawable.ic_bike)
                .setAutoCancel(true)
        NotificationManagerCompat.from(this).notify(1, notification.build());
        super.onMessageReceived(message);
}
```

На серверній стороні логіка реалізована наступним чином: у Cloud Functions[22] ми хостимо хмарну функцію, яка реагуватиме на певну зміну у Firestore Database, після чого виконуватиме певний алгоритм (див. рис. 3.1). У нашому випадку тригером є запис нового ID юзера у колекцію likes певного поста.

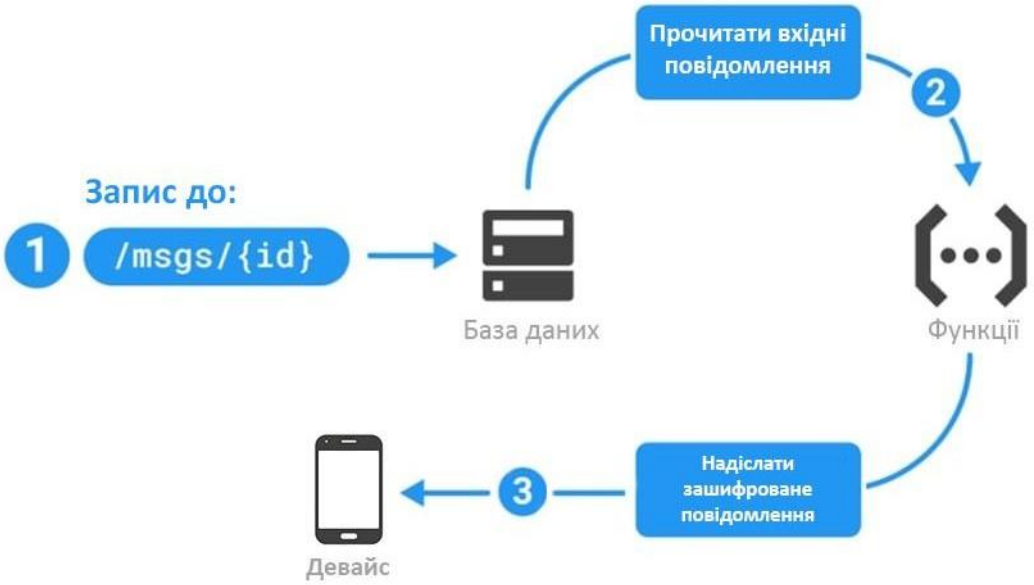
Було створено дві хмарні функції для надсилання нотифікацій користувачам. Перша функція showNotificationToOrganizator використовуватиме тригер onCreate, який реагуватиме на створення нового документа у колекції «likes», тобто на надсилання запиту організатору від

певного користувача, після чого надішле організатору повідомлення за його токенами:

exports.showNotificationToOrganizator = functions.firestore
.document("users/{userId}/posts/{postId}/likes/{joinedId}").onCreate(async (snap, context) => {…}

Друга функція showNotificationToJoinedUser використовує тригер onUpdate, який реагуватиме на зміну поля «accepted» у документах у колекції «likes», тобто на прийняття певного користувача організатором.

exports.showNotificationToJoined = functions.firestore
.document("users/{userId}/posts/{postId}/likes/{joinedId}").onUpdate(async (snap, context) => {...}



«Рисунок 3.1 – Принцип організації сповіщень»

ВИСНОВКИ

В ході виконання дипломної роботи було проаналізовано спортивні додатки для велосипедистів, що забезпечують інтеракцію між користувачами. Після постановки задачі, вже існуючий мобільний додаток було покращено шляхом реалізації авторизації користувачів та збереження даних на сервері. Новий функціонал додатку забезпечує соціалізацію користувачів, а вдосконалена версія GPS-трекера нажає користувачам більше можливостей.

Оскільки спостерігається ріст популярності велосипедних поїздок, то додаток і далі залишатиметься актуальним. У майбутньому планується розміщення мобільного додатку на онлайн-платформах, наприклад на Google Play, для того, що б користувачі могли використовувати його.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Strava [Електронний ресурс] – Режим доступу до ресурсу: https://www.strava.com/
2. Join [Електронний ресурс] – Режим доступу до ресурсу: https://join.cc/
3. Java Docs [Електронний ресурс] – Режим доступу до ресурсу: https://dev.java/learn/
4. Cloud Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore
5. Cloud Storage [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/storage
6. Firebase Authentication [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/auth
7. XML [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/
8. Node JS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/
9. Правила дорожнього руху [Електронний ресурс] – Режим доступу до ресурсу: https://vodiy.ua/pdr/
10. Bound Service [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/components/bound-services
11. Activity [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/android/app/Activity

12. Google location services API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/geolocation/overview
13. Navigation Drawer [Електронний ресурс] – Режим доступу до ресурсу: https://material.io/components/navigation-drawer
14. Android-Image-Cropper [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ArthurHub/Android- Image-Cropper
15. Circle Image View [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/hdodenhof/CircleImageView
16. Lottie Animations [Електронний ресурс] – Режим доступу до ресурсу: https://lottiefiles.com/blog/working-with-lottie/getting-started-with-lottie-animations-in-android-app
17. Google Maps Static API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/maps-static/overview
18. Weather API [Електронний ресурс] – Режим доступу до ресурсу: https://openweathermap.org/api
19. OK HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://square.github.io/okhttp/
20. Firebase Paging Adapter [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/firebase-developers/firestore-pagination-in-android-using-firebaseui-library-1d7fe1a75704
21. Firebase Cloud Messaging [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/cloud-messaging

22. Cloud Functions [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/functions