Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

***КУРСОВА РОБОТА***

з дисципліни: **«Компоненти програмної інженерії-4»**

на тему: **«Тестова документація проєкту “Мобільний додаток моніторингу перевищення швидкості та оптимальної швидкості транспортним засобом”»**

|  |  |
| --- | --- |
| Керівник – доцент Варава І.А. | Виконав Н.Г. Вознюк |
| Допущено до захисту | Студент 3-го курсу |
| «25» грудня 2024 р. | Групи ТВ-22 |
| Захищено з оцінкою |  |
|  |  |

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кафедра інженерії програмного забезпечення в енергетиці

**КУРСОВА РОБОТА**

з дисципліни: «Компоненти програмної інженерії-4»

на тему: «Тестова документація проєкту “Мобільний додаток моніторингу перевищення швидкості та оптимальної швидкості транспортним засобом”»

Студента 3 курсу групи ТВ-22

напряму підготовки 121 «Інженерія програмного забезпечення»

Вознюка Н. Г.

Керівник: доцент Варава І.А.

Національна оцінка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_ доцент Варава І.А. \_

(підпис) (посада, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (посада, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (посада, науковий ступінь, прізвище та ініціали)

Київ – 2024

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Інститут: Навчально-науковий атомної та теплової енергетики

Кафедра: Інженерія програмного забезпечення в енергетиці

Напрям підготовки 121 Інженерія програмного забезпечення

**ЗАВДАННЯ**

НА КУРСОВУ РОБОТУ СТУДЕНТА

Вознюку Назару Геннадійовичу

1.Тема роботи: «Тестова документація проєкту “Мобільний додаток моніторингу перевищення швидкості та оптимальної швидкості транспортним засобом”»

Керівник проєкту(роботи): Варава Іван Андрійович\_доцент

2.Срок подання студентом роботи: 28 грудня 2024 р.

3.Вихідні дані до проєкту(роботи): надати водію інформацію про перевищення швидкості та наближення до таких об’єктів як заправка та населений пункт

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): опис проекту із включенням усіх вимог, складання traceability matrix, тест-план з усіма пунктами, баг-репорт та практичні тести.

5. Дата видачі завдання: 03 жовтня 2024 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назви етапів виконання курсової роботи | Строк виконання етапів проєкту(роботи) | Примітка |
| 1 | Затвердження теми роботи | 09.10.2024 |  |
| 2 | Розробка програмного продукту | 07.11.2024 |  |
| 3 | Розробка тест плану | 13.11.2024 |  |
| 4 | Розробка баг репорту | 18.11.2024 |  |
| 5 | Опис вимог | 22.11.2024 |  |
| 6 | Тестування програми | 20.12.2024 |  |
| 7 | Оформлення пояснювальної записки | 28.12.2024 |  |

**Студент** Вознюк Н.Г.

**Керівник курсової роботи** доцент Варава Іван Андрійович

**АНОТАЦІЯ**

Виконуючи цю курсову роботу було створено мобільний додаток та описано процес його розробки та тестування. Для розробки використовувалися IDE для Android розробки AndroidStudio та мову програмування Kotlin.

Було проведено тестування програмного забезпечення, що дало змогу підтвердити виконання всіх поставлених завдань правильно. Головними методами тестування були unit-тестування та мануальне тестування. Також було розроблено супровідну документацію, з метою планування та спрощення виконання процесу тестування.

Обсяг пояснювальної записки 60 аркушів, кількість ілюстрацій 13, кількість таблиць 1, кількість додатків 3.

**ANNOTATION**

In this course work, a mobile application was created and the process of its development and testing was described. For the development, the IDE for Android development AndroidStudio and the Kotlin programming language were used.

The software was tested, which made it possible to confirm that all the tasks were performed correctly. The main testing methods were unit testing and manual testing. Also supporting documentation was developed to plan and simplify the testing process.

The volume of the explanatory note is 60 sheets, the number of illustrations is 13, the number of tables is 1, the number of applications is 3.

**ЗМІСТ**

[ВСТУП 8](#_Toc186856707)

[РОЗДІЛ 1 ОПИС ПРОЄКТУ 10](#_Toc186856708)

[1.1 Функціонал додатку 10](#_Toc186856709)

[1.2 Інтерфейс користувача 11](#_Toc186856710)

[1.3 Використані програмні засоби 14](#_Toc186856711)

[1.4 Вимоги до проєкту 15](#_Toc186856712)

[РОЗДІЛ 2 ТЕСТ-ПЛАН ПРОЄКТУ 16](#_Toc186856713)

[2.1 Ідентифікатор плану тестування 17](#_Toc186856714)

[2.2 Цілі тестування 17](#_Toc186856715)

[2.4 Елементи тестування 17](#_Toc186856716)

[2.5 Функціонал, що тестується 18](#_Toc186856717)

[2.5.1 Отримання даних про геолокацію 18](#_Toc186856718)

[2.5.2 Обробка даних геолокації 18](#_Toc186856719)

[2.5.3 Відправка даних на користувацький інтерфейс 19](#_Toc186856720)

[2.5.4 Перевірка роботи частин інтерфейсу 20](#_Toc186856721)

[2.6 Функціонал, що не тестується 20](#_Toc186856722)

[2.7 Підхід 21](#_Toc186856723)

[2.7.1 Інструменти тестування 21](#_Toc186856724)

[2.7.2 Збір показників та метрики 21](#_Toc186856725)

[2.7.3 Конфігурації що будуть тестуватися 21](#_Toc186856726)

[2.7.4 Підхід до тестування додатку 21](#_Toc186856727)

[2.8 Критерії проходження/провалу для плану тестування 22](#_Toc186856728)

[2.9 Критерії призупинення та умови відновлення тестування 22](#_Toc186856729)

[2.9.1 Критерії призупинення тестування 23](#_Toc186856730)

[2.9.2 Критерії відновлення тестування 23](#_Toc186856731)

[2.10 Результати тестування 23](#_Toc186856732)

[2.11 Тестові завдання 23](#_Toc186856733)

[2.11.1 Залежності між тестами 23](#_Toc186856734)

[2.11.2 Рівень навичок 24](#_Toc186856735)

[2.11.3 Ключові етапи 24](#_Toc186856736)

[2.12 Потреби середовища 24](#_Toc186856737)

[2.12.1 Джерела інформації 24](#_Toc186856738)

[2.12.2 Безпека 24](#_Toc186856739)

[2.12 Відповідальності 25](#_Toc186856740)

[2.13 Ризики 25](#_Toc186856741)

[2.13.1 Зміна дат 25](#_Toc186856742)

[2.13.2 Надто низька толерантність до помилок 25](#_Toc186856743)

[2.14 Затвердження 25](#_Toc186856744)

[РОЗДІЛ 3 РОЗРОБКА БАГ РЕПОРТУ 26](#_Toc186856745)

[3.1 Список знайдених багів 26](#_Toc186856746)

[3.2 Складений баг репорт 26](#_Toc186856747)

[3.2.1 Тема багу 26](#_Toc186856748)

[3.2.2 Детальний опис багу 27](#_Toc186856749)

[3.2.3 Кроки до відтворення 27](#_Toc186856750)

[3.2.4 Результати 27](#_Toc186856751)

[3.2.5 Пріоритетність, серйозність, статус та оточення 27](#_Toc186856752)

[3.3 Розбір рішення багу 28](#_Toc186856753)

[3.3.1 Розбір причини проблеми 28](#_Toc186856754)

[3.2.2 Огляд можливих рішень 28](#_Toc186856755)

[3.2.3 Результат виправлення 29](#_Toc186856756)

[Розділ 4 ТЕСТУВАННЯ ПРОЄКТУ 30](#_Toc186856757)

[4.1 Середовище для тестування 30](#_Toc186856758)

[4.2 Unit-тестування 31](#_Toc186856759)

[4.2.1 Переваги та недоліки unit-тестування 31](#_Toc186856760)

[4.2.2 Розробка unit-тестів 32](#_Toc186856761)

[4.3 Інтеграційне тестування 34](#_Toc186856762)

[4.3.1 Написання інтеграційних тестів 34](#_Toc186856763)

[4.4 Результати unit та інтеграційного тестування 38](#_Toc186856764)

[4.5 Мануальне тестування 39](#_Toc186856765)

[ВИСНОВОК 41](#_Toc186856766)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 42](#_Toc186856767)

[**ДОДАТОК 1** 43](#_Toc186856768)

[**ДОДАТОК 2** 45](#_Toc186856769)

[**ДОДАТОК 3** 49](#_Toc186856770)

# ВСТУП

В наш час автомобілів на дорогах стає все більше і більше, тому за дорогою треба спостерігати ще більше. Водії постійно користуються своїм смартфоном для того, щоб прокладати маршрути і орієнтуватися на місцевості, проте це може відволікати і відвернути увагу водія від підтримання швидкісного ліміту. Це може в кращому випадку призвезти до отримання водієм штафу, а в гіршому — аварії на дорозі. Через це корисним буде створити систему, яка зможе попередити водія про те, що він перевищив швидкість відрізку дороги де він перебуває. Для цього проєкту тестування є важливим етапом, щоб забезпечити правильність збору та відображення даних.

Вибір виду тестування базується на функціоналі, який надає проєкт, специфіки його роботи та вимог, які перед ним ставляться. У випадку мобільного додатка, який визначає швидкість руху, важливим є коректність отримання даних, відображення їх користувачу, бо додаток має надавати інформацію як про швидкість руху, так і про швидкісні обмеження і найближчі населені пункти та заправні станції. З цього можна зробити висновок, що головними видами тестування, в цьому випадку, є функціональне тестування, щоб впевнитися в правильності роботи функцій програми, інтеграційне тестування, щоб перевірити взаємодію між такими компонентами як GPS та візуальний інтерфейс та юзабіліті тестування, яке зебезпечує простоту і зручність використання інтерфейсу додатку.

Тестування проєкту, що створюється в рамках цієї курсової роботи, забезпечується unit-тестами, які перевіряють різні умови та впевнюються, що дані обробляються правильно. Також невід’ємною частиною є тестування візуального інтерфейсу на правильність відображення даних. Це досягається шляхом мануального тестування та тестування або використанням фреймворків таких як Espresso. У випадку додатку визначення швидкості додатково потрібно використовувати mock-дані про геолокацію.

Використання такого роду тестів допоможе ефективно виявити помилки, та, відповідно, пришвидшити їх виправлення. Також при їх розробці важлива увага приділяється створенню якомога-більшої кількості сценарів, щоб зменшити шанси на виникнення багів різного ступеню критичності.

Таким чином, в рамках цієї курсової роботи, було створено не лише проєкт, який надаватиме користувачу інформацію про перевищення швидкості. Невід’ємною його частиною, є проведення тестування, щоб забезпечити правильність роботи та зручність використання на найвищому рівні.

# РОЗДІЛ 1 ОПИС ПРОЄКТУ

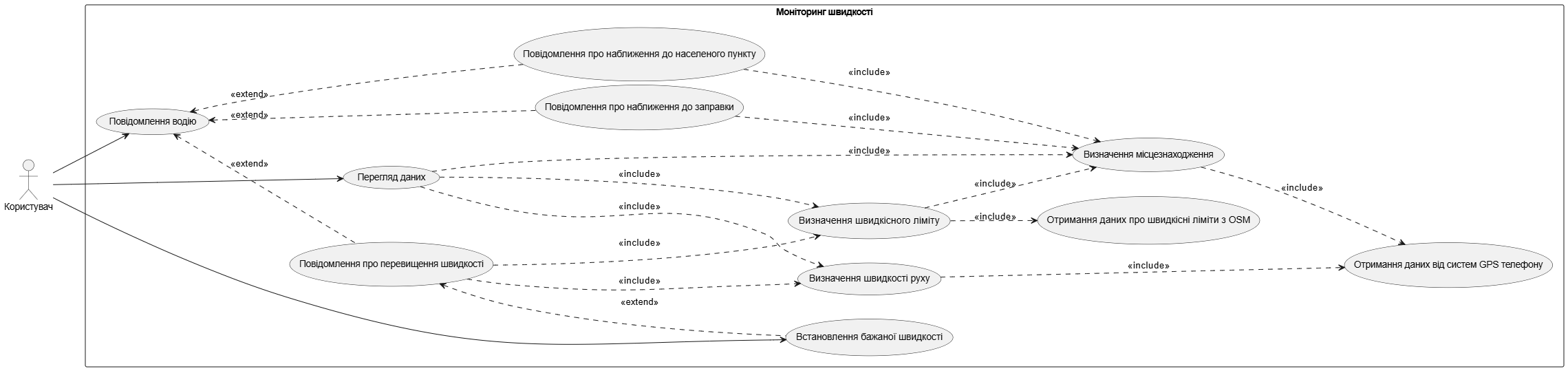
В рамках курсової роботи був створений мобільний додаток для визначення швидкості руху автомобіля та оповіщення водія про перевищення швидкості. Цей додаток має поєднувати зручність та інтуїтивність користування, а також функціонал, який зможе допомогти водію краще орієнтуватися в дорозі та зважати на обмеження швидкості. Також варто відзначити, що додаток не зберігає даних про користувача, а лише використовує поточне місцезнаходження, для пошуку інформації на OSM картах.

## Функціонал додатку

Весь функціонал додатку можна розділити на 2 головні частини. Перша це відстеження швидкості та місцезнаходження. Це ядро системи, яке забезпечує пошук інформації про швидкісні обмеження та навколишні населені пункти та заправні станції. Воно дає можливість зв’язуватися з сервісами, які містять потрібну інформацію. Другий компонент функціоналу системи є оповіщення. Ключове його завдання, сповістити водія про те, що він перевищує швидкість, проте тут ще є додатковий функціонал. По-перше, додаток визначає навколишні населені пункти в радіусі десяти кілометрів відносно користувача та заправні станції в радіусі півтора кілометра, то в разі зміни найближчого поселення, або заправної стації він сповіщає про це водія. По-друге, оскільки додаток працює постійно в фоновому режимі, то він створює сповіщення з актуальною швидкістю руху користувача, яке постійно оновлюється.

Також варто відзначити можливість користувача задавати власну бажану швидкість, щоб поїздка була ще комфортнішою. У випадку, якщо бажана швидкість руху користувача буде більшою за дозволену на певному відрізку дороги, то користувач буде оповіщений про перевищення обмеження загального, але у випадку, коли бажана швидкість менша — користувач буде оповіщений про перевищення саме бажаної швидкості.

Функціональну декомпозицію, яка ілюструє зв’язок між розглянутим функціоналом можна побачити на рисунку 1.1.

Рисунок 1.1 — Функціональна декомпозиція додатку

## Інтерфейс користувача

Інтерфейс користувача можна побачити на рисунку 1.2.



Рисунок 1.2 — Інтерфейс користувача при вході в додаток

Інтерфейс користувача складається з 5 основних компонентів. Перший «Швидкість» відображає актуальну швидкість руху користувача, яка береться з даних про геолокацію. Друге поле «Швидкісний ліміт» відображає користувачу інформацю про обмеження швидкості на частині дороги де зараз перебуває користувач. Якщо інформації про обмеження швидкості не було на OSM картах, то в цьому полі буде напис «Н/З». Приклад, коли швидкісне обмеження не було знайдено можна побачити на рисунку 1.3.



Рисунок 1.3 — Інтерфейс додатку, коли швидкість не знайдено

Третє поле це «Власний ліміт». Воно надає користувачу можливість ввести числове значення, що більше нуля, яке позначатиме його бажану швидкість відповідно. Четверте поле «Близько до» об’єднує в собі інформацію про найближчий населений пункт та відстань до нього, та найближчу заправну станцію. У разі, якщо ніякої інформації не було знайдено, то користувач побачить напис «Не знайдено» у полі де мала бути назва, та прочерк у полі де мала бути швидкість. Це можна побачити на рисунку 1.4. П’ятий та останній елемент інтерфейсу користувача це кнопка «Зупинити». Вона припиняє відстеження місцезнаходження та закриває додаток.



Рисунок 1.4 — Інтерфейс користувача, коли інформацію про заправки не знайдено

Також непрямим елементом інтерфейсу користувача можна вважати сповіщення, які він отримує. Приклад таких сповіщень можна побачити на рисунку 1.5.

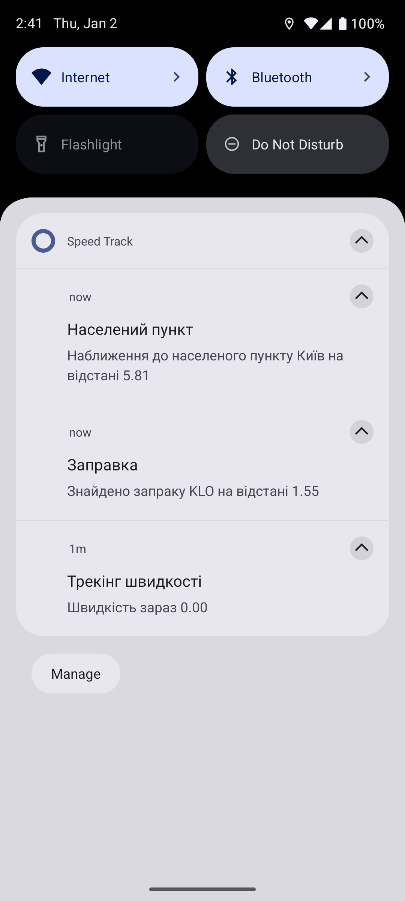


Рисунок 1.5 — Сповіщення, які може отримати користувач

## Використані програмні засоби

Проєкт був розроблений з використанням мови Kotlin, яка поширена в сучасній розробці мобільних додатків на Android. Як середовище розробки використовувалося IDE AndroidStudio. Його функціонал, забезпечує, як можливості розробки, так і тестування. У ньому є можливість одразу створювати візуальний інтерфейс, переглядати його на різних девайсах, бо в сучасних реаліях, кількість девайсів, які використовують Android дуже велика. Також це середовище надає повноцінний емулятор, який підтримує всі версії операційної системи Android.

Для отримання даних про швидкісні обмеження, населені пункти та заправні станції було використано Overpass API. Це API, яке надає користувачам широкий інструментарій для пошуку та початкової обробки інформації, що зберігається на OSM картах. В свою чергу OSM (OpenStreetMap) — це міжнародний проєкт, який покликаний створити відкриту мапу світу. Вони надають можливість всім користувачам додавати інформацію про місця, о’бєкти та інфраструктуру.

Для визначення відстані до найближчої заправної станції використовувалося також API під назвою OSRM. Воно надає функціонал з пошуку відстані до об’єкту, але на напряму, а за найкоротшим маршрутом. Цей сервіс є безкоштовним на відміну від аналогічного функціоналу API GoogleMaps, проте воно має трохи меншу точність. Для створення запитів до цих API використовувалася бібліотека okhttp3. А для опрацювання результатів, які повертають раніше згадані сервіси бібліотека Gson.

Для тестування функціональної частини, використовувалися JUnit4 тести. Вони підтримуються мовою Kotlin, та можуть за потреби виконуватися напряму на мобільному пристрої. Також використовувалася бібліотека MockK для створення підставних об’єктів для перевірки. Для мануального тестування, використовувалися віртуальна машина AndroidStudio, а також функціонал для симуляції руху, який надається цим же середовищем розробки.

## Вимоги до проєкту

Вимоги до проєкту представляються у вигляді функціональних та нефункціональних вимог. Список функціональних вимог можна побачити в таблиці 1. До нефункціональних вимог додатку, що визначає швидкість та повідомляє водія про перевищення, можна віднести зручність користування та інтуїтивність інтерфейсу, що дасть можливість швидко сприймати інформацію, яка надається.

Таблиця 1 — Вимоги до програмного забезпечення

|  |
| --- |
| Додаток має визначати швидкість транспортного засобу, щоб потім застосувати цю інформацію в інших розрахунках, та для виведення на екран користувачу. |
| Додаток має визначати швидкісний ліміт на ділянці дороги. |
| Додаток має повідомити водія, якщо відбулося перевищення швидкості |
| Додаток має давати можливість встановити бажану швидкість їзди, після чого додаток повідомлятиме, коли потрібно пригальмувати, щоб тримати оптимальну швидкість |
| Додаток має визначати положення водія на місцевості та повідомляти його про назву місця(населеного пункту), де зараз транспортний засіб. |
| При наближенні до населеного пункту додаток має повідомляти водія про це. |
| Додаток має розраховувати відстань до найближчої АЗС та повідомляти водія про це. |

# РОЗДІЛ 2 ТЕСТ-ПЛАН ПРОЄКТУ

**ПЛАН ТЕСТУВАННЯ ДЛЯ ПРОЄКТУ  
«МОБІЛЬНИЙ ДОДАТОК МОНІТОРИНГУ ПЕРЕВИЩЕННЯ ШВИДКОСТІ ТА ОПТИМАЛЬНОЇ ШВИДКОСТІ ТРАНСПОРТНИМ ЗАСОБОМ»**

**ST01-01**

**Підготував**

**Вознюк Назар Геннадійович**

## 2.1 Ідентифікатор плану тестування

ST01-01

## 2.2 Цілі тестування

План тестування мобільного додатку моніторингу перевищення швидкості та оптимальної швидкості транспортним засобом повинен забезпечити наступні цілі:

1. Систематизувати план тестування;
2. Визначити межі тестування;
3. Визначити середовище та інструменти тестування.

**2.3 Сфера застосування**

Цей план визначає тестування всієї системи обробки геолокації, яку отримує телефон. Також тестується відображення даних інтерфейсом після їх обробки.

## 2.4 Елементи тестування

В процесі тестування мають бути протестовані елементи наведені в списку нижче:

1. Отримання даних про геолокацію
2. Обробку даних геолокації
3. Відправку даних на користувацький інтерфейс
4. Перевірка роботи частин інтерфейсу

Відправка даних на користувацький інтерфейс передбачає як перевірку виклику функції встановлення значень, так і комплексне мануальне тестування на емуляторі, який симулює рух по координатах.

Тестування всіх пунктів спершу відбувається ізольовано. Пункти 1 та 2 можуть бути протестовані разом, з метою перевірки правильності взаємодії і передачі даних між ними.

## 2.5 Функціонал, що тестується

### 2.5.1 Отримання даних про геолокацію

1. Отримання даних з девайсу (Location Manager)
   1. Опис: Location Manager — клас, який отримує дані про геолокацію від девайсу, щоб передати їх на подальшу обробку. Єдиною функцією що тестується є trackLocation, яка постійно отримує дані для подальшої передачі.
   2. Пріоритет: Високий — отримання даних про геолокацію є ключовим елементом у роботі всього додатку, бо на основі цих даних відбуваються всі подальші процеси.

### 2.5.2 Обробка даних геолокації

1. gazerNewLocation
   1. Опис: gazerNewLocation — функція, яка отримує нові дані про геолокацію та запускає їх подальше опрацювання.
   2. Пріоритет: Високий — опрацювання даних про геолокацію є основою роботи додатку та забезпечує користувача всім необхідним функціоналом.
2. getNearestLocality
   1. Опис: getNearestLocality — функція, яка шукає дані про навколишні населені пункти в радіусі 10 кілометрів.
   2. Пріоритет: Середній — хоча відображення даних про навколишні населені пункти є важливою частиною додатку, воно не є його основною метою.
3. getNearestFuelStation
   1. Опис: getNearestFuelStation — функція, яка шукає дані про навколишні заправні станції в радіусі 1,5 кілометра.
   2. Пріоритет: Середній — хоча відображення даних про навколишні заправні станції є важливою частиною додатку, воно не є його основною метою.
4. getSpeedLimit
   1. Опис: getSpeedLimit — функція, яка шукає дані про швидкісний ліміт в тому місці, де знаходиться користувач.
   2. Пріоритет: Високий — відображення і знаходження правильного швидкісного обмеження, є критично важливим, для забезпечення правильного функціонування головних функцій системи.

### Відправка даних на користувацький інтерфейс

1. updateCurrentSpeed
   1. Опис: updateCurrentSpeed — функція, яка реалізує відображення користувачу, даних, про швидкість.
   2. Пріоритет: Високий — швидкість це головне значення, яке має бути відображене користувачу.
2. updateSpeedLimit
   1. Опис: updateSpeedLimit — функція, яка реалізує відображення користувачу швидкісного ліміту.
   2. Пріоритет: Високий — швидкісний ліміт це значення, на основі якого водій має приймати рішення, з якою швидкістю має рухатися користувач
3. updateNearestTown
   1. Опис: updateNearestTown — функція, яка реалізує відображення назви населеного пункту та дистанції до нього
   2. Пріоритет: Середній — це додаткова інформація, яка може знадобитися користувачу, але не є основною.
4. updateNearestFuel
   1. Опис: updateNearestFuel — функція, яка реалізує відображення назви населеного заправної та дистанції до нього
   2. Пріоритет: Середній — це додаткова інформація, яка може знадобитися користувачу, але не є основною.

### Перевірка роботи частин інтерфейсу

1. handleInput
   1. Опис: handleInput — функція реалізує збір даних з поля вводу, для встановлення власного швидкісного ліміту.
   2. Пріоритет: Високий — це є частиною головного функціоналу і одним з ключових параметрів в сповіщенні про перевищення швидкості, топу введення має працювати коректно.
2. setStopClick
   1. Опис: setStopClick — функція, яка прив’язує до кнопки «Зупинити» завершення роботи додатку, та зупинку сервісу збору геолокації.
   2. Пріоритет: Високий — користувач має мати можливість в будь-який момент зупинити збір даних про геолокацію додатком.

## Функціонал, що не тестується

До функціоналу що не тестується відносяться сторонні API, які надають інформацію відносно геолокації користувача. Також не тестуються бібліотеки, які реалізують з’єднання за HTTP (okhttp3), та обробляють отримані в форматі JSON дані (Gson). Це означає, що додаткових тестів для перевірки правильності роботи цих елементів, навіть попри їх помилки знайдені в процесі функціонального тестування.

## Підхід

Під час тестування, об’єкт що тестується потрібно віднести до типу, для якого створено систематизований підхід та використати зазначений метод тестування під час розробки тестів. Варто відзначити, що можуть бути елементи, для яких немає прописаного систематичного підходу.

### 2.7.1 Інструменти тестування

* JUnit4 — інструмент для створення unit та інтеграційних тестів;
* Espresso API — фреймворк для автоматизації тестування UI для андроїд додатків;
* Емулятор AndroidStudio — надає можливість створення фіктивних координат, що є надзвичайно зручним для мануального тестування.

### 2.7.2 Збір показників та метрики

* Успішним тестуванням вважається проходження всіх заданих unit та інтеграційних тестів;
* Відсутність критичних помилок при мануальному тестуванні на емуляторі теж входить до критерію успішного тестування.

### Конфігурації що будуть тестуватися

* Операційна система Android (sdk 27+)

### Підхід до тестування додатку

* + - 1. Встановлення бажаної швидкості

До перевірки входить:

* Перевірка того, що поле приймає лише додатнє числове значення;
* Дані з поля встановлюють значення в системі (при перевищенні встановленого значення відбувається сповіщення про перевищення швидкості)
  + - 1. Натискання кнопки зупинити

Перевірка передбачає зупинку додатку після натискання кнопки, та зупинку сервісу з збору даних (Можна тестувати мануально).

* + - 1. Функції обробки даних

Перевірка включає:

* Перевірку отримання геолокації;
* Перевірку передачі отриманих даних на подальшу обробку;
* Перевірку відображення отриманих після обробки даних.

## Критерії проходження/провалу для плану тестування

* За виникнення помилки в критичному тесті план тестування вважається проваленим, подальші тести не виконуються.
* Для проходження плану тестування всі тести мають бути виконані.
* Всі критичні помилки мають бути виправлені перед наступним тестуванням, або перед додаванням нового функціоналу.
* Будь яка помилка помітна користувачу, вважається критично, та підлягає усуненню, якщо вона не викликана бібліотекою. У разі, якщо таку проблему викликає бібліотека, варто розглянути альтернативні бібліотеки.

## Критерії призупинення та умови відновлення тестування

Якщо під час виправлення помилки буде внесено значні зміни до логіки роботи додатку, може виникнути потреба внести зміни до існуючих, або створити додаткові тести.

### 2.9.1 Критерії призупинення тестування

Критичні помилки, що блокують подальшу можливість коректної роботи додатку, є підставою для повного призупинення процесу тестування, до моменту усунення такої помилки.

Пов’язані тести — це група тестів, які можуть бути виконані лише, якщо всі попередні тести були виконані правильно.

### 2.9.2 Критерії відновлення тестування

Тестування може бути відновлене лише у випадку виправлення критичних помилок у разі наявності таких, та всіх помилок у пов’язаних тестах. До повного їх усунення проєкт не підлягає до подальшого тестування.

## Результати тестування

Головним результатом тестування має бути усунення критичних помилок, та помилок, які порушують бізнес-логіку проєкту. Оцінити результати тестування модульними тестами, можна в середовищі розробки. Оцінка мануального тестування проводиться на основі визначених критеріїв. Кількість пройдених тестів може слугувати показником якості роботи додатку.

## Тестові завдання

### 2.11.1 Залежності між тестами

Основною залежністю між тестами є зв’язок між сервісом отримання даних про геолокацію, та клієнтом, який опрацьовує ці дані. У разі, якщо дані про геолокацію отримуються та передаються неправильно, інтеграційне тестування обробки даних, є недоцільним, бо воно видасть завідомо неправильні результати.

### 2.11.2 Рівень навичок

Для написання тестів потрібне знання з створення unit-тестів, знання мови програмування, на якій створено проєкт, та розуміння принципу роботи додатку загалом. Для мануального тестування, потрібно мати базові навички роботи з емуляторами, та вміння налаштувати надсилання фіктивних даних про геолокацію, для перевірки додатку.

### Ключові етапи

* Розробка тестів отримання даних про геолокацію;
* Розробка тестів опрацювання даних про геолокацію;
* Розробка тестів оновлення інтерфейсу;
* Мануальне тестування, для перевірки цілісності системи, та нефункціональних вимог.

## 2.12 Потреби середовища

### 2.12.1 Джерела інформації

Для додатку існує два головних джерела інформації. Перше це дані про геолокацію, друге дані що повертають API на основі геолокації. Через це тестування може проводитися як на фіктивних даних про геолокацію, або фіктивних даних, що повертає сервер, так і на реальних даних.

### 2.12.2 Безпека

Система не зберігає жодних даних про користувача і лише використовує дані про його геолокацію, для пошуку інформації про місцевість з застосуванням третьої сторони. Третій стороні ніяких даних крім геолокації девайсу, на якому встановлений додаток не передається. Також користувач може в будь-який момент вимкнути додаток і призупинити збір геолокації.

## Відповідальності

Єдиний учасник проєкту Вознюк Назар Геннадійович. Виконує всі функції розробки і тестування самостійно. Несе повну відповідальність за якість кінцевого продукту.

## Ризики

### 2.13.1 Зміна дат

Через можливість критичних помилок, усунення яких займе багато часу, або ускладнення ситуації з електропостачанням, можливі зсуви у графіку розробки та тестування з метою завершення роботи у повному об’ємі.

### 2.13.2 Надто низька толерантність до помилок

На момент написання плану проєкт має нульову толерантність навіть до найменших помилок, які не несуть ніякої загрози правильності роботи системи, або враженням користувача. Проте це може створити надто велике навантаження на розробку, через що цей поріг може бути піднятий, для функціоналу, який не має критичного значення.

## Затвердження

Керівник курсової роботи виступає в ролі незалежного експерта, з метою оцінки якості і оцінки проєкту вимогам з точки зору користувача. У разі знаходження ним помилок, або наявності зауважень, які не дають затвердити проєкт, він буде повернений на доопрацювання, після чого процес затвердження повториться.

# РОЗДІЛ 3 РОЗРОБКА БАГ РЕПОРТУ

Створення баг репортів є невід’ємною частиною розробки будь якого програмного забезпечення. Воно дає можливість класифікувати проблеми за рівнями важливості та критичності для проєкту, та правильно розподілити сили на їх виправлення.

Баги ж в свою чергу це дефекти в програмному застосунку, які можуть призводити до відмови, або збою в програмному забезпеченні. При описі багу потрібно розписати такі атрибути як: тема, детальний опис, кроки для відтворення, результати, пріоритетність, серйозність, статус та оточення. Також до кожного багу варто додавати вкладення, такі як скріншоти, відео або лог-файли. Це спрощує розуміння суті проблеми та її відтворення.

## 3.1 Список знайдених багів

1. Програма неочікувано припиняє роботу у разі відсутності GPS
2. Запити не надсилаються до API
3. Сповіщення не припиняють надсилатися, після першого оповіщення.
4. Інтерфейс користувача не оновлюється, якщо пробувати це робити з фонового процесу
5. Неможливість передати дані з фонового процесу до головного
6. Неправильне отримання інформації GPS

## 3.2 Складений баг репорт

### 3.2.1 Тема багу

Сповіщення не припиняють надсилатися, після першого оповіщення.

### 3.2.2 Детальний опис багу

Коли користувач перевищив швидкість (загальну або власну), програма має сповістити його про це один раз і сповістити його знову лише у випадку, якщо він сповільнився до допустимої відмітки і перевищив швидкість знову. Проте програма не припиняє сповіщати користувача про перевищення, після того як він перевищив швидкість, аж до моменту його сповільнення до нормальної швидкості.

### 3.2.3 Кроки до відтворення

Для простоти перевірки і відтворення потрібно встановити бажану швидкість на дуже мале значення, наприклад один кілометр на годину, після цього запустити симуляцію маршруту в налаштуваннях емулятора. Після виконання цих кроків, можемо побачити постійні повідомлення. Якщо це буде продовжуватися досить довго є ризик того, що Android автоматично почне приглушувати повідомлення через спам.

### 3.2.4 Результати

Очікувалося 1 сповіщення про перевищення, яке знову з’явиться за умови, що користувач знову перевищить швидкість. Отриманий результат — постійний потік сповіщень, до моменту повернення в межі швидкісного обмеження, які можуть відволікати користувача.

### 3.2.5 Пріоритетність, серйозність, статус та оточення

Пріоритетність цього багу середня, бо здебільшого він не впливає на роботу програмного продукту.

Серйозність незнача, бо баг не порушує роботу, але може заважати користувачу.

Стутус — виправлено.

Оточення — операційна система Android (SDK 35)

## 3.3 Розбір рішення багу

### 3.3.1 Розбір причини проблеми

Сповіщення з’являються з тою ж самою частотою, з якою оновлюється швидкість руху. Це пов’язано з тим, що функція, яка перевіряє наявність перевищення швидкості та сповіщає користувача про це (compareSpeed), викликається щоразу, коли отримується нова інформація з GPS. Це можна побачити з коду, який наведений на рисунку 3.1. Функція в цьому випадку викликається всередині іншої функції gazerNewLocation, яка є колбеком, який викликається при оновленні даних GPS.

Отже розуміючи причину багу можна перейти до розгляду можливих рішень.



Рисунок 3.1 Код функції gazerNewLocation

### 3.2.2 Огляд можливих рішень

В загальному проблема є досить тривіальною, проте теоретично вона може бути вирішена декількома способами.

Перший спосіб передбачає виклик функції compareSpeed не щоразу, а з певною періодичністю, цього можна досягти додавши до класу, який містить функцію gazerNewLocatio змінну, яка буде підраховувати кількість оновлень між порівняннями.

Другий спосіб передбачає виділення цієї перевірки на окремий потік, де використовуючи функцію sleep та цикл, який працює впродовж роботи програми, щоб робити перерви між перевірками. Цей спосіб також потребує логіки передавання даних з потоку на якому виконується функція gazerNewLocatio до потоку з функцією compareSpeed.

Третій спосіб передбачає використання булевої змінної, яка імітує собою стан. Ця змінна буде відстежувати чи було вже показане сповіщення про перевищення швидкості. Якщо повідомлення вже було показане, то відбудеться лише швидкісного обмеження та швидкості руху без додаткового сповіщення.

### 3.2.3 Результат виправлення

Для виправлення цього багу було обрано третій спосіб описаний у попередньому пункті. Причиною такого вибору було поєднання простоти реалізації, та відсутність потреби зменшувати кількість перевірок швидкості. Результуючу логіку можна побачити на рисунку 3.2.



Рисунок 3.2 — Фінальний варіант функції порівняння швидкості

# Розділ 4 ТЕСТУВАННЯ ПРОЄКТУ

Важливість тестування для будь-якого проєкту важко переоцінити. З його допомогою можна швидко та оперативно виявити та виправити проблеми, які виникають в програмі під час її написання. Методів тестування існує досить велика кількість, проте для цього проєкту було вирішено використовувати unit-тестування з додаванням інтеграційного тестування, а також мануальне тестування. Особливістю unit-тестування в цьому випадку також є те, що використовується такий підвид, як інструментальне тестування. За його допомогою тести виконуються напряму на пристрої Android, що дає змогу використовувати не фіктивні елементи, а реальні підсистеми операційної системи.

## 4.1 Середовище для тестування

Як середовище розробки тестів було обрано IDE AndroidStudio, воно надає підтримку виконання unit та інтеграційних тестів, а також має вбудований емулятор. Віг дає змогу тестувати елементи напряму на девайсі, що дуже корисно для використання інтеграційних тестів. Ще одним плюсом цього середовища розробки є зручний інтерфейс, який дає з легкістю зрозуміти, які тести були виконані, який час зайняло їх виконання, та в чому причина провалу, якщо таке сталось. Інтерфейс можна побачити на рисунку 4.1.

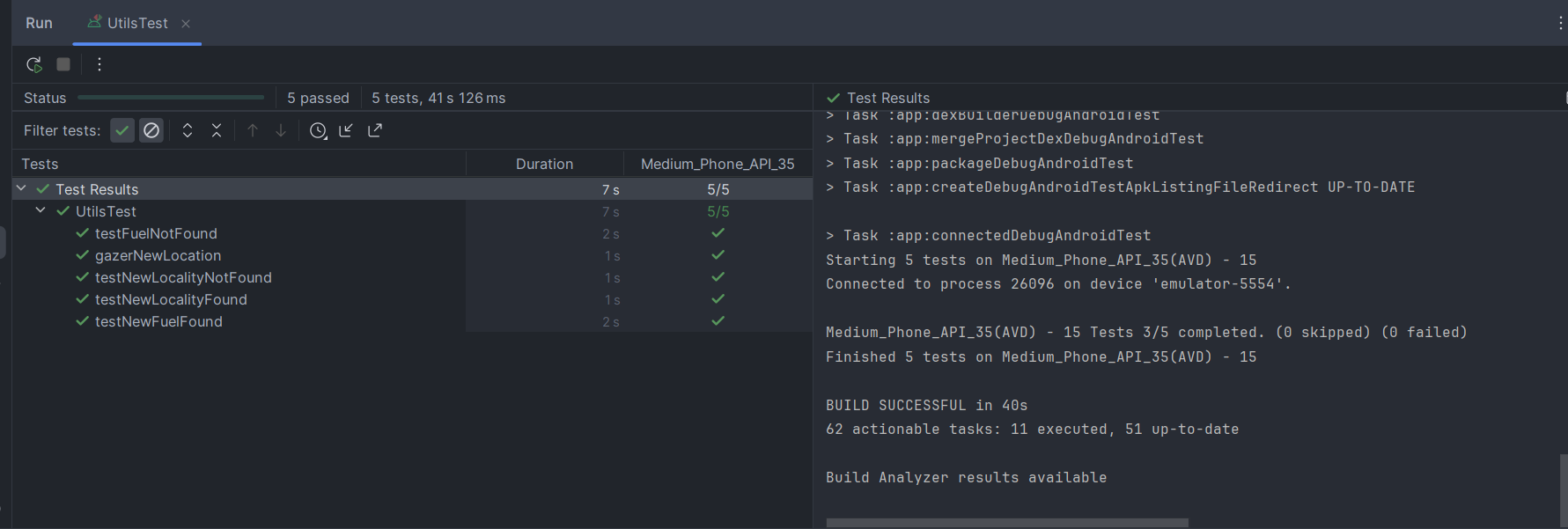


Рисунок 4.1 — Інтерфейс з виконаними тестами

Це ж середовище розробки використовувалося і для виконання мануального тестування. Згаданий раніше емулятор, надає інструменти, які дають можливість спростити цей процес, і позбутися потреби тестувати програму одразу на справжньому девайсі.

## 4.2 Unit-тестування

Unit-тестування це один з різновидів тестування програмного забезпечення, головною метою якого є виконання та перевірка невеликих частин коду, таких як клас або функція, щоб впевнитися в правильності їх роботи у відриві від решти системи. Головною метою цього тестування не є виявлення вже існуючих багів, а недопущення появи нових.

### 4.2.1 Переваги та недоліки unit-тестування

До переваг unit-тестування можна віднести наступні пункти:

* Покращення якості коду — це позначається в тому, що для написання unit-тестів потрібно наперед продумувати архітектуру, яка буде підлягати тестуванню.
* Раннє виявлення помилок — таке тестування, тісно пов’язане з функціоналом, який воно тестує, через це у разі винекнення помилки, одразу можна ідентифікувати її причину. Також unit-тести автоматизують перевірку, що спрощує весь процес тестування, під час активної розробки.
* Ефективність написання коду — це можна пояснити тим, що проблеми, які виникають в коді виправляються одразу перед написанням подальшого коду, бо в разі ігнорування помилки, час на її виправлення і пошук в майбутньому може вирости в рази.
* Простота внесення змін — оскільки код, який пишеться з врахуванням зручності подальшого тестування, являє собою невеличкі функції, які реалізують собою 1 або декілька простих дій, то до нього легко вносити зміни, працюючи лише з потрібним блоком коду і не переймаючись за інші частини програми.

Проте, звичайно, у unit-тестування наявні і недоліки. Вони можуть бути описані наступними пунктами:

* Розробка забирає багато часу — щоб створити правильний набір unit-тестів потрібно провести дослідження, що потрібно тестувати, які об’єкти та допоміжні функції потрібно використати та як правильно застосувати той чи інший тест, для досягнення максимальної ефективності. Одним з головних методів такого дослідження є метод спроб і помилок, через що написання навіть одного продуманого unit-тесту може займати цілий день.
* Потреба в постійній підтримці — в процесі розробки проєкт постійно зазнає змін, а через це і його логіка. Це вимагає постійного доопрацювання, щоб покрити усі можливі сценарії, та відповідати умовам, для яких створюється додаток. Відповідно це займає додатковий час, адже процес описаний в попередньому пункті повторюється.
* Unit-тести важно застосовувати до користувацьких інтерфейсів та комплексних систем — це спричинено складністю роботи таких систем. Вони задіюють багато різних підсистем та класів, їх логіка може варіюватися від просто складної до взагалі незрозумілої, для розробника з базовим рівнем знань. Також unit-тести покликані виявляти помилки в індивідуальних компонентах, що робить їх неефективними для цього завдання.

### 4.2.2 Розробка unit-тестів

Розглянувши елементи з яких складається система було створено чотири тести, які покривають перевірку функціоналу відображення на екрані. Кожен з цих тестів перевіряє свою функцію. Вони виконуються не на емуляторі, а на комп’ютері. Додатково перед запуском цих тестів потрібно створити фіктивні елементи, які будуть імітувати поведінку реальних полів, які мають бути встановлені. Для цього використовується бібліотека MockK. Далі пояснено кожен з цих тестів:

* Перевірка оновлення поточної швидкості:
  + Тестується функція updateCurrentSpeed;
  + Очікуваний результат — текст зміниться на той, який передається до функції;
  + Умова проходження: підставний елемент, який імітує TextView зафіксує виклик функції встановлення тексту з потрібним значенням в текстовому полі;
  + Критичність проходження тесту: Висока — користувач не може переглянути швидкість у разі провалу.
* Перевірка оновлення швидкісного обмеження:
  + Тестується функція updateSpeedLimit;
  + Очікуваний результат — текст зміниться на той, який передається до функції;
  + Умова проходження: підставний елемент, який імітує TextView зафіксує виклик функції встановлення тексту з потрібним значенням в текстовому полі;
  + Критичність проходження тесту: Висока — користувач не може переглянути швидкісне обмеження на частині дороги у разі провалу, що може призвести до поганих наслідків.
* Перевірка оновлення найближчого населеного пункту:
  + Тестується функція updateNearestTown;
  + Очікуваний результат — текст зміниться на той, який передається до функції;
  + Умова проходження: обидва підставні поля, зафіксують виклик відповідних функцій встановлення текту з потрібними значеннями (тими, які були передані як параметри);
  + Критичність проходження тесту: Низька — користувач не може переглянути інформацію про найближчий населений пункт, проте це не впливає на головну функцію додатку і не є критично важливим для водія.
* Перевірка оновлення найближчої заправної станції:
  + Тестується функція updateNearestFuel;
  + Очікуваний результат — текст зміниться на той, який передається до функції;
  + Умова проходження: обидва підставні поля, зафіксують виклик відповідних функцій встановлення текту з потрібними значеннями (тими, які були передані як параметри);
  + Критичність проходження тесту: Низька — користувач не може переглянути інформацію про найближчу заправну станцію, проте це не впливає на головну функцію додатку і не є критично важливим для водія.

## 4.3 Інтеграційне тестування

Інтеграційне тестування також є дуже важливою частиною будь-якого проєкту. Воно передбачає поєднання різних модулів програми між собою, з метою перевірки правильності взаємодії їх один з одним. Інтеграційне тестування варто проводити після unit-тестування та перед фінальною верифікацією проєкту. Для проведення тестів попередньо потрібно надати доступ до дозволів. Потім для перевірки створюється підставна геолокація та запускається програма, це проводиться до кожного тесту.

### 4.3.1 Написання інтеграційних тестів

* Перевірка отримання нової геолокації:
  + Тестується функція gazerNewLocation
  + Очікуваний результат — текст швидкості зміниться на ту, яка була заявлена в фіктивній локації
  + Умова проходження: порівняння значення, яке було встановлено та значення, яке передавалося має повернути позитивний результат
  + Критичність проходження тесту: Висока — якщо значення швидкості в цьому тесті було встановлено неправильно, то це означає, що дані геолокації отримуються та передаються неправильно, що повністю порушує вимоги до роботи додатку.
* Перевірка знаходження нового населеного пункту:
  + Тестується функція getNearestLocality
  + Очікуваний результат — поле назви населеного пункту та дистанції до нього зміниться на значення, які знаходить функція
  + Умова проходження: порівняння значення, яке було встановлено та значення, яке було знайдено відповідно до координат, які були виставлені для фіктивної GPS локації на картах мають співпадати.
  + Критичність проходження тесту: Середня — якщо тест провалився, то це може позначати як помилку в отриманні GPS локації, так і про можливу помилку з сторони стороннього API. Пріоритет не високий, бо правильність визначення цієї інформації не є критичною для роботи додатку.
* Перевірка встановлення значень за замовчуванням у разі відсутності знайдених даних про найближчі населені пункти:
  + Тестується функція getNearestLocality
  + Очікуваний результат — поля назви населеного пункту та відстані до нього заповняться значеннями за замовчуванням
  + Умова проходження: в поля назви населеного пункту та відстані до нього були встановлені значення за замовчуванням «Не знайдено» та «---» відповідно.
  + Критичність проходження тесту: Висока — якщо значення поля не були виставленні як значення за замовчуванням, це свідчить про помилку в логіці опрацювання результатів.
* Перевірка отримання інформації про нову заправну станцію:
  + Тестується функція getNearestFuelStation
  + Очікуваний результат — поля назви заправної станції та дистанції до неї заповняться значеннями
  + Умова проходження: встановлені назва заправної станції та дистанція до неї, співпадають з значеннями, які були знайдені на картах скориставшись тестовими координатами.
  + Критичність проходження тесту: Середня — якщо тест провалився, то це може позначати як помилку в отриманні GPS локації, так і про можливу помилку з сторони стороннього API. Пріоритет не високий, бо правильність визначення цієї інформації не є критичною для роботи додатку.
* Перевірка встановлення значень за замовчуванням у разі відсутності знайдених даних про найближчі заправні станції:
  + Тестується функція getNearestFuelStation
  + Очікуваний результат — поля назви заправної станції та відстані до неї заповняться значеннями за замовчуванням
  + Умова проходження: в поля назви населеного пункту та відстані до нього були встановлені значення за замовчуванням «Не знайдено» та «---» відповідно.
  + Критичність проходження тесту: Висока — якщо значення поля не були виставленні як значення за замовчуванням, це свідчить про помилку в логіці опрацювання результатів.
* Перевірка отримання інформації про швидкісний ліміт:
  + Тестується функція getSpeedLimit
  + Очікуваний результат — поле обмеження швидкості заповниться значенням, яке було знайдено
  + Умова проходження: встановлений швидкісний ліміт має співпадати з швидкісним лімітом, який був знайдений за тестовими координатами на картах.
  + Критичність проходження тесту: Висока — якщо тест провалився, це свідчить або про помилковість отримування координат, або про неефективність API. Ці дані є критично важливими для роботи додатку.
* Перевірка встановлення значень за замовчуванням у разі відсутності знайдених даних про обмеження швидкості:
  + Тестується функція getSpeedLimit
  + Очікуваний результат — поле обмеження швидкості заповниться значенням за замовчуванням
  + Умова проходження: встановлений швидкісний ліміт має відповідати значенню за замовчуванням «Н/З»
  + Критичність проходження тесту: Висока — якщо тест провалився, це свідчить або про помилковість отримування координат, або про проблеми в внутрішній логіці функції опрацювання результатів.
* Перевірка зчитування даних:
  + Тестується функція handleInput
  + Очікуваний результат — поле зчитає значення і передасть його до об’єкту класу Utils
  + Умова проходження: значення, яке було введене до поля вводу, має співпадати з значенням, яке зберігається у полі wantedSpeed класу Utils
  + Критичність проходження тесту: Висока — якщо тест провалився, це свідчить про помилку в логіці зчитування значення, або в логіці опрацювання введеного значення.
* Перевірка натискання кнопки «Зупинити»
  + Тестується функція setStopClick
  + Очікуваний результат — додаток закриється і припиниться збір даних GPS
  + Умова проходження: після натискання кнопки, головна активність має перейти до стану isFinishing
  + Критичність проходження тесту: Висока — якщо тест провалився, це свідчить про помилку в завершенні програми або налаштуваннях кнопки, що може відібрати в користувача можливість зупинити відстеження GPS за його бажанням.

## 4.4 Результати unit та інтеграційного тестування

Результати тестування можна побачити на рисунках 4.2, 4.3 та 4.4, які наведені нижче.

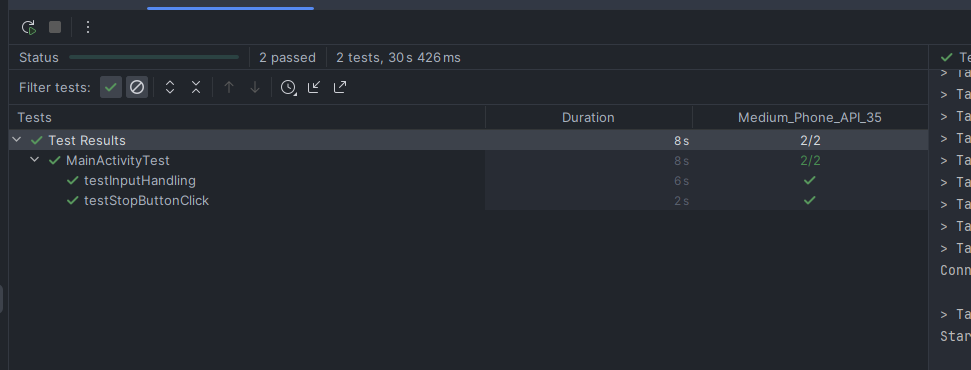


Рисунок 4.2 — Результати тестування введення і натискання кнопки «Зупинити»

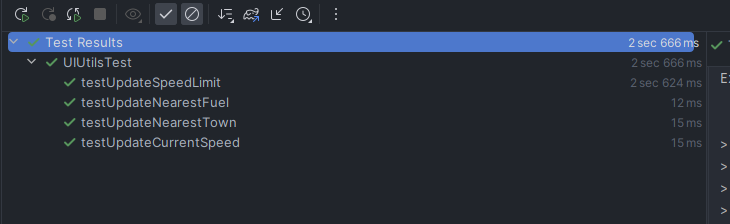


Рисунок 4.3 — Результати виконання unit-тестів оновлення даних

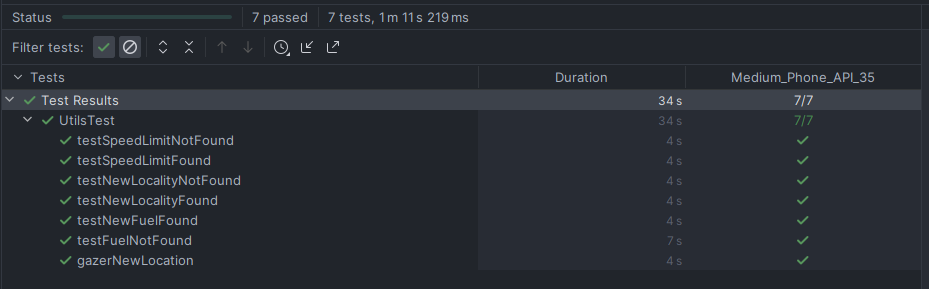


Рисунок 4.4 — Результати тестування функцій обробки даних геолокації

## 4.5 Мануальне тестування

Цей підхід до тестування програмного забезпечення передбачає виконання тестувальником всіх етапів тестування вручну, не використовуючи автоматизованих засобів. Ціль мануального тестування, це пошук багів, дефектів та інших проблем, які виникають у програмному забезпеченні. Воно знаходить проблеми пов’язані з функціональністю, швидкодією та зручністю використання.

У створенні цього проєкту мануальне тестування використовувалося для перевірки правильності роботи системи сповіщень та додаткового тестування всіх описаних вище чатин. Для досягнення цього використовувалися можливості підміни геолокації, які надаються емулятором AndroidStudio.

Для тестування перевірки правильності визначення даних створюється набір точок, які є статичними і імітують перебування на конкретній точці. Набір точок для визначення різної інформації можна побачити на рисунку 4.5.

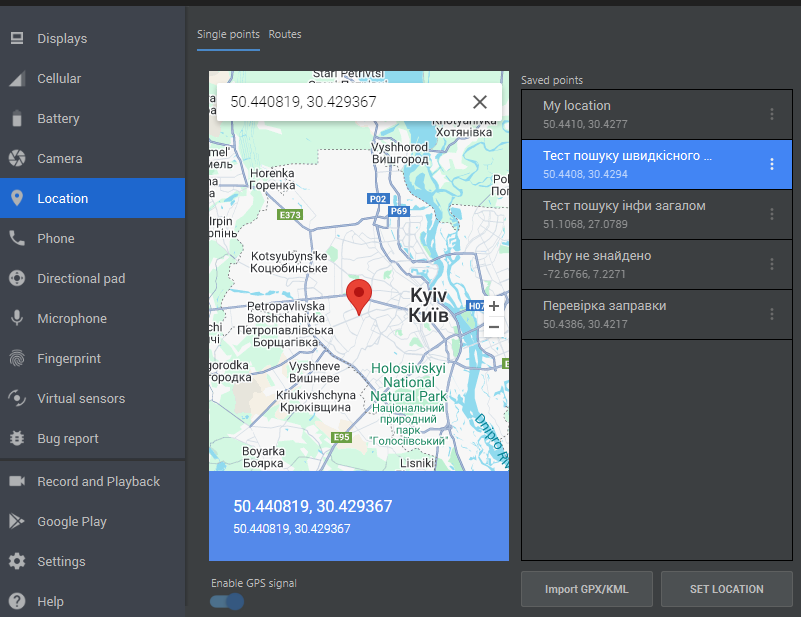


Рисунок 4.5 — Точки для імітації геолокації

Для перевірки сповіщення про перевищення швидкості використовувався функціонал прокладення маршрутів. Емулятор імітує переміщення з різною швидкістю, що досить зручно для перевірки перевищення швидкісного ліміту. Єдиним недоліком цієї системи є неможливість встановити власну постійну швидкість. Прокладений тестовий маршрут можна побачити на рисунку 4.6.

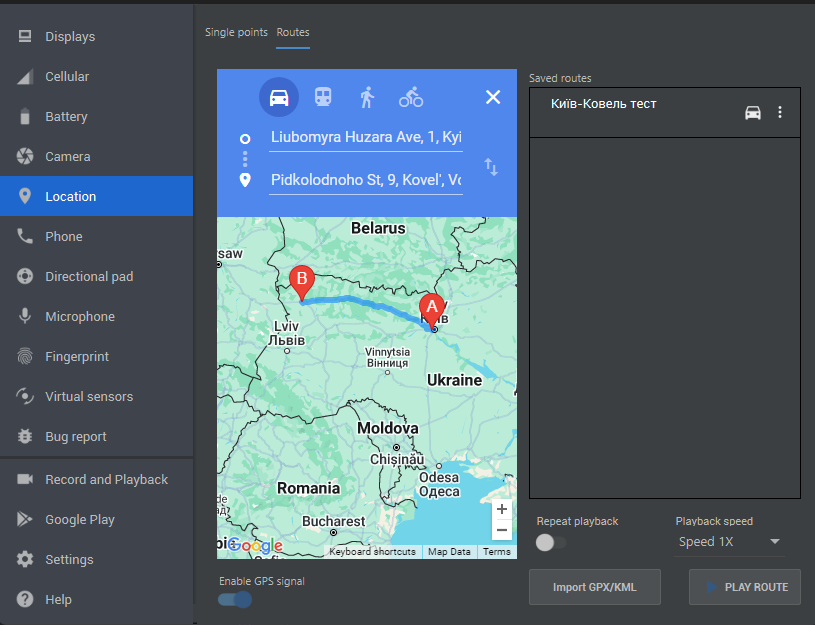


Рисунок 4.6 — Тестовий маршрут налаштований в емуляторі

# ВИСНОВОК

В результаті виконання курсової роботи було створено мобільний додаток, який визначає швидкість руху користувача, та допомагає йому дотримуватися оптимальної швидкості. Додаток надсилає користувачу сповіщення, щоб повідомити його про наближення до населеного пункту, до заправної станції, або про перевищення швидкості. Користувачу надається можливість задавати власну бажану швидкість руху, і якщо вона не перевищує швидкість встановлену на частині дороги, де перебуває користувач, додаток буде моніторити і повідомляти користувача про перевищення бажаної швидкості. Для зручності користування було створено мінімалістичний візуальний інтерфейс.

Ключовим етапом в розробці програмного забезпечення стало його тестування. Щоб впевнитися в правильності роботи всіх компонентів додатку, використовувалися різні програмні засоби. До таких засобів увійшли інструментальні інтеграційні тести, та unit-тести, які допомагали відстежувати відсутність помилок впродовж розробки. Також було проведене повне мануальне тестування, яке включало симуляцію руху, з метою перевірки працездатності системи, та оцінки зручності користування.

Для полегшення процесу розробки тестів, було створено розгорнутий тест-план, який включає в себе всю інформацію про елементи, які мають бути протестовані, та як їх потрібно тестувати.

У разі виникнення багів, навіть попри відсутність використання спеціального програмного забезпечення для їх відстеження, вони занотовувалися та підлягали негайному виправленню.

У підсумку розроблений мобільний додаток, став зручним і функціональним інструментом, який може допомогти користувачам дотримуватися правил дорожнього руху і уникати небезпечних ситуацій. Подальша ж розробка передбачає покращення структури, та поглиблення знань з розробки Android додатків, щоб розширити функціонал, та виправити всі помилки, які могли залишитися непоміченими.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Build instrumented tests  |  Android Developers. *Android Developers*. URL: <https://developer.android.com/training/testing/instrumented-tests>.
2. Build local unit tests  |  Android Developers. *Android Developers*. URL: <https://developer.android.com/training/testing/local-tests>.
3. Develop a UI with Views  |  Android Studio  |  Android Developers. *Android Developers*. URL: <https://developer.android.com/studio/write/layout-editor>.
4. Espresso basics  |  Android Developers. *Android Developers*. URL: <https://developer.android.com/training/testing/espresso/basics>.
5. Kotlin Docs | Kotlin. *Kotlin Help*. URL: <https://kotlinlang.org/docs/home.html>.
6. MockK. *MockK*. URL: <https://mockk.io/>.
7. Services overview  |  Background work  |  Android Developers. *Android Developers*. URL: <https://developer.android.com/develop/background-work/services> (date of access: 04.01.2025).
8. What Are Advantages and Disadvantages of Unit Testing? - MethodPoet. *MethodPoet*. URL: <https://methodpoet.com/unit-testing-advantages-and-disadvantages/>.

**ДОДАТОК 1**

Діаграма класів додатку

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 1

Київ – 2024



**ДОДАТОК 2**

Unit-тести

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 3

Київ – 2024

UIUtilsTest

package com.example.myapplication

import android.app.Activity

import android.widget.TextView

import org.junit.Before

import io.mockk.\*

import org.junit.Test

class UIUtilsTest {

private lateinit var mockActivity: Activity

private lateinit var uiUtils: UIUtils

@Before

fun setUp() {

mockActivity = mockk(relaxed = true)

uiUtils = UIUtils(mockActivity)

}

@Test

fun testUpdateCurrentSpeed() {

val testTextView = mockk<TextView>(relaxed = true)

every { mockActivity.findViewById<TextView>(R.id.speedShow) } returns testTextView

uiUtils.updateCurrentSpeed("50")

verify { testTextView.text = "50" }

}

@Test

fun testUpdateSpeedLimit() {

val testTextView = mockk<TextView>(relaxed = true)

every { mockActivity.findViewById<TextView>(R.id.speedLimitShow) } returns testTextView

uiUtils.updateSpeedLimit("50")

verify { testTextView.text = "50" }

}

@Test

fun testUpdateNearestTown() {

val testTownField = mockk<TextView>(relaxed = true)

val testDistanceField = mockk<TextView>(relaxed = true)

every { mockActivity.findViewById<TextView>(R.id.townName) } returns testTownField

every { mockActivity.findViewById<TextView>(R.id.townDistance) } returns testDistanceField

uiUtils.updateNearestTown("Київ", "500")

verify { testTownField.text = "Київ" }

verify { testDistanceField.text = "500" }

}

@Test

fun testUpdateNearestFuel() {

val testFuelStationField = mockk<TextView>(relaxed = true)

val testDistanceField = mockk<TextView>(relaxed = true)

every { mockActivity.findViewById<TextView>(R.id.fuelName) } returns testFuelStationField

every { mockActivity.findViewById<TextView>(R.id.fuelDistance) } returns testDistanceField

uiUtils.updateNearestFuel("UPG", "500")

verify { testFuelStationField.text = "UPG" }

verify { testDistanceField.text = "500" }

}

}

**ДОДАТОК 3**

Інтеграційні тести

**НТУУ «КПІ» ІАТЕ ІПЗЕ**

Листів 10

Київ – 2024

MainActivityTest

package com.example.myapplication

import android.Manifest

import androidx.test.core.app.ActivityScenario

import androidx.test.ext.junit.runners.AndroidJUnit4

import androidx.test.rule.GrantPermissionRule

import androidx.test.espresso.Espresso.onView

import androidx.test.espresso.action.ViewActions.\*

import androidx.test.espresso.assertion.ViewAssertions.matches

import androidx.test.espresso.matcher.ViewMatchers.\*

import org.junit.Rule

import org.junit.Test

import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)

class MainActivityTest {

@get:Rule

val grantPermissionRule: GrantPermissionRule = GrantPermissionRule.grant(

Manifest.permission.ACCESS\_FINE\_LOCATION,

Manifest.permission.ACCESS\_COARSE\_LOCATION,

Manifest.permission.INTERNET,

Manifest.permission.ACCESS\_NETWORK\_STATE,

Manifest.permission.POST\_NOTIFICATIONS

)

@Test

fun testInputHandling() {

val scenario = ActivityScenario.launch(MainActivity::class.java)

onView(withId(R.id.ownSpeedInput)).perform(

clearText(),

typeText("50.0"),

pressImeActionButton()

)

var storedValue: String? = null

scenario.onActivity {

activity ->

storedValue = String.format("%.1f", activity.utils.wantedSpeed)

}

onView(withId(R.id.ownSpeedInput)).check(matches(withText(storedValue)))

}

@Test

fun testStopButtonClick() {

val scenario = ActivityScenario.launch(MainActivity::class.java)

onView(withId(R.id.stopButton)).perform(click())

scenario.onActivity { activity ->

assert(activity.isFinishing)

}

}

}

UtilsTest

package com.example.myapplication

import android.Manifest

import android.location.Location

import android.os.SystemClock

import androidx.test.core.app.ActivityScenario

import androidx.test.ext.junit.runners.AndroidJUnit4

import androidx.test.rule.GrantPermissionRule

import androidx.test.espresso.Espresso.onView

import androidx.test.espresso.assertion.ViewAssertions.matches

import androidx.test.espresso.matcher.ViewMatchers.\*

import androidx.test.platform.app.InstrumentationRegistry

import com.google.android.gms.location.FusedLocationProviderClient

import com.google.android.gms.location.LocationServices

import org.junit.Rule

import org.junit.Test

import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)

class UtilsTest {

@get:Rule

val grantPermissionRule: GrantPermissionRule = GrantPermissionRule.grant(

Manifest.permission.ACCESS\_FINE\_LOCATION,

Manifest.permission.ACCESS\_COARSE\_LOCATION,

Manifest.permission.INTERNET,

Manifest.permission.ACCESS\_NETWORK\_STATE,

Manifest.permission.POST\_NOTIFICATIONS,

)

@Test

fun gazerNewLocation() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

val mockLocation = Location("test\_provider").apply {

latitude = 37.7749

longitude = -122.4194

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 10.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.speedShow)).check(matches(withText("36.0")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

@Test

fun testNewLocalityFound() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

val mockLocation = Location("test\_provider").apply {

latitude = 50.4386

longitude = 30.4217

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 10.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.townName)).check(matches(withText("Софіївська Борщагівка")))

onView(withId(R.id.townDistance)).check(matches(withText("4.95")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

@Test

fun testNewLocalityNotFound() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

// Точка десь в антарктиці

val mockLocation = Location("test\_provider").apply {

latitude = -72.6766

longitude = 7.2271

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 0.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.townName)).check(matches(withText("Не знайдено")))

onView(withId(R.id.townDistance)).check(matches(withText("---")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

@Test

fun testNewFuelFound() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

// Заправка Amic energy, на відстані 68 (приблизно 70) метрів

// Дані для перевірки взято з Google Maps

val mockLocation = Location("test\_provider").apply {

latitude = 50.4386

longitude = 30.4217

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 10.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.fuelName)).check(matches(withText("Amic energy")))

onView(withId(R.id.fuelDistance)).check(matches(withText("0.07")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

@Test

fun testFuelNotFound() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

// Точка десь в антарктиці

val mockLocation = Location("test\_provider").apply {

latitude = -72.6766

longitude = 7.2271

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 0.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.fuelName)).check(matches(withText("Не знайдено")))

onView(withId(R.id.fuelDistance)).check(matches(withText("---")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

@Test

fun testSpeedLimitFound() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

val mockLocation = Location("test\_provider").apply {

latitude = 50.4386

longitude = 30.4217

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 10.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.speedLimitShow)).check(matches(withText("50.0")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

@Test

fun testSpeedLimitNotFound() {

ActivityScenario.launch(MainActivity::class.java)

val context = InstrumentationRegistry.getInstrumentation().targetContext

val fusedLocationProviderClient: FusedLocationProviderClient =

LocationServices.getFusedLocationProviderClient(context)

// Точка десь в антарктиці

val mockLocation = Location("test\_provider").apply {

latitude = -72.6766

longitude = 7.2271

altitude = 10.0

accuracy = 3.0f

time = System.currentTimeMillis()

elapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos()

speed = 0.0f

}

fusedLocationProviderClient.setMockMode(true).addOnSuccessListener {

fusedLocationProviderClient.setMockLocation(mockLocation).addOnSuccessListener {

Thread.sleep(4000)

onView(withId(R.id.speedLimitShow)).check(matches(withText("Н/З")))

}

}

fusedLocationProviderClient.setMockMode(false)

}

}