Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА IOS ДОДАТКУ ДЛЯ ОПОВІЩЕНЬ ПРО НАБЛИЖЕННЯ ДО ШВИДКІСНОГО ЛІМІТУ

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Владислав КИРИЛЕНКО
(прізвище та ініціали)

Керівник: асистент каф. КН
Єгор СІЛАГІН
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: Ph.D., ст. викл. каф. САІТ
Ольга ВОЙЦЕХОВСЬКА
(прізвище та ініціали)

«05» червня 2025 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Андрій ЯРОВИЙ
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖЕНО
Завідувач кафедри КН
проф., д.т.н. Андрій ЯРОВИЙ
«05» травня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Кириленко Владиславу Віталійовичу

1. Тема роботи: Розробка iOS додатку для оповіщень про наближення до швидкісного ліміту
керівник роботи: Сілагін Єгор Олексійович, асистент кафедри КН
затверджені наказом вищого навчального закладу "20" 03 2025 року №97
2. Термін подання студентом роботи 30 травня 2025 р.
3. Вихідні дані до роботи : об'єктно-документний метод; дані про користувача; геолокація користувача, відомості про швидкісне обмеження для теперішнього місця перебування за наявності, показник швидкості; аудіозаписи зі звуком для сповіщення; інтерфейс програмного продукту — графічний інтерфейс користувача
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): визначення об'єкту, предмета, мети та задач подальшого дослідження; аналіз предметної області, існуючих методів, алгоритмів та інструментів для спостереження за швидкістю пересування на дорозі та оповіщення про неї; проєктування додатку для спостереження за швидкістю та оповіщення при наближенні до ліміту; програмна реалізація для спостереження за швидкістю та оповіщення при наближенні до ліміту
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): загальна архітектура додатку для оповіщень про наближення до швидкісного ліміту; UML-діаграма класів системи для оповіщень про наближення до швидкісного ліміту; алгоритми роботи основних модулів системи (графічний інтерфейс, менеджер геолокації, менеджер звукових сповіщень); об'єктно-орієнтована модель даних предметної області "Оповіщення про наближення до швидкісного ліміту"; Зовнішній вигляд роботи програмного модулю.

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Єгор СІЛАГІН асистент каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Визначення об'єкта, предмета, мети та задачі подальшого дослідження	05.05.25-07.05.25	
2	Аналіз предметної області, існуючих методів, алгоритмів та інструментів для спостереження за швидкістю руху та швидкісних лімітів;	08.05.25-11.05.25	
3	Проектування додатку для сповіщення про перевищення швидкісного ліміту	12.05.25-15.05.25	
4	Програмна реалізація додатку сповіщення про перевищення швидкості	16.05.25-01.06.25	
5	Оформлення пояснювальної записки	02.06.25-04.06.25	

Студент _____ Владислав КИРИЛЕНК
(підпис) (прізвище та ініціали)

Керівник роботи _____ Єгор СІЛАГІН
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 61 сторінки формату А4, на яких є 17 рисунків та 1 таблиця, список використаних джерел містить 21 найменування.

Ця робота присвячена розробці мобільного додатку для iOS, який реалізує функцію сповіщення водія про наближення до встановленого обмеження швидкості. Головною метою роботи є розширення функціональних можливостей додатків для оповіщення про наближення до швидкісного ліміту.

Під час роботи було проаналізовано сучасні підходи до контролю швидкості транспортних засобів, зазначено особливості впровадження таких систем у мобільних додатках, а також проведено порівняльний аналіз мов програмування Swift та Objective-C. За результатами аналізу мову Swift було обрано найефективнішою для реалізації програмного продукту на iOS.

Розроблена архітектура додатку включає окремі модулі: інтерфейс користувача (SwiftUI), менеджер геолокації (CoreLocation), модуль звукових сповіщень (AVFoundation) та системні налаштування. Реалізовано функціональну обробку GPS-координат, визначення швидкості та порівняння з даними обмеження швидкості, отриманими з бази даних OpenStreetMap. Також реалізовано підтримку фіксованого обмеження швидкості, встановлення порогу сповіщень та можливість вимкнення звуку.

Додаток було протестовано як у середовищі симулятора, так і в реальних умовах. Результати тестування підтвердили коректну роботу системи, її стабільність та точність виявлення перевищення швидкості з допустимими перервами GPS. Додаток показав високу готовність до практичного використання та має потенціал для подальшого розвитку.

Ключові слова: контроль швидкості, перевищення швидкості, мобільний додаток, iOS, Swift, GPS, безпека дорожнього руху, звукові сповіщення.

ABSTRACT

The bachelor's thesis consists of 61 A4 pages, which contain 17 drawings and 1 table, the list of sources used contains 21 names.

This thesis is devoted to the development of a mobile application for iOS, which implements the function of notifying the driver about approaching a built-up bone. The main goal of the work is to improve road safety by timely informing the driver about exceeding the speed limit using software.

Time to work and check the speed of transport resources, the implementation of such a system and mobile accessories, as well as a comparative analysis of the Swift and Objective-C programming languages, were noted. According to the results of the analysis of the Swift language, the most effective was selected for the implementation of the software product on iOS.

An additional structure was developed: a user interface (SwiftUI), a geolocation manager (CoreLocation), a sound notification module (AVFoundation) and system settings. Functional GPS coordination, coordinated speed and comparison with speed limit data obtained from the OpenStreetMap database were implemented. Here's how to make a home speed measurement, set the notification threshold and mute the sound.

In addition, we have already tested the simulator, yes, we have real contracts. The results of testing robotic systems, its stability and accuracy of detecting overspeeding with tolerance and GPS performance. In addition, we are ready for practical classes. and has the potential for further development.

Keywords: speed control, overspeeding, mobile application, iOS, Swift, GPS, safe access to the car, navigation notification system.

ЗМІСТ

ВСТУП

Актуальність дослідження. В реаліях сьогодення керування автомобілем стає чим далі тим більш поширеним заняттям та необхідним вмінням для виживання у нашому суспільстві. Невід'ємною частиною цього процесу є дотримання швидкісних лімітів, часто це робиться за допомогою спідометрів. Згідно з Правилами дорожнього руху України, забороняється експлуатація транспортного засобу при несправності спідометра [2].

Одна з основних проблем, яку вирішує програмний модуль, – це недостатній контроль за дотриманням обмежень швидкості в дорожньому русі. Багато водіїв не завжди вчасно помічають обмеження швидкості або не усвідомлюють, що перевищують допустиму межу. Це часто призводить до порушень правил дорожнього руху, аварій та підвищеного ризику отримання шкоди майну чи здоров’ю учасників дорожнього руху. Традиційні заходи контролю, такі як дорожні знаки або періодичні огляди, не забезпечують водієві постійного зворотного зв'язку. Запропонований програмний модуль дозволяє своєчасно повідомляти користувача про наближення до обмеження швидкості та перевищення встановленого ліміту, що сприяє підвищенню безпеки дорожнього руху та зменшенню кількості порушень.

Додаток допоможе покращити безпеку дорожнього руху, визначаючи поточну швидкість транспортного засобу та надаючи своєчасні сповіщення про перевищення встановленого обмеження швидкості. Це не лише дозволить водіям керувати транспортним засобом комфортніше, але й зменшить ризик аварій, сприяючи загальній безпеці всіх учасників дорожнього руху.

Метою дослідження є розширення функціональних можливостей додатків для оповіщення про наближення до швидкісного ліміту.

Предметом дослідження виступає процес автоматизованого контролю за швидкістю руху транспортних засобів із застосуванням спеціального програмного забезпечення.

Об'єктом дослідження є програмний компонент, призначений для контролю швидкості на дорогах, його функціональні особливості, способи обробки вхідної інформації та взаємодія з інтерфейсом користувача.

Задачі дослідження:

1. Провести аналіз предметної області, існуючих методів, алгоритмів та інструментів для спостереження за швидкістю руху та швидкісних лімітів.

2. Проєктування архітектури додатку для оповіщення про наближення до швидкісного ліміту.

3. Розробка алгоритму функціонування додатку для оповіщення про наближення до швидкісного ліміту.

4. Програмна реалізація додатку для оповіщення про наближення до швидкісного ліміту за допомогою мови програмування Swift.

5. Тестування додатку для оповіщення про наближення до швидкісного ліміту.

6. Розробка інструкції користувача додатку для оповіщення про наближення до швидкісного ліміту.

Апробація роботи

Результати роботи були апробовані на LIV Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2025)

Публікації: За результатами бакалаврської кваліфікаційної роботи опубліковано тези доповіді на тему «Програмне забезпечення для допомоги водієві» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ МЕТОДІВ, АЛГОРИТМІВ ТА ІНСТРУМЕНТІВ ДЛЯ СПОСТЕРЕЖЕННЯ ЗА ШВИДКІСТЮ РУХУ ТА ШВИДКІСНИХ ЛІМІТІВ

1.1 Аналіз ключових аспектів моніторингу швидкості та швидкісних лімітів під час дорожнього руху

Дотримання швидкісних обмежень – один з ключових факторів безпеки на дорозі. В умовах сучасного інтенсивного трафіку та щільної забудови, перевищення швидкості значно збільшує ризик аварій, наражаючи на небезпеку як водія, так і інших учасників дорожнього руху.

Законодавчо визначені обмеження швидкості ґрунтуються на результатах численних досліджень з безпеки дорожнього руху, що враховують характеристики дорожнього покриття, особливості забудови, рівень аварійності та час реакції водія. Наприклад, навіть незначне перевищення швидкості в міських умовах може призвести до критичних наслідків при наїзді на пішохода: зіткнення на швидкості 30 км/год часто спричиняє травми, а на 50 км/год – стає смертельно небезпечним [4].

Сучасні дослідження наголошують, що швидкість впливає не лише на тяжкість наслідків ДТП, а й скорочує час на прийняття рішень. Водій, що рухається з перевищенням дозволеної швидкості, має менше часу на реакцію у випадку непередбаченої небезпеки, що є критичним в складних дорожніх ситуаціях. Більше того, висока швидкість знижує ефективність гальмування та збільшує гальмівний шлях.

Особливо важливо враховувати динамічні умови руху, такі як погодні умови, освітленість, стан дороги та поведінку інших водіїв. Навіть за наявності встановленого обмеження швидкості, безпечна швидкість може бути значно нижчою. Навіть дозволена швидкість 90 км/год на автомобільних дорогах може становити загрозу в умовах туману чи дощу, якщо зчеплення коліс із

дорогою погіршене. На мокрій поверхні навіть при швидкості 80 км/год зчеплення шин із дорогою зменшується приблизно вдвічі, а в сильну зливу — більш ніж у п’ять разів, що суттєво збільшує гальмівний шлях і ризик втрати керованості. Під час дощу або туману видимість і стан дорожнього покриття погіршуються, тому водій повинен зменшити швидкість і вибирати її відповідно до умов, навіть якщо вона нижча за дозволену. На мокрій або слизькій дорозі гальмівний шлях може збільшуватися в 4–5 разів, а ризик аквапланування для нових шин виникає вже на швидкості 60–90 км/год [5].

Важливу роль у дотриманні швидкісного режиму відіграють сучасні цифрові технології. GPS-навігатори, моніторинг швидкості в реальному часі, мобільні додатки з попередженням про перевищення ліміту – все це дозволяє не тільки інформувати водія про допустиму швидкість в певній зоні, а й формувати звички відповідальної поведінки за кермом.

Також активно впроваджуються автоматизовані засоби контролю, такі як камери фіксації швидкості, інтегровані з базами даних транспортних органів. Їх ефективність у зменшенні кількості порушень підтверджено в багатьох країнах. Дослідження показують, що на ділянках доріг із встановленими камерами фіксації порушень аварійність знижується в середньому на 20–30%. Наприклад, міжнародний огляд понад 45 досліджень підтверджує, що камери зменшують кількість аварій із травмами на 25–30%. У Португалії впровадження системи контролю середньої швидкості дозволило зменшити смертельні випадки на ділянках із камерами на 35%, а кількість ДТП із потерпілими — на 74%. В Україні, за даними поліції, кількість ДТП із потерпілими на ділянках із камерами знизилась утричі [6].

Особливу увагу слід приділяти просвітницькій та інформаційній роботі з водіями. Дотримання швидкісного режиму не повинно зводитись лише до уникнення штрафів, а має стати свідомим елементом культури поведінки на дорозі. Формування відповідального ставлення до правил дорожнього руху – це процес, що починається з навчання, підтримується особистим прикладом, громадським впливом та ефективною політикою безпеки.

Також важливо зауважити, що процес дорожнього руху це не лише знання та дотримання правил. Керуючи автівкою, водій зобов'язаний одночасно відслідковувати низку показників. Серед них: швидкість, розташування транспортного засобу на смузі, відстань до інших учасників, стан дорожнього покриття, світлофорні сигнали, дорожні знаки, пішоходи, маневри інших водіїв, а також внутрішні параметри автомобіля, скажімо, рівень пального, температура двигуна, індикація несправностей. У складних ситуаціях, таких як їзда містом або при поганій видимості, обсяг інформації, що надходить, збільшується багаторазово. Водночас, обмежений час на ухвалення рішень вимагає від водія постійної зосередженості, швидкої реакції та миттєвої оцінки. Саме тому, системи допомоги водієві мусять бути не лише точними, а й інтуїтивно зрозумілими, щоб не навантажувати його зайвою інформацією в критичні моменти.

У системах керування автомобілем суттєво забезпечити не тільки візуальні, але й аудіальні сповіщення. Під час руху водій не завжди має змогу безперервно стежити за приладовою панеллю або дисплеєм, особливо в динамічних умовах. Звукові сигнали дозволяють моментально звернути увагу на важливі події, не відволікаючись від дороги. Аудіоповідомлення можуть бути як короткими сигналами, так і голосовими підказками, що додатково збільшують обізнаність та дозволяють швидко реагувати. Поєднання візуального та звукового каналів передачі інформації підвищує надійність сприйняття та зменшує вірогідність пропущених сповіщень у стресових обставинах.

Підсумовуючи, дотримання швидкісних обмежень – це не просто вимога закону, а елемент системного підходу до забезпечення безпеки дорожнього руху. Його ефективність залежить не тільки від суворості контролю, а й від рівня обізнаності, технічних засобів підтримки та особистої відповідальності кожного водія.

1.2 Аналіз існуючих методів для контролю швидкості руху та швидкісних лімітів

За час існування історії автотранспорту людство знаходило все більше і більше способів для виміру швидкості пересування. Дані методи та прилади є невід'ємною складовою дорожнього руху сьогодення. Для ефективного впровадження новітніх технологій необхідно розглянути вже існуючі.

Одним з найпоширенішим приладом для виміру швидкості транспортного засобу є Спідометри. Ці прилади, винайдені Йосипом Белушичем у 1888 році є невід'ємною частиною кожного автомобіля в сучасності. Спідометри можна класифікувати за двома показниками: механізмом дій, способом відображення інформації.

За механізмом дій спідометри поділяються на:

- Механічні спідометри - базуються на перенесенні обертального руху від трансмісії до показового механізму, використовуючи гнучкий трос. В корпусі обертається магніт, поміщений в поле металевого диска, що змушує виникати вихрові струми, котрі спричиняють обертання стрілки на шкалі. Переваги: простота, автономність, незначна затримка відображення. Недоліки: вразливість до зносу троса, механічні похибки, складнощі з калібруванням [8].

- Електромеханічні спідометри - задіюють датчик обертання (наприклад, на коробці передач), котрий перетворює механічний рух в електричні імпульси. Ці імпульси надсилаються до приводу стрілки. Переваги: точніші за механічні, менше рухливих частин, покращена стабільність показів. Недоліки: залежність від живлення, складніші в ремонті [8].

- Електронні спідометри - швидкість обчислюється на основі частоти імпульсів від датчика швидкості, а обробка даних виконується мікроконтролером. Покази відображаються на цифровому чи аналоговому індикаторі. Переваги: висока точність, гнучкість налаштувань, здатність

інтегруватися з іншими електронними системами (GPS, ABS, CAN-шина). Недоліки: потребують живлення, складна конструкція, потребують калібрування [8].

- GPS-спідометри - швидкість визначається шляхом відстеження зміни географічних координат транспортного засобу у часі (через супутникові сигнали). Переваги: не залежать від трансмісії чи коліс, точні на великих прямих ділянках. Недоліки: нестабільна робота в тунелях, щільній забудові чи при слабкому сигналі GPS [8].

За способом відображення інформації спідометри поділяються на:

- Аналогові (стрілочні) – мають колову шкалу з позначками чисел, якою ковзає стрілка, показуючи поточну швидкість.Такі спідометри найбільш зустрічаються у традиційних автомобілях та мотоциклах. Плюси: комфортне відстеження змін швидкості, візуальна простота. Мінуси: менш точне визначення значення, ніж у цифрових [8].

- Цифрові – швидкість демонструється у формі конкретного числового показника на LED або LCD-дисплеї. Цим спідометрам віддають перевагу у сучасних авто, електрокарах, мотоциклах новітніх моделей. Переваги: точність, швидке зчитування, сучасний дизайн. Недоліки: складніше оцінити динаміку змін, особливо при короткочасних коливаннях [9].

- Проєкційні (HUD — Head-Up Display) - показують швидкість на лобовому склі або на прозорому екрані перед водієм, застосовуючи проєкцію. Зазвичай зустрічаються в авто преміум-класу або як окремий додаток. Плюси: водій менше відволікається від дороги, що знижує ризик неуважності. Мінуси: вимагають точного налаштування, можуть бути обмежені у сумісності з різними типами лобового скла [9].

Окрім спідометрів що встановлені безпосередньо в автомобілях також існують інші способи виміру швидкості автомобіля на дорозі, та забезпечення дотримання швидкісного ліміту:

- Радари, або радіолокаційні прилади для вимірювання швидкості, базуються на ефекті Доплера. Вони надсилають радіохвилі, які відбиваються

від транспортного засобу, що рухається. Шляхом вимірювання зміни частоти відбитого сигналу порівняно з випроміненим, обчислюється швидкість об'єкта. Існують стаціонарні та переносні варіанти, їх монтують на узбіччях, мостах чи інших відповідних конструкціях [10].

- Лідари, або Light Detection and Ranging, вимірюють відстань та швидкість, використовуючи лазерні імпульси. Пристрій випускає світловий імпульс, який відбивається від автомобіля. За часом повернення сигналу визначають відстань, а з послідовних вимірювань — швидкість. Лідари забезпечують високу точність вимірювання швидкості на великих відстанях, але їх ефективність може знижуватися в складних погодних умовах. Зазвичай, вони використовуються як ручні пристрої або встановлюються на штативах [10].

Однак додатково існує цікава система оповіщення про порушення швидкісного ліміту у вигляді дзвіночків. Широкого поширення вона набула у Японії у 1980х роках [7]. Так звані сповіщення про перевищення швидкості (інколи їх називають звуковим попередженням про швидкість) — це звуковий сигнал, що автоматично активується, коли транспортний засіб їде швидше за певну межу (нерідко це 105 км/год) [11]. У японських авто такий сигнал найчастіше втілювався як тихе, ритмічне дзеленчання, подібне до звуку дзвінка чи електронного сигналу. Замість різких гудків чи відчутної вібрації, дзвіночок реалізовував свій вплив як ніжне, тактильне нагадування. Він не дратував водія, а натомість викликав легкий дискомфорт. Це спонукало до зниження швидкості. У поєднанні з японською культурою, що характеризується дисципліною та турботою про безпеку, цей підхід виявився надзвичайно ефективним у запобіганні порушенням [7].

Ми вивчили приклади обладнання для вимірювання швидкості, її індикації, інформування та забезпечення дотримання обмежень швидкості. Зокрема, було ретельно описано спідометри, як ключові інструменти прямого контролю швидкості водієм, класифіковані згідно з принципом дії та методом відображення даних. Також розглянуто роботу радарів та лідарів, що

забезпечують безконтактне, високоточне вимірювання швидкості транспортних засобів в русі, що дозволяє ефективно контролювати дорожню обстановку як на окремих відрізках, так і на великих автошляхах.

1.3 Аналіз сучасних програмних засобів для показу швидкості транспортного засобу

На сучасному ринку існує широкий спектр програмних продуктів, розроблених для контролю швидкості руху пристрою або транспортного засобу в реальному часі. Ці рішення зазвичай реалізовані як мобільні додатки для смартфонів або вбудоване програмне забезпечення в автомобільних інформаційно-розважальних системах. Вони надають користувачам візуальні та звукові сповіщення, що спрямовані на забезпечення дотримання встановлених обмежень швидкості. Найбільш поширені категорії таких застосунків включають: навігаційні програми, які, крім прокладання маршруту в реальному часі, також інформують про поточну швидкість, дозволену швидкість на певній ділянці дороги, наявність камер контролю швидкості та зон підвищеної уваги; спідометри, які імітують аналоговий або цифровий спідометр на екрані смартфону та дозволяють користувачу налаштувати граничний поріг швидкості для звукових попереджень; а також універсальні системи моніторингу водіння, які збирають статистику поїздок, аналізують стиль водіння та сповіщають про небезпечні маневри або перевищення швидкості.

Більшість таких програм працюють у зв'язці з GPS-модулями пристрою, що дає змогу з високою точністю визначати швидкість та порівнювати її з даними про швидкісні обмеження, які постійно оновлюються через підключення до мережі. Окрім цього, деякі програми інтегруються з зовнішніми датчиками або системами транспортного засобу через інтерфейси Bluetooth або OBD-II, забезпечуючи ще точніший та швидкий обмін даними,

однак ми їх розглядати не будемо оскільки вони потребують додаткових технологій та вкладень.

Розглянемо деякі з них (рис. 1.1):

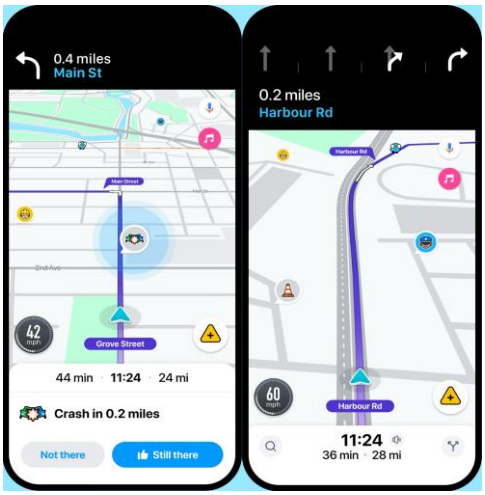

Рисунок 1.1 – Загальний вигляд інтерфейсного вікна програмного засобу «Waze»

На представленому вікні видно головний інтерфейс застосунку Waze, відомого навігаційного рішення. Головна задача Waze – це прокладання маршрутів у реальному часі. Разом з тим, Waze виступає як корисний інструмент для контролю швидкості під час керування автомобілем. Програма не тільки відслідковує поточну швидкість вашого пристрою, користуючись GPS, але й зіставляє її з дозволеними швидкісними лімітами на конкретній ділянці дороги. Завдяки активній спільноті користувачів, котрі регулярно надсилають актуальну інформацію про стан доріг, розташування камер спостереження та зміни лімітів, Waze гарантує точні та своєчасні попередження про перевищення швидкості. Це робить застосунок незамінним помічником для безпечної їзди. Однак існують і додатки основною функцією яких є саме відображення швидкості (рис. 1.2).

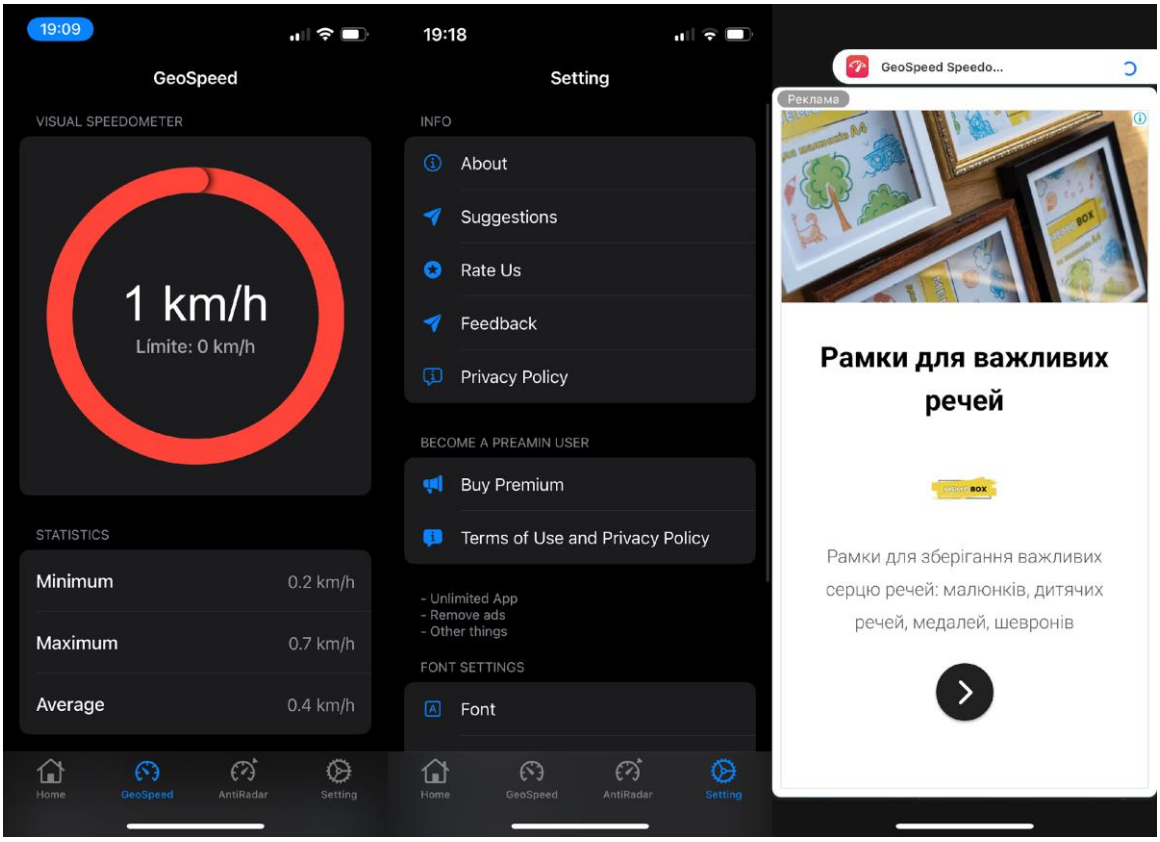

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна програмного засобу «GeoSpeed»

Цей додаток вже зосереджений винятково на функціях спідометра. Як і у випадку з «Waze», що вже розглядалось, він застосовує GPS для розрахунку швидкості руху. Але, на відміну від навігаційних сервісів, програма не отримує дані про ліміти швидкості з будь-яких баз. Натомість, користувач повинен самостійно вказувати максимальну швидкість, при перевищенні якої змінюється лише зовнішній вигляд – колір контуру навколо спідометра. Слід відзначити, що відсутність звукових сигналів зменшує оперативність реакції в складних дорожніх умовах. До того ж, серйозним недоліком є нав'язлива реклама, що з'являється під час руху, що не тільки суперечить етиці, але й може бути небезпечною, відволікаючи водія від дороги. Розглянемо ще один додаток схожого функціоналу (рис. 1.3).

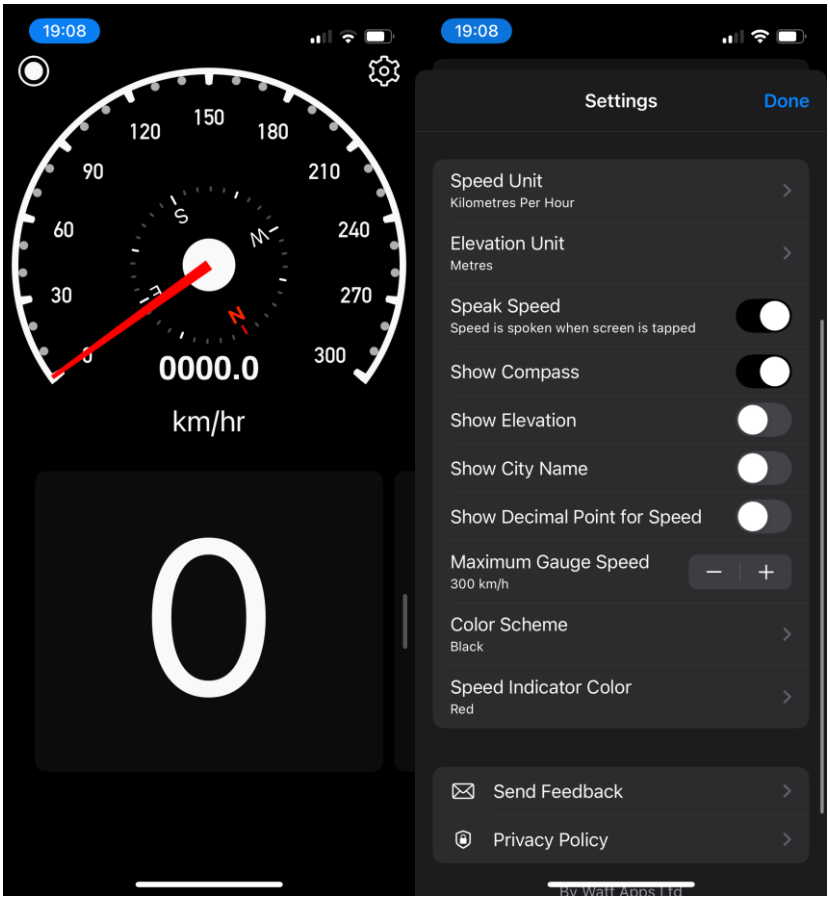

Рисунок 1.3 – Загальний вигляд інтерфейсного вікна програмного засобу «SimpleSpeedometer»

Цей додаток, як і попередні, застосовує GPS для встановлення поточної швидкості машини. Одночасно з цим він надає декілька додаткових можливостей, здатних стати у пригоді водієві, наприклад: вмонтований компас, інтегровані мапи й опцію оголошення швидкості при торканні екрана. Варто, однак, звернути увагу на те, що у додатоку немає підтримки щодо отримання даних про обмеження швидкості на маршруті, а також не передбачає системи сповіщення про наближення до дозволеної межі. Брак подібних функцій зменшує його корисність як засобу для гарантування безпечного дотримання швидкості.

Для кращої візуалізації, було виділено ключові критерії для порівняння, такі як наявність стрілочного та цифрового спідометрів, інформації про тип ліміту швидкості, простий та зрозумілий інтерфейс, сповіщень про перевищення швидкості. Результат порівняння наведено у таблиці 1.1.

Таблиця 1.1 - Результат порівняння існуючих додатків

Додаток	Спідометр	Ліміт швидкості	Простий інтерфейс	Сповіщення про перевищення швидкості
Waze	Цифровий	За геопозицією	Відсутній	Відсутні
GeoSpeed	Цифровий	жорсткий	Відсутній	Відсутні
Simle Speedometer	Цифровий, аналоговий	жорсткий	Відсутній	Відсутні
Розробка	Цифровий, аналоговий	За геопозицією та жорсткий	Присутній	Присутні

Як видно з аналізу, лише один з доступних додатків пропонує функціональність отримання обмеження швидкості залежно від геолокації, що суттєво обмежує можливості користувача налаштовувати контроль швидкості до конкретних дорожніх умов. Водночас більшість цих додатків мають перевантажений або заплутаний інтерфейс, що ускладнює взаємодію, особливо під час керування автомобілем. Деякі з них додатково містять інтегровану рекламу, яка може відволікати водія та створювати потенційно небезпечні ситуації. Крім того, жоден з розглянутих додатків не реалізує механізм активних сповіщень у режимі реального часу про перевищення встановленого обмеження швидкості, що є критично важливою функцією для забезпечення безпечного стилю водіння та своєчасного реагування на небезпечні ситуації на дорозі. Це свідчить про те, що ринок все ще потребує інтуїтивно зрозумілого, ефективного та ненав'язливого рішення, орієнтованого на безпеку та зручність користувача.

Аналіз виявив основні функції, сильні та слабкі сторони програм, створених для відображення швидкості авто. Наприклад, такі сервіси, як Waze, надають не тільки точні дані про швидкість, але й попереджають про перевищення дозволеної, використовуючи актуальну інформацію з баз даних. Інші ж акцентують увагу на візуалізації, не маючи засобів для автоматичного

контролю лімітів чи сповіщення водія. Додаткові опції, як голосове озвучення швидкості чи компас, покращують комфорт користування, але не заповнюють прогалини у системах попередження. Відтак, продуктивність таких додатків багато в чому залежить від балансу між точністю вимірювань, зрозумілістю інтерфейсу та інтеграцією з інформацією про ситуацію на дорозі.

1.4 Висновок до розділу 1

У першому розділі було висвітлено ключові питання, які стосуються регулювання швидкості на дорогах, сучасного обладнання для вимірювання швидкості, а також програмних рішень, які сприяють дотриманню швидкісних обмежень.

Аналіз показав, що дотримання швидкісного режиму — це не тільки питання виконання законодавчих норм, але й вкрай важливий фактор дорожньої безпеки. Перевищення швидкості значно збільшує ризик виникнення аварій, зменшує час на реакцію водія, збільшує гальмівний шлях, а в певних умовах (дощ, туман, ожеледиця) робить навіть дозволену швидкість потенційно небезпечною. Сучасні технології, як-от GPS-навігатори, мобільні додатки, системи допомоги водієві, суттєво підвищують обізнаність водія, а автоматизовані системи контролю (камери, радари, лідари) ефективно знижують кількість порушень та аварій.

З технічної точки зору існує широкий спектр пристроїв для вимірювання швидкості: від традиційних спідометрів (механічних, електронних, GPS) до зовнішніх пристроїв контролю (радарів, лідарів). Особливу увагу слід приділити інтеграції аудіо- та візуальних попереджувальних сигналів, що дає водієві змогу швидше реагувати на можливі загрози, не відволікаючись від дорожньої ситуації [16]. Крім того, аналіз японського досвіду застосування м'яких звукових сигналів як поведінкового стимулу відкриває перспективу більш гуманного підходу до контролю.

Окрему нішу займають програмні рішення контролю швидкості, які завдяки використанню GPS та інтерфейсів попередження можуть виступати додатковими або навіть основними інструментами моніторингу швидкості. Вони сприяють формуванню культури відповідального водіння, постійно інформуючи про дозволені обмеження та можливі порушення.

Отже, ефективний контроль швидкості на дорогах має бути системним і включати три основні компоненти:

- інженерно-технічне забезпечення (спідометри, радари, лідари);
- цифрові рішення та засоби оповіщення (звукові та візуальні сповіщення, GPS-системи);
- просвітницька робота та заохочення культури безпеки серед водіїв.
- Комплексне застосування цих засобів не лише забезпечує дотримання правил, але й сприяє створенню безпечнішого, більш передбачуваного та гуманного дорожнього середовища.

Для розробки програми, що має бути помічником водія, знадобиться ґрунтовне розуміння технічних аспектів, поряд з можливістю користуватися перевіреними сховищами даних, котрі оперативно надають потрібні відомості, при цьому не відвертаючи увагу водія від безпосереднього управління автівкою.

2 ПРОЕКТУВАННЯ IOS ДОДАТКУ ДЛЯ СПОВІЩЕННЯ ПРО ПЕРЕВИЩЕННЯ ШВИДКІСНОГО ЛІМІТУ

2.1 Розробка архітектури iOS додатку для сповіщення про перевищення швидкісного ліміту

Архітектура додатку, що відповідає за контроль швидкості транспортного засобу, є визначальним фактором, що впливає на загальну організацію та структуру всієї системи. Вона передбачає конкретне розділення головних складових частин, визначення їхніх обов'язків і взаємодій, формуючи концептуальний фундамент для ефективної розробки програмного забезпечення. Добре розроблена архітектура гарантує стабільність, надійність та здатність системи пристосовуватися до реальних дорожніх умов.

Розробка архітектури додатку для контролю швидкості транспортного засобу потребує всебічного підходу, який враховує як технічні тонкощі втілення, так і специфіку взаємодії з водієм в реальному дорожньому середовищі. Ключовим аспектом є впровадження дієвої системи аудіо-сповіщень, що гарантувала б негайне інформування водія, не відволікаючи його надмірно. До того ж, необхідно розробити зрозумілий та стабільний інтерфейс, який надасть водієві можливість миттєво коригувати гучність сповіщень або тимчасово приглушувати їх, не відволікаючись від керування. Усі ці складові безпосередньо впливають на архітектурні рішення системи, визначаючи її компоненти, їхні взаємодії та логіку роботи, беручи до уваги потреби безпеки, зручності та пристосованості до різних експлуатаційних умов.

На рисунку 2.1 представлено загальну архітектуру додатку для сповіщення про перевищення швидкісного ліміту.

Рисунок 2.1 – Загальна архітектура додатку для сповіщення про перевищення швидкісного ліміту

На зображенні подано компонентно-орієнтовану архітектуру програмного забезпечення, розроблену для контролю швидкості руху на основі даних геолокації. Архітектура базується на принципі чіткого розподілу відповідальності між окремими модулями, кожен з яких виконує визначену функцію та взаємодіє з іншими компонентами системи через зрозумілі інтерфейси. Цей підхід забезпечує високу модульність, масштабованість та спрощує як розробку, так і подальшу підтримку та вдосконалення програмного продукту.

Центральним елементом архітектури є графічний інтерфейс користувача. Він є головним каналом взаємодії між системою та водієм, відображаючи ключову інформацію, зокрема поточну швидкість транспортного засобу та сповіщення про її перевищення. Крім того, інтерфейс надає доступ до налаштувань, дозволяючи користувачеві адаптувати функціональність системи до власних потреб: регулювати гучність аудіосповіщень, обирати їх тип або вимикати їх повністю. Простота та інтуїтивність інтерфейсу є критично важливими у контексті безпеки дорожнього руху, оскільки дозволяють водієві отримувати інформацію швидко та без зайвого відволікання від керування.

Графічний інтерфейс взаємодіє з менеджером геолокації – компонентом, відповідальним за збір та обробку даних про поточне місцезнаходження та швидкість руху пристрою. Менеджер ініціює запити на доступ до геоданих, отримує оновлення координат у реальному часі та надсилає відповідні дані до інтерфейсу користувача. Крім того, він звертається до бази даних швидкісних лімітів для отримання дозволеного значення швидкості для конкретної ділянки дороги. У разі виявлення перевищення ліміту менеджер сповіщає про це інші компоненти системи для подальшої реакції.

Звукові сповіщення генеруються окремим модулем - менеджером звукових сповіщень, що обробляє сигнали про перевищення швидкості та формує відповідні аудіосигнали. Особливу увагу приділено можливості налаштування гучності та швидкого доступу до вимкнення сповіщень, що критично важливо в дорожньому середовищі, де фактори навколишнього шуму та навантаження на увагу водія можуть змінюватися.

Для взаємодії з базою даних швидкісних лімітів є окремий автономний компонент, який надсилає до неї запити з координатами користувача, а база повертає відповідний ліміт. Такий розподіл дозволяє забезпечити ефективну роботу з великим обсягом даних, а також спрощує оновлення та масштабування інформаційної частини системи без впливу на основну логіку програми.

Окремий блок виділено для користувацьких налаштувань, де зберігаються всі параметри, вибрані користувачем. Це дає змогу зберігати індивідуальні переваги між сеансами використання та підтримувати персоналізовану взаємодію із системою.

Підсумовуючи, така архітектура забезпечує високу гнучкість, надійність та простоту у використанні, що є критично важливим для систем, які взаємодіють з користувачем у режимі реального часу та в умовах підвищених вимог до безпеки.

Додаток буде розроблено з використанням об’єктно-орієнтованого програмування (ООП), яке забезпечить логічну структуризацію коду,

модульність і можливість повторного використання компонентів. Проте ключовою технологією в реалізації додатку стане SwiftUI — сучасна декларативна фреймворк-платформа для розробки інтерфейсів на iOS. SwiftUI дозволяє створювати інтерфейс, який реагує на зміну даних у реальному часі, що критично важливо для задачі контролю швидкості руху транспортного засобу. Використання SwiftUI значно спрощує розробку інтуїтивно зрозумілого та адаптивного інтерфейсу, який може швидко оновлюватися відповідно до вхідних геоданих або змін параметрів користувача.

Об’єктно-орієнтована модель проєкту передбачає створення окремих менеджерів у вигляді класів для ключових підсистем: геолокації, звукових сповіщень, взаємодії з базою даних лімітів швидкості та керування користувацькими налаштуваннями. Кожен із цих менеджерів інкапсулює відповідну логіку, обробляє дані в межах своєї зони відповідальності та взаємодіє з іншими модулями через чітко визначені інтерфейси. Такий підхід підвищує стабільність системи, забезпечує надійний захист даних, а також дозволяє ефективно виявляти та усувати потенційні помилки без впливу на інші частини програми.

Менеджер геолокації буде відповідати за збирання поточних координат користувача, визначення швидкості руху та обробку запитів на отримання дозволених швидкісних обмежень у поточному місці. Він періодично буде оновлювати дані та надаватиме актуальну інформацію інтерфейсу SwiftUI для відображення або реагування на перевищення ліміту. У свою чергу, менеджер звуку контролюватиме активацію аудіальних сповіщень, регулювання гучності, відключення сигналів і вибір типу сповіщення — що дозволить користувачу гнучко адаптувати систему під власні уподобання.

SwiftUI забезпечить зручне відображення змін у режимі реального часу: наприклад, коли швидкість перевищує допустиме значення — інтерфейс автоматично змінює колір, виводить попередження або активує аудіосигнал без необхідності ручного оновлення стану. Завдяки підтримці двостороннього

зв’язку з даними (data binding) SwiftUI дозволяє безперервно синхронізувати стан системи з її візуальним відображенням.

Крім того, архітектура проєкту включатиме механізми збереження налаштувань користувача (наприклад, вибраного рівня гучності, жорсткого швидкісного ліміту або порогу спрацювання сповіщень) за допомогою вбудованого сховища UserDefaults [18], яке є стандартним інструментом для збереження простих даних у середовищі iOS/macOS. UserDefaults дозволяє ефективно зберігати та швидко отримувати значення типів Bool, String, Int, Double, масивів та словників без необхідності складного налаштування або використання сторонніх бібліотек. Завдяки простоті інтеграції, UserDefaults забезпечує мінімальне навантаження на систему, не вимагає підключення до зовнішніх серверів або баз даних, а також надає можливість миттєвого зчитування та запису значень, що особливо зручно під час роботи з елементами інтерфейсу, які змінюють поведінку програми в режимі реального часу. Крім того, це рішення дозволяє уникнути зайвої складності під час налаштування проєкту, підвищує стабільність роботи програми, а в поєднанні з синхронізацією iCloud забезпечує можливість збереження та автоматичного відновлення налаштувань користувача на різних пристроях, що позитивно впливає на користувацький досвід.

Загалом, об’єднання переваг ООП і SwiftUI створює ефективну основу для реалізації модульного, масштабованого й надійного застосунку, який зможе оперативно реагувати на зміну контексту руху та не потребуватиме постійної взаємодії користувача з екраном.

На рисунку 2.2 представлено загальний алгоритм роботи програмного модуля для сповіщення про перевищення швидкісного ліміту.

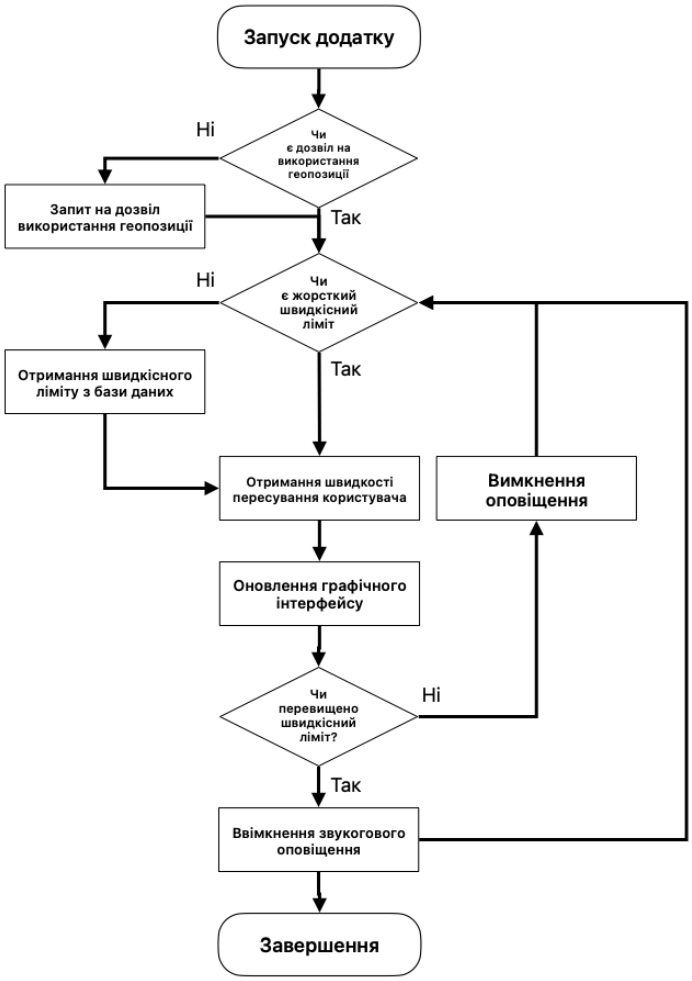

Рисунок 2.2 – Загальний алгоритм роботи додатку для сповіщення про перевищення швидкісного ліміту

Алгоритм, зображений на блок-схемі, описує порядок дій під час запуску і роботи мобільного додатку, який виконує контроль швидкості руху користувача, базуючись на його геолокації з подальшим звуковим сповіщенням у разі перевищення встановленого ліміту.

Функціонування додатку починається з етапу ініціалізації після запуску, що передбачає завантаження основних складових системи, як-от менеджери геолокації, аудіо, графічного інтерфейсу, а також встановлення зв’язку з локальною чи віддаленою базою даних, яка містить ліміти швидкості для конкретних ділянок дороги.

Наступним кроком є перевірка наявності дозволу на використання геолокації. Якщо такий дозвіл раніше не було надано, система ініціює запит до користувача на його надання. Без цього дозволу система не може отримати інформацію про місцеперебування, тому робота додатку, по суті, припиняється або переходить у пасивний режим до моменту надання дозволу.

У разі отримання дозволу, система перевіряє, чи задано жорсткий швидкісний ліміт – заздалегідь встановлене значення швидкості, перевищення якого є критичним, незалежно від місця перебування. Якщо такого ліміту немає або він динамічний, додаток звертається до бази даних для отримання актуального обмеження швидкості, що відповідає поточній геопозиції користувача.

Після визначення ліміту швидкості система переходить до отримання інформації про швидкість пересування користувача шляхом аналізу GPS-даних у реальному часі. Ці дані безперервно оновлюються і передаються до модуля оновлення графічного інтерфейсу, який візуалізує поточну швидкість руху, а також інші параметри – наприклад, граничний ліміт, статус геолокації, сигналізацію та інше.

На основі отриманих даних система здійснює перевірку: чи перевищено швидкісний ліміт. Якщо ліміт не перевищено, додаток продовжує функціонувати в режимі моніторингу. Якщо ж швидкість вища за дозволену – ініціюється ввімкнення звукового сповіщення. Це може бути звуковий сигнал, голосове попередження або інший аудіовиклик, що миттєво інформує водія про порушення.

Далі система очікує, поки користувач зменшить швидкість або вимкне сповіщення власноруч через налаштування в інтерфейсі. Якщо швидкість знижується до допустимого рівня або користувач вибирає вимкнення сповіщень, звуковий сигнал припиняється.

Завершується алгоритм логічною точкою завершення роботи додатку, яка може бути зумовлена його закриттям або зупинкою поточного сеансу контролю.

Цей алгоритм підкреслює важливість безперервного збору даних у реальному часі, гнучкість у роботі з лімітами (локальними або жорсткими), а також надання зручного і оперативного зворотного зв'язку для користувача за допомогою звукових сповіщень, що відповідає концепції безпечного і персоналізованого моніторингу швидкості.

2.2 Розробка взаємодії програмних модулів системи для сповіщення про перевищення швидкісного ліміту

Розробка ефективної мобільної програми, яка відстежує швидкість на основі даних геолокації, потребує гармонійної взаємодії між декількома спеціалізованими програмними блоками. Кожен модуль відповідає за певну сукупність функцій, але лише співпрацюючи один з одним, вони забезпечують стабільну, гнучку та інформативну систему сповіщень про перевищення швидкості, з орієнтацією на зручність для користувача.

Архітектура застосунку включає п'ять основних програмних модулів, кожен з яких відіграє важливу роль у реалізації загальної функціональності системи. До них належать: модуль інтерфейсу користувача, модуль звукового сповіщення, модуль налаштувань користувача, модуль геолокації та база даних обмежень швидкості. Всі ці компоненти працюють у тісній взаємодії, забезпечуючи повний цикл обробки даних – від моменту отримання поточної геолокації користувача до створення та відправлення сповіщення про перевищення дозволеної швидкості в режимі реального часу. Такий розподіл обов'язків між модулями дозволяє досягти ефективності, масштабованості та гнучкості в розробці та супроводі програмного забезпечення. Структурна схема взаємозв'язків між конкретними компонентами показана на рисунку 2.3.

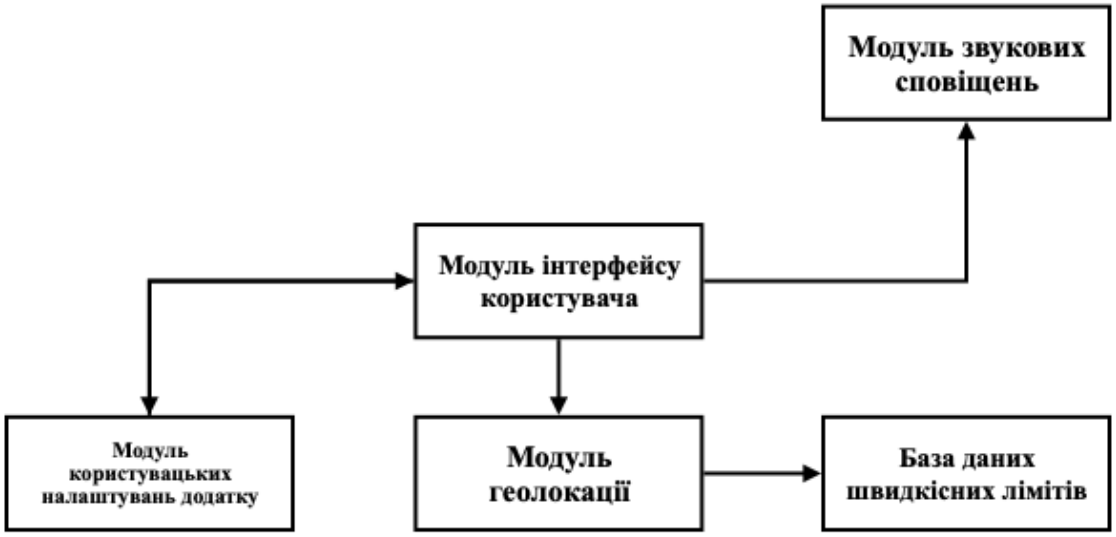

Рисунок 2.3 – Структурна схема програмних модулів додатку для оповіщення про наближення до швидкісного ліміту

Модуль користувацького інтерфейсу (UI) - це критичний елемент системної архітектури, котрий забезпечує безпосередній взаємозв'язок між користувачем та програмою. Його головна функція - візуалізація всіх важливих даних у зручному та зрозумілому форматі. Через цей модуль користувач має доступ до ключових показників, таких як поточна швидкість руху, встановлена швидкість для даної зони, стан звукових сповіщень, а також загальний статус системи.

Окрім відображення інформації, UI-модуль дозволяє керувати параметрами користувача, включаючи увімкнення/вимкнення сповіщень, перемикання режимів перегляду, регулювання гучності та інші налаштування. Таким чином, він функціонує як центральний канал комунікації між користувачем та внутрішніми процесами системи.

Важливу роль модуль UI відіграє у вигляді тригера для запуску інших модулів. Більшість дій користувача, наприклад, запуск програми, надання доступу до геолокації чи зміна налаштувань, запускають відповідні процеси у модулях геолокації, звукових сповіщень або персональних налаштувань. Ця архітектурна логіка спрямована на мінімізацію фонових обчислень та

підвищення загальної продуктивності системи, активуючи лише необхідні компоненти у певний момент часу.

Враховуючи використання SwiftUI, інтерфейс реалізується як реактивна система, що дозволяє оновлювати дані в реальному часі без потреби ручного перезавантаження компонентів. Це значно покращує взаємодію з користувачем, оскільки всі зміни, як-от зміна швидкості або перевищення ліміту, негайно відображаються на екрані та автоматично запускають супутні дії, включаючи активацію звукових сповіщень або відображення попереджень.

Підсумовуючи, модуль UI не тільки відображає інформацію, а й координує всю систему, забезпечуючи динамічну взаємодію між користувачем та програмною логікою програми. Алгоритм роботи модулю графічного інтерфейсу зображено на рисунку 2.4.

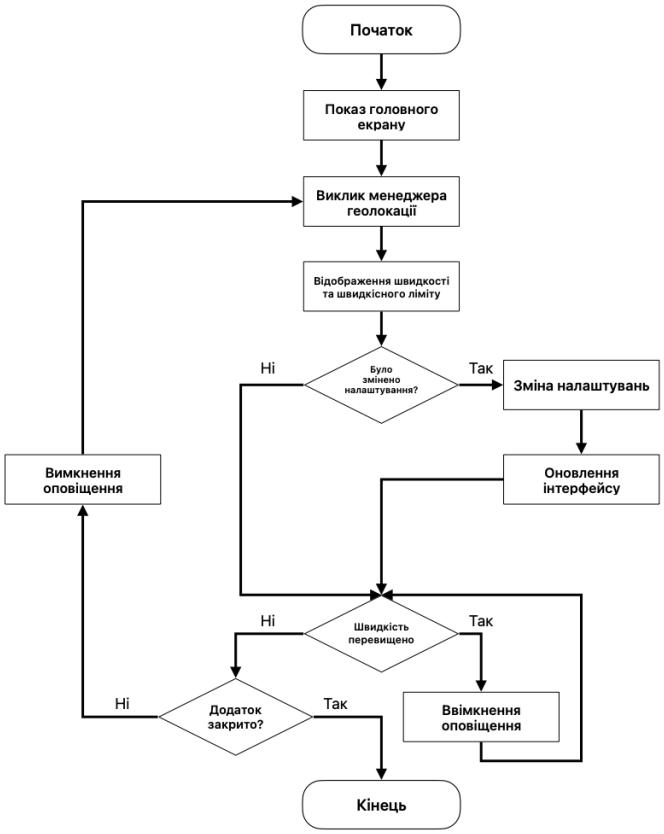

Рисунок 2.4 – Алгоритм роботи модуля графічного інтерфейсу

Модуль звукових сповіщень відіграє ключову роль у миттєвому інформуванні водія про перевищення швидкості. Головна його функція - опрацювання запитів на відтворення звукових сигналів, що надходять у випадках, коли фіксується порушення ліміту швидкості. Це дозволяє водієві оперативно реагувати, не зосереджуючи увагу на постійному контролю екрану пристрою. Алгоритм роботи модуля звукового сповіщення зображено на рисунку 2.5.

Цей модуль тісно взаємодіє з інтерфейсом користувача, надаючи гнучке управління звуковими сповіщеннями. Користувач може вмикати і вимикати звукові попередження в реальному часі, регулювати гучність, а також повністю вимкнути функцію, якщо це потрібно. Усі налаштування виконуються через інтерфейс, а модуль сповіщень миттєво реагує на зміни параметрів, гарантуючи безперебійну та адаптивну роботу системи.

Окрім основної функції сповіщень, модуль також займається налаштуванням та збереженням обраного режиму оповіщень. При повторному запуску програми, він відтворює попередні налаштування користувача, що збільшує зручність використання та персоналізацію.

Важливо зазначити, що модуль реалізовано з використанням об’єктно-орієнтованого підходу, який дозволяє чітко відокремити управління звуком від інших компонентів системи. Це сприяє кращій масштабованості та полегшує розширення функціоналу, наприклад, додавання різних звукових сигналів або можливості їх налаштування. Таким чином, модуль звукових сповіщень не тільки збільшує безпеку на дорозі, але й відповідає потребам кожного водія.

Модуль геолокації є ключовим компонентом системи контролю швидкості, оскільки він надає доступ до свіжих відомостей про місцезнаходження користувача.

Рисунок 2.5 – Алгоритм роботи модуля звукового сповіщення

Головне завдання цього модуля – постійно відслідковувати координати пристрою у реальному часі, на підставі яких обчислюється поточна швидкість. Ці дані використовуються іншими модулями для подальших операцій, наприклад, для порівняння фактичної швидкості з допустимими обмеженнями.

Окрім визначення швидкості, модуль також запускає запит до бази даних обмежень швидкості. В залежності від геолокації користувача, він

формує запит для отримання відповідного обмеження швидкості для конкретної ділянки дороги. Це дозволяє динамічно оновлювати контрольні параметри та оперативно реагувати на зміни умов руху, наприклад, при переїзді з міста на трасу.

Відмінною та ключовою особливістю модуля геолокації є його глибока інтеграція з іншими компонентними системами, що забезпечує послідовну та ефективну взаємодію з ними. Крім того, модуль постійно обмінюється даними з інтерфейсом користувача, а також тісно співпрацює з модулем звукового сповіщення, створюючи єдину інформаційну екосистему. Дані GPS у режимі реального часу дозволяють швидко визначити поточну швидкість транспортного сполучення, а також порівняти її з допустимими обмеженнями для конкретного місця. Ця інформація негайно передається на графічний інтерфейс, відображається у зручному для водія форматі, за допомогою візуальних індикаторів або цифрових значень. У разі перевищення дозволеної швидкості система автоматично активує звукове попередження, яке не тільки інформує користувача про порушення, але й сприяє швидкому реагуванню для усунення можливих штрафів або надзвичайних ситуацій. Такий рівень інтеграції забезпечує високу точність, швидкість та зручність використання, що є критичними факторами для додатків безпеки дорожнього руху.

Технічна реалізація модуля базується на принципах об'єктно-орієнтованого програмування, що гарантує масштабованість та надійність роботи. Це дає змогу легко адаптувати модуль до змін у вимогах – наприклад, додати підтримку альтернативних джерел даних про геолокацію або вдосконалити обробку даних у зонах з поганим покриттям. Все це робить модуль геолокації критично важливою частиною системи, формуючи основу її функціональної точності та адаптивності. Алгоритм роботи модуля геолокації зображено на рисунку 2.6

Рисунок 2.6 — Алгоритм роботи модуля геолокації

База даних обмежень швидкості для системи ґрунтуватиметься на зовнішньому джерелі. Ручний збір і підтримання актуальності таких даних вимагають надмірних ресурсів та безперервного супроводу. Під час аналізу було вивчено кілька відомих сервісів, зокрема Google Roads API та HERE Maps. Хоча ці рішення пропонують точні й зручні інтерфейси, вони мають значні недоліки – наприклад, обмежене число безкоштовних запитів та комерційні ліцензії. Враховуючи це, було вирішено використати відкриту платформу OpenStreetMap, яка є безкоштовною, гнучкою у використанні та постійно оновлюється зусиллями спільноти користувачів. Такий вибір дозволяє досягти балансу між точністю даних, доступністю та масштабованістю системи без додаткових фінансових вливань.

Модуль користувацьких налаштувань призначений для адаптації функціоналу застосунку до індивідуальних потреб кожного користувача. Завдяки цьому модулю можна змінювати параметри, такі як гучність звукових сповіщень, тип повідомлень, а також можливість їх повного вимкнення. Реалізація цього модуля відбуватиметься з використанням технології UserDefaults, що є стандартним і перевіреним рішенням у середовищі SwiftUI для зберігання легковагових налаштувань.

UserDefaults дає можливість зберігати дані навіть після завершення роботи з застосунком, що гарантує збереження персональних налаштувань між сесіями користування. Його застосування є оптимальним для конфігураційних даних, які змінюються рідко, але мають критичний вплив на зручність взаємодії з додатком. Завдяки інтеграції з SwiftUI, налаштування можна змінювати в режимі реального часу, а інтерфейс миттєво відображатиме зміни, що покращує загальний досвід користування.

У розглянутій системі контролю швидкості, що використовує дані геолокації, взаємодія між складовими реалізована за принципом централізованої координації. Модуль інтерфейсу користувача (UI) виконує роль ключового елемента, ініціюючи та координуючи роботу всіх інших модулів. Ця ієрархічна структура дозволяє зосередити логіку управління системою в одному місці, забезпечуючи прозорість взаємодії, узгодженість даних та зручність масштабування.

При запуску додатку UI ініціалізує модуль геолокації, модуль звукових сповіщень. Після ініціалізації кожен з цих модулів залишається доступним через посилання з центрального модуля, що дозволяє графічному модулю ефективно передавати команди, обробляти події та оновлювати дані на екрані в режимі реального часу.

Наприклад, після старту системи модулю геолокації розпочинає відслідковування координат та розрахунок швидкості, також він формує запит до бази даних швидкісних лімітів для визначення допустимого значення в поточній точці. Ці дані передаються до інтерфейсу, який оновлює інтерфейс

користувача. У випадку виявлення перевищення швидкості, графічний модуль надсилає команду модулю звукових сповіщень на активацію аудіосигналу, при цьому враховуються налаштування гучності та активності сповіщень, встановлені в UserDefaults.

Система також передбачає зворотний зв'язок: якщо користувач змінює параметри в налаштуваннях через інтерфейс, графічний модуль передає ці зміни до модулю користувацьких налаштувань, який зберігає їх у UserDefaults для подальшого використання. Будь-яка зміна негайно застосовується до роботи відповідних модулів без необхідності перезавантаження системи.

Отже, цей модуль служить не лише графічним інтерфейсом для відображення даних, а фактично виступає центральним ядром, яке координує взаємодію між усіма технічними компонентами системи. Саме через нього обробляються вхідні дані, передаються сигнали до відповідних підсистем, таких як модулі звукового сповіщення, контролю швидкості та геолокації, а також оновлюється візуальне представлення інформації на екрані. Такий підхід дозволяє досягти високого рівня архітектурної гнучкості: зміни в одному модулі не вимагають масштабного переписування решти системи, що значно прискорює цикл розробки та мінімізує ризик помилок. Крім того, централізація логіки взаємодії полегшує процес модульного тестування, оскільки кожен компонент можна тестувати окремо з чітко визначеними вхідними та вихідними параметрами. Це відкриває широкі можливості для масштабованості – нові функціональні можливості, такі як додаткові типи сповіщень, розширена аналітика або персоналізація інтерфейсу, можна швидко інтегрувати та не порушуючи загальної стабільності та узгодженості архітектури застосунку..

На рисунку 2.7 представлена UML-діаграма класів системи, яка відображає структуру основних компонентів та їх взаємозв'язки.

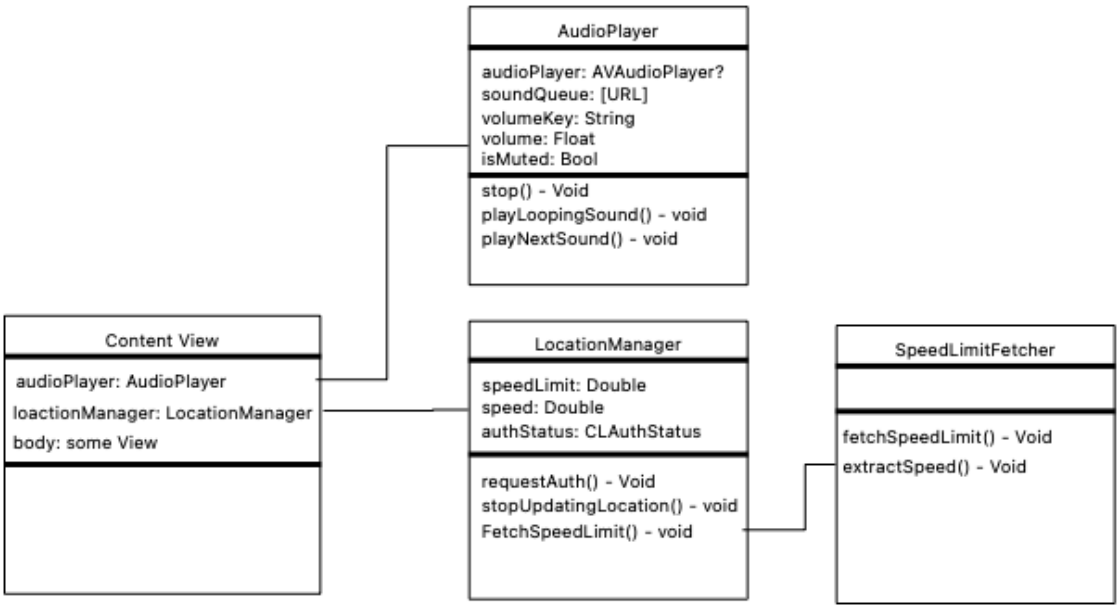

Рисунок 2.7 — UML-діаграма класів системи сповіщення про наближення до швидкісного ліміту

UML-діаграма представляє архітектуру ключових класів системи контролю швидкості з опорою на геолокацію, а також взаємозв'язки між ними. Головним елементом виступає клас ContentView, що відповідає за інтерфейс користувача та координує діяльність інших компонентів. У його складі містяться об'єкти audioPlayer і locationManager, котрі забезпечують відповідно відтворення звукових сповіщень та відстеження геолокаційних даних. Через цей клас реалізується графічне відображення поточної інформації, як-от швидкості руху та перевищення ліміту.

Клас AudioPlayer реалізує логіку керування аудіосигналами. Він утримує екземпляр системного об'єкта AVAudioPlayer, чергу аудіофайлів, значення гучності, стан приглушення та ключ доступу до гучності. Методи класу передбачають зупинку звуку, відтворення звукового сигналу в циклі або вибір наступного елемента з черги. Цей модуль тісно інтегрований з інтерфейсом користувача, дозволяючи змінювати стан звукового оповіщення залежно від дій користувача або змін у швидкості.

LocationManager виконує основні функції зчитування координат пристрою, розрахунку швидкості пересування, зберігання поточного швидкісного ліміту та статусу авторизації доступу до геолокації. Він містить методи для запиту дозволу, припинення оновлення координат та отримання швидкісного ліміту. При зверненні до бази даних швидкісних обмежень LocationManager використовує об’єкт класу SpeedLimitFetcher, що відповідає за обробку відповідних запитів та повернення валідних значень швидкісного ліміту.

Клас SpeedLimitFetcher виступає як інтерфейс до зовнішнього джерела даних (наприклад, OpenStreetMap), забезпечуючи завантаження інформації про швидкісні ліміти для конкретного географічного положення. Він містить методи для виконання запиту (fetchSpeedLimit) та обробки отриманих результатів (extractSpeed).

Загальна структура побудована на принципі централізованого управління, де ContentView ініціалізує всі ключові модулі та забезпечує їх взаємодію. Такий підхід сприяє підтримці цілісності програми, дозволяючи ефективно обробляти події в реальному часі, оновлювати інтерфейс користувача та своєчасно реагувати на зміну швидкості або перевищення встановленого ліміту.

2.3 Висновок до розділу 2

У результаті виконаної роботи були визначено, що розробка додатку, котрий інформує про перевищення швидкісного ліміту, спирається на добре організовану та взаємопов’язану структуру. Вона поєднує низку ключових компонентів: інтерфейс користувача, модуль геолокації, систему звукових сповіщень, базу даних обмежень швидкості та модуль персональних налаштувань. Кожен із цих модулів виконує свою унікальну функцію, але всі

вони працюють узгоджено, гарантуючи стабільність, гнучкість та безперервний обмін інформацією.

Центральний елемент взаємодії — інтерфейс, розроблений за допомогою SwiftUI. Він дозволяє в реальному часі відстежувати зміни швидкості та негайно надавати водієві візуальні та аудіальні сповіщення. Об’єктно-орієнтований підхід до організації коду забезпечує модульність та спрощує розширення системи. Використання UserDefaults для збереження налаштувань гарантує персоналізацію без складних залежностей.

Алгоритм роботи модуля охоплює всі критично важливі етапи — від ініціалізації до реагування на перевищення швидкості, гарантуючи надійну роботу в умовах постійно мінливої дорожньої обстановки. Завдяки чіткій взаємодії між модулями, система здатна швидко обробляти геолокаційні дані, порівнювати швидкість з обмеженнями та безпечно повідомляти користувача про необхідність зниження швидкості. Цей підхід не тільки підвищує ефективність контролю за дотриманням швидкісного режиму, а й сприяє зменшенню кількості ДТП, зберігаючи зручність та мінімізуючи відволікання водія під час руху.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ СПОВІЩЕННЯ ПРО ПЕРЕВИЩЕННЯ ШВИДКОСТІ

3.1 Обґрунтування вибору мови програмування

Для реалізації додатку для оповіщення про наближення до швидкісного ліміту на платформі iOS було обрано мову програмування Swift. Обґрунтуємо це рішення, проаналізувавши основні переваги та недоліки цієї мови, а також вказавши на її унікальні особливості.

Під час вибору мови програмування враховувалися особливості цільової платформи (iOS), вимоги до створення швидкого та зручного графічного інтерфейсу, необхідність взаємодії з GPS-модулем у режимі реального часу, потреба у точному та надійному сповіщенні водія про перевищення швидкості, забезпечення адаптивного та інтуїтивного інтерфейсу для водіїв, підтримка сучасних функцій безпеки та продуктивності, забезпечення стабільної роботи додатку без значного навантаження на систему [11-14].

Основними критеріями оцінки були зручність реалізації об'єктно-орієнтованої архітектури, ефективність роботи з системними API та простота подальшого обслуговування коду.

Розглядалися два основні варіанти: мова програмування Swift [15] та мова Objective-C.

Ключові переваги Swift:

1. Сучасна парадигма розробки Swift підтримує сучасні підходи до проектування програм, включаючи функціональне та реактивне програмування, що спрощує реалізацію таких речей, як обробка даних GPS за допомогою Combine або Swift Concurrency (async/await).

2. Безпечна система типів Swift запобігає багатьом поширеним помилкам під час виконання за допомогою додаткових системних умов, умов

захисту та механізмів безпечного розпакування. Це зменшує ймовірність збоїв під час роботи з даними датчиків або API.

3. Висока продуктивність Swift має порівнянну з C/C++ продуктивність завдяки компіляції в власний код. Це забезпечує стабільну обробку даних у режимі реального часу без затримок під час отримання координат та обчислення швидкості.

4. Легка інтеграція з системами Apple Swift спрощує використання CoreLocation (визначення координат та швидкості пересування пристрою), AVFoundation (для звукових сповіщень) та інших фреймворків, необхідних для роботи програми.

5. Підтримка сучасних інтерфейсів За допомогою SwiftUI ви можете створювати адаптивні, гнучкі інтерфейси, що адаптуються до умов руху (великі шрифти, мінімалістичний дизайн, темна тема тощо).

6. Можливість використання Swift Package ManagerSwift має вбудовану систему керування залежностями, яка спрощує підключення сторонніх бібліотек, наприклад, для роботи з аналітикою, картами або телеметрією [11-14].

Недоліки мови Swift:

1. Молодість мови - незважаючи на швидкий розвиток, Swift все ще оновлюється. При перемиканні між версіями можливі несумісності, що вимагає коригування коду [11-14].

2. Складність підтримки старих системЯкщо потрібна підтримка дуже старих версій iOS (до iOS 9), Swift може мати обмеження або вимагати обгорток Objective-C.

Для порівняння з Swift було також розглянуто мову програмування Objective-C як альтернативний варіант реалізації додатку. Objective-C є потужна мова програмування, орієнтована на об'єкти, яку широко застосовують у розробці застосунків для macOS та iOS. Вона є надмножиною мови C, тобто включає в себе всі її функції, додаючи об'єктно-орієнтовані можливості та динамічне середовище виконання. [11-14].

Основні переваги мови програмування Objective-C:

1. Стабільність та перевірена довговічність Objective-C використовується в розробці для iOS з самого початку. Величезна база коду, приклади та документація, особливо для старіших API.

2. Сумісність з C/C++ Дозволяє працювати з низькорівневими функціями, наприклад, для дрібнозернистого управління пам'яттю, реалізації користувацьких обчислень тощо.

3. Повна інтеграція з усіма фреймворками Apple Objective-C історично був основною мовою для iOS SDK, тому всі фреймворки мають повну підтримку.

Недоліки Objective-C:

1. Складний синтаксис Нестандартна система виклику методів (наприклад, [object methodWithParam:value]) ускладнює читання, особливо для нових розробників.

2. Менша кількість сучасних мовних конструкцій Відсутність вбудованої підтримки опцій, async/await, функціонального стилю - все це робить реалізацію алгоритмів менш безпечною та більше схожою на шаблон.

3. Менша активність спільноти Apple та розробників зосереджена переважно на мові Swift, тому більшість нових рішень, бібліотек та фреймворків реалізовані спеціально для мови Swift.

Після ретельного дослідження, Swift було обрано як основну мову програмування для впровадження програмного забезпечення iOS. Це оптимальний вибір завдяки сучасному синтаксису, універсальності, високій швидкості роботи та щільній взаємодії з екосистемою Apple. Не дивлячись на сильні сторони Objective-C, такі як надійність та сумісність з вже існуючими проектами, Swift є більш перспективною мовою для розробки нової програми, яка використовує поточні можливості платформи iOS. Зокрема, Swift дозволяє оперативно розробити систему обробки даних GPS, генерувати сповіщення про обмеження швидкості та створювати інтуїтивно зрозумілий інтерфейс для

користувача - водія, що безпосередньо сприяє покращенню безпеки дорожнього руху.

3.2 Програмна реалізація iOS додатку для оповіщень про наближення до швидкісного ліміту

Додаток для сповіщення про обмеження швидкості реалізовано на основі попередньо розробленої архітектури, яка враховує вибрані технологічні рішення. Для розробки було обрано мову програмування Swift у поєднанні зі SwiftUI [17], що забезпечує сучасний та декларативний підхід до побудови інтерфейсу. Використання SwiftUI дозволяє ефективно впроваджувати оновлення інтерфейсу в режимі реального часу, що є критично важливим для надання водієві відповідних сповіщень про перевищення або наближення до встановленого обмеження швидкості. Такий підхід не лише покращує взаємодію з користувачем, але й підвищує загальну безпеку дорожнього руху.

Основна структура проекту складається з шести основних файлів, кожен з яких забезпечує коректну функціональність поодиноких складових системи:

TrueChimeApp.swift – головний файл запуску програми

SpeedometerView.swift – реалізація головного вікна графічного інтерфейсу додатку

SettingsView.swift – реалізація допоміжного вікна графічного інтерфейсу додатку

LocationManager.swift – файл що відповідає за роботу з бібліотекою CoreLocation [19]

SoundPlayer.swift – файл що відповідає за роботу з бібліотекою AVFoundation [20] та роботу звукових оповіщень

SpeedLimitFetcher.swift – файл що відповідає за отримання інформації про швидкісні ліміти з бази даних OpenStreetMap

Файл «TrueChimeApp.swift» є точкою входу в додаток. Даний файл містить мінімальну кількість коду необхідну для конфігурації системи та виклику головного вікна програми.

Файл «SpeedometerView.swift» є реалізацією головного вікна додатку. Саме у ньому відбувається порівняння швидкості зі швидкісним лімітом, викликаються менеджери для отримання інформації або для ввімкнення та вимкнення звукових оповіщень.

Файл «SettingsView.swift» є другорядною, але не менш важливою частиною графічного інтерфейсу. Його основна задача надати користувачу можливість налаштувати додаток відповідно до його комфорту. Наприклад гучність оповіщень, або ввімкнення жорсткого ліміту швидкості та похибку швидкості на включення сповіщень.

Файл «LocationManager.swift» містить клас LocationManager, який відповідає за конфігурацію та управлінням доступу до геолокації користувача.

Файл «SoundPlayer.swift» містить клас SoundPlayer. Даний клас є необхідним для правильного налаштування виводу звукових сповіщень з урахуванням налаштувань заданих користувачем.

Файл «SpeedLimitFetcher.swift» містить клас SpeedLimitFetcher котрий забезпечує ледь не найважливішу складову додатку, а саме отримання швидкісних лімітів з відкритої бази даних та конвертація отриманих даних у формат потрібний для подальшої їх обробки.

Розглянемо файл «TrueChimeApp.swift» котрий є основою для запуску і подальшої коректної роботи додатку, хоч і містить мінімальну кількість коду:

```
import SwiftUI
@main
struct TrueChimeApp: App {
    var body: some Scene {
        WindowGroup {
            SpeedometerView()
        }
    }
}
```

Цей код показує, як легко запустити iOS-додаток за допомогою Swift та SwiftUI. Структура TrueChimeApp, позначена атрибутом @main, визначає точку входу до додатка. Основний інтерфейс реалізовано як WindowGroup, яка автоматично керує життєвим циклом вікна. Усередині WindowGroup ініціалізовано основне представлення інтерфейсу — SpeedometerView, яке містить логіку відображення швидкості та сповіщення користувача про наближення до обмеження швидкості. Такий підхід дозволяє створити чисту, декларативну та сучасну структуру додатка, оптимізовану для iOS.

Головним центром дій системи є файл «SpeedometerView.swift». Саме ц ньому ми створюємо об’єкти класів менеджерів, прописуємо інтерфейс користувача та визначаємо подальші дії системи. Пропоную розглянути ключові аспекти змінної body структури SpeedometerView:

```
NavigationStack {
  VStack {
  }
}
```

Даний фрагмент коду описує так звані стаки у яких буде створюватись графічний інтерфейс, наприклад NavigationStack надає можливість використовувати NavigationLink для переходу на інші вікна додатку, а VStack означає що усі майбутні елементи будуть розташовані вертикально на вікні.

```
NavigationLink {
     SettingsView(soundPlayer: soundPlayer)
    } label: {
     Image(systemName: "gearshape")
      .resizable()
      .frame(width: 25, height: 25)
      .foregroundStyle(.gray)
    }
   }
```

Ось і саме використання NavigationLink, екраном який відкриватиметься даною процедурою буде SettingsView, а графічним компонентом який бачитиме користувач буде сіра шестерня.

```
HStack(alignment: .bottom) {
```

```
        ForEach(0 ..< 4) { i in
         if i == 3 {
          Rectangle()
            .foregroundStyle(.red)
            .frame(width: 4, height: 4)
         }

         if let symbol = segmentedSpeed[i] {
          SegmentedDisplay(segments:
segmentsForDigit(symbol), thickness: 4, length: 10)
            .padding()
         } else {
          SegmentedDisplay(thickness: 4, length: 10)
            .padding()
         }
        }
       }
```

У цьому блоці ми відображатимемо швидкість за допомогою чотирицифрового семисегментного дисплею створеного окремо в додатку.

```
.onAppear() {
   locationManager.requestAuthorization()
   if hardSpeedLimitOn {
    segmentedSpeedLimit                                =
convertTo4DigitArray(Double(hardSpeedLimitValue)  ??
50.0)
   }
  }
```

Однак тут починається ключовий функціонал структури. Дана функція зазначає що при появі NavigationView оголошеного раніше на екрані пристрою буде викликані наступні дії: запит на використання геолокації та перевірка наявності дозволів, а також перевірка на встановлений жорсткий швидкісний ліміт користувачем.

```
.onChange(of:      locationManager.speedInKmh)      {
oldValue, newValue in
```

```
    withAnimation() {
     segmentedSpeed = convertTo4DigitArray(newValue)
    }

    if Int(newValue) - 1 > (hardSpeedLimitOn ?
(Int(hardSpeedLimitValue) ?? 50) :
locationManager.speedLimit) +
Int(speedLimitMargin)! && !isChimeOn {
     soundPlayer.playLoopingSound(named:
[soundName])
     isChimeOn = true
    } else {
     soundPlayer.stop()
     isChimeOn = false
    }

    if newValue > 130 {
     withAnimation() {
      isFastAsHell = true
     }
    } else {
     withAnimation() {
      isFastAsHell = false
     }
    }
   }
   .padding()
   .background {
    Color.black
     .ignoresSafeArea()
```

```
    }
```

Тут відображений один з викликів функції .onChange яка дозволяє спостерігати за змінними класів котрі відповідають протоколу ObservableObject. Завдяки цьому при зміні значення певної змінної буде викликана певна дія. У цьому випадку ми отримуємо інформацію про зміну швидкості користувача, оновлюємо графічний інтерфейс і показання спідометра стрілкового та цифрового і за необхідності включаємо або виключаємо звукові сповіщення.

Наступним розглянемо основні елементи класу LocationManager який керує всіма процесами пов’язаними з геолокацією користувача:

```
@Published var speedInKmh: Double = 0.0
```

Так виглядає оголошення змінної, яка надаватиме активну інформацію про зміну свого значення.

```
override init() {
  super.init()
     locationManager.delegate = self
locationManager.desiredAccuracy                    =
kCLLocationAccuracyBest
locationManager.activityType                       =
.automotiveNavigation
     locationManager.allowsBackgroundLocationUpdates
= true
locationManager.pausesLocationUpdatesAutomatically
= false
 }
```

Дана функція є ініціалізатором класу який є необхідним для правильного початкового налаштування об’єкту, наприклад він надає йому право перевіряти геолокацію користувача навіть у той час коли екран пристрою заблоковано.

```
private func fetchSpeedLimit(for coordinate:
CLLocationCoordinate2D) {
  speedLimitFetcher.fetchSpeedLimit(for:
coordinate) { [weak self] limit in
   self?.speedLimit = limit
  }
 }
```

В той як я ця частина коду за допомогою об’єкту класу SpeedLimitFetcher отримує дані про швидкісний ліміт за координатами користувача.

Крім вказаних методів клас LocationManager також містить функціонал для отримання швидкості користувача у кілометрах на годину через встановлений проміжок часу, метод отримання та перевірки наявності дозволів на використання геолокації.

Наступним ми розглянемо головні елементи класу SoundPlayer:

```
var audioPlayer: AVAudioPlayer?
 var soundQueue: [URL] = []
 private let volumeKey = "SoundPlayerVolume"
 var volume: Float {
  get {
   let stored = UserDefaults.standard.float(forKey:
volumeKey)
   return stored == 0 ? 1.0 : stored
  }
  set {
   let clampedVolume = max(0.0, min(newValue, 1.0))
   UserDefaults.standard.set(clampedVolume, forKey:
volumeKey)
   if !isMuted {
    audioPlayer?.volume = clampedVolume
   }
```

```
    }
  }
  var isMuted: Bool = false {
    didSet {
      audioPlayer?.volume = isMuted ? 0.0 : volume
    }
  }
```

Це змінні котрі будуть використовуватись у класі. А саме аудіоплеєр для спрощеного використання бібліотеки AVFoundation, черга звуків, у яку ми здаватимемо звук для сповіщень, змінна гучності яка автоматично змінюється в залежності від користувацьких налаштування та змінної isMuted котра є булевою змінною і відповідає за те чи виключив користувач тимчасово оповіщення

```
  private func playNextSound() {
    guard !soundQueue.isEmpty else { return }

    let soundURL = soundQueue[0]
    do {
      audioPlayer = try AVAudioPlayer(contentsOf:
  soundURL)
      audioPlayer?.delegate = self
      audioPlayer?.volume = isMuted ? 0.0 : volume
      audioPlayer?.prepareToPlay()
      audioPlayer?.play()
    } catch {
      print("Error playing sound:
  \(error.localizedDescription)")
    }
  }
```

Дана функція відповідає безпосередньо за відтворення звуку, вона викликається у SpeedometerView у випадку коли швидкість користувача перевищує ліміт.

Також цей клас містить функції для конфігурації аудіосесії, що дозволяє відтворювати звук без виключення або приглушення музики з інших додатків та відтворення звуків через зовнішні динаміки та із заблокованим екраном пристрою, функцію початку гри сповіщень та функцію для припинення відтворення сповіщень.

Не менш важливими є клас SpeedLimitFetcher, який містить функції для відправки запитів на API бази даних, отримання відповіді та серіалізації JSON у змінні. Ще одним ключовим файлом є SettingsView, який забезпечує збереження користувацьких налаштувань у постійну пам’ять за допомогою UserDefaults.

Впроваджена система гарантує повний функціонал додатку для сповіщень про наближення до швидкісного ліміту, ефективно інтегруючи складне збереження та обробку даних зі зручним візуальним інтерфейсом та звуковими сповіщеннями. Модульна структура дає змогу легко масштабувати функціональність системи та додавати нові можливості без зміни поточної логіки.

3.3 Тестування роботи iOS додатку для оповіщень про наближення до швидкісного ліміту

Під час тестування додатку для оповіщень про наближення до швидкісного ліміту необхідно перевірити наявність звукових сповіщень, коректність відображення інформації на головному екрані, можливість приглушення звуку, зберігаємість користувацьких налаштувань. Тестування проводилося поетапно з перевіркою всіх основних функціональних можливостей програми.

Для запуску додатку спочатку необхідно встановити його на пристрій, це виконується або безпосередньо розробником або за допомогою сервісів як AppStore та TestFlight. Надалі запуск відбувається як і з будь яким іншим застосунком на телефон.

Одразу після запуску відкривається головне вікно програми. Дизайн та саме вікно зображені на рисунку 3.1.

Рисунок 3.1 – Загальний вигляд інтерфейсу головного вікна додатку

Тестуватимемо ми відображення швидкісних лімітів, відображення швидкості та наявність звукових сповіщень у випадку перевищення швидкості. Для цього ми скористаємось вбудованою функцією симулятору, котра дозволяє змінити геолокацію на встановлені пресети, такі як фіксована

геолокація, прогулянка містом, поїздка велосипедом або швидкісним шосе. Ми використаємо останні три опції. Результати тестування зображені на рисунку 3.2.

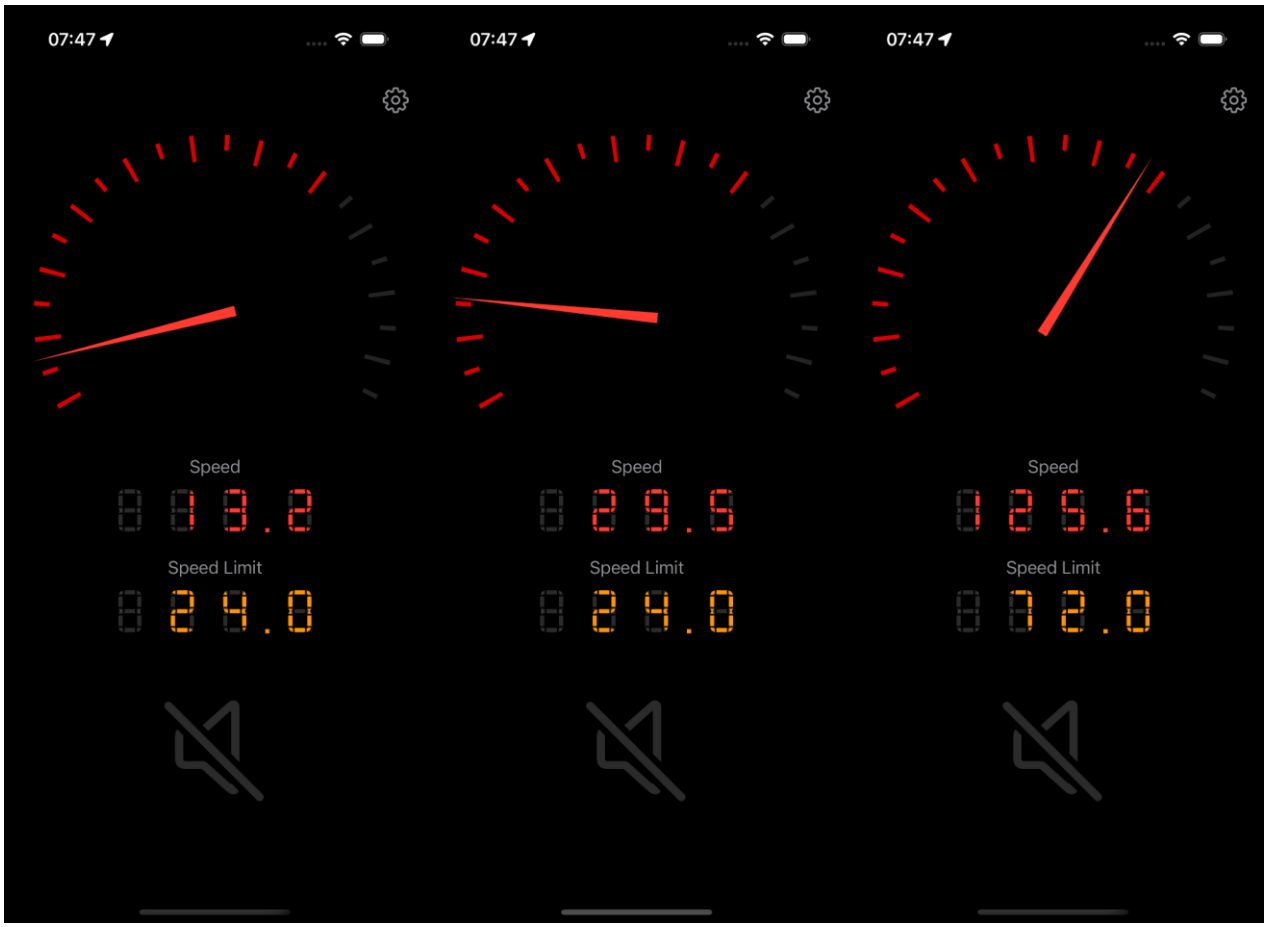

Рисунок 3.2 – Результат тестування швидкості та ліміту швидкістю за допомогою вбудованих функцій симулятору

Як ми можемо спостерігати швидкісний ліміт для пішого ходу чи поїздки велосипедом у пішохідній зоні відрізняється від шосе, так само як і сама швидкість пересування.

Наступним етапом увімкнемо звукові сповіщення (рис. 3.3) і чекатимемо на момент коли швидкість пересування перевищує ліміт швидкості.

Рисунок 3.3 – Знімок екрану з включеними сповіщеннями і перевищенням швидкісного ліміту

Як ми можемо спостерігати оповіщення присутні, на жаль показати їх у текстовому варіанті неможливо, за допомогою інтерфейсу додатку, однак звернемось до консолі середовища розробки для цього, а саме додамо тимчасовий виклик функції `print()` з виведенням слова “ding” кожен раз як викликається функція програвання звуку (рис. 3.4).

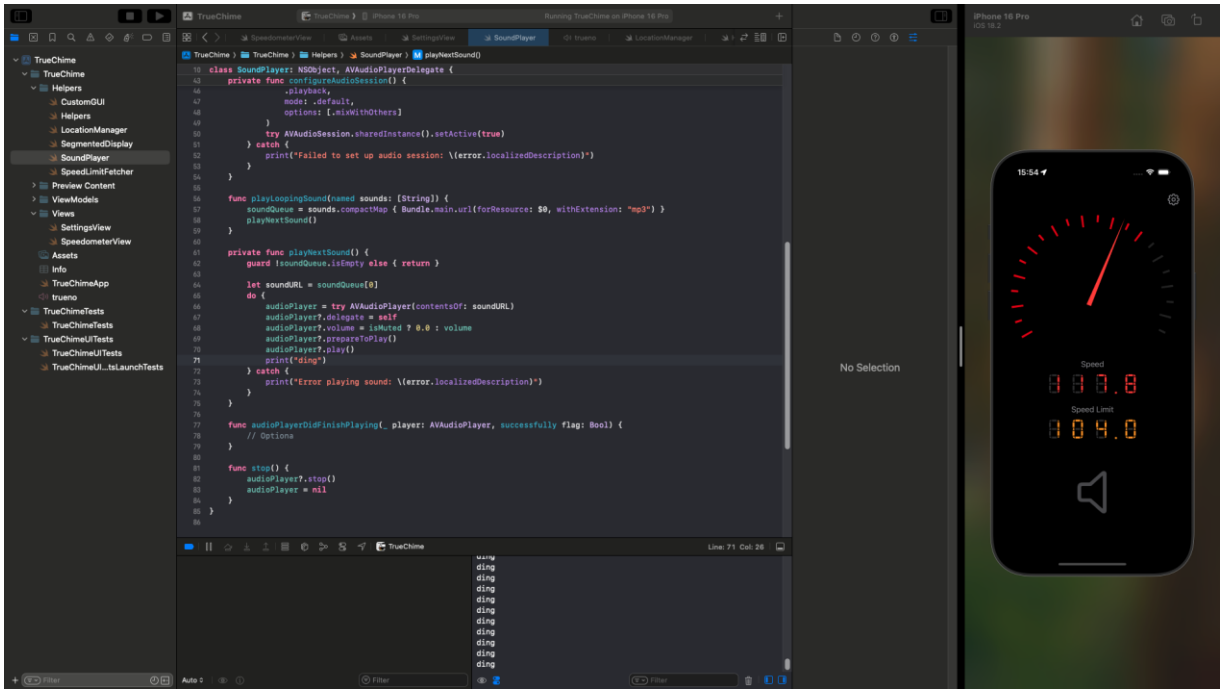

Рисунок 3.4 – Результат функціонування додатку з тимчасовими змінами для відображення увімкнення звукового сигналу

Наступним етапом буде перевірка роботи проміжку між різницею швидкісного ліміту і порогу включення звукових сповіщень. Оскільки додаток надихався JDM дзвіночком, а він спрацьовував при перевищенні ліміту на 10 км/год [10], таке ж значення є стандартним, що ми можемо перевірити у екрані налаштувань (рис. 3.5)

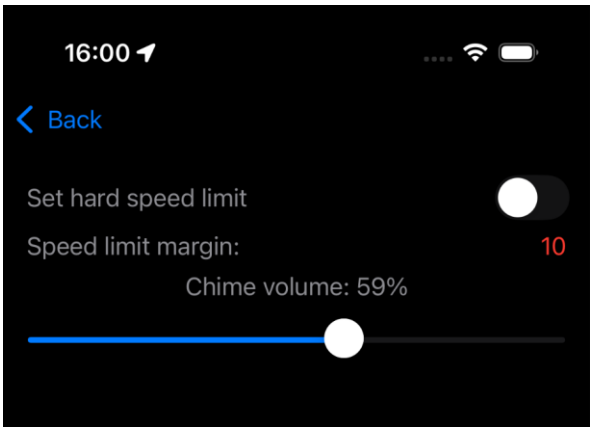

Рисунок 3.5 – Загальний вигляд інтерфейсу вікна налаштувань додатку з порогом спрацювання дзвіночка у 10 км/год

Змінимо значення порогу на 25 км/год та перевіримо чи будуть наявні сповіщення при тій самій швидкості, що і з порогом у 10 км/год (рис. 3.6).

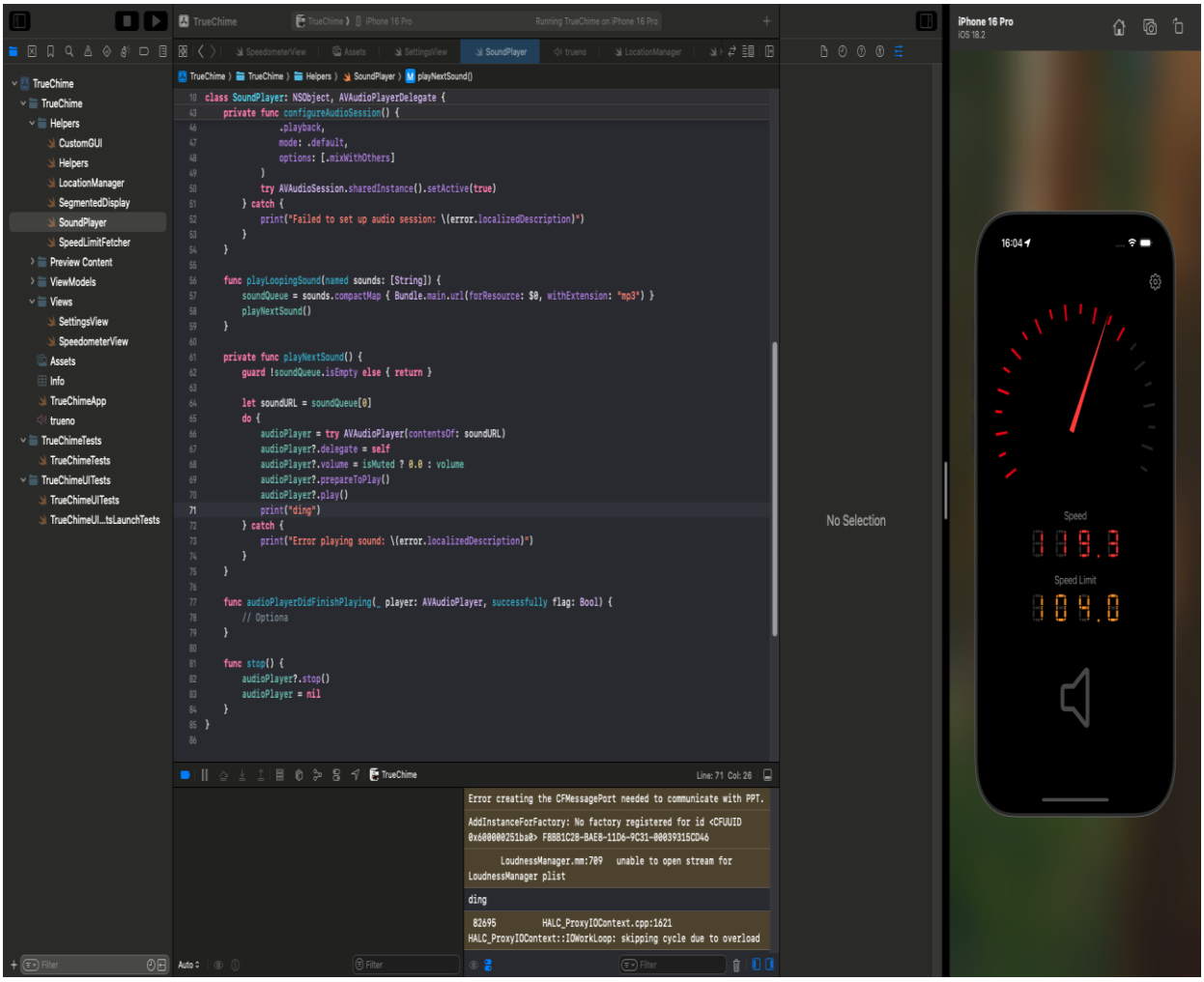

Рисунок 3.6 – Результат роботи програми при встановленому порогу сповіщень у 25 км/год

Як ми можемо спостерігати, звукові сповіщення майже відсутні загалом, і повністю відсутні на момент скріншоту.

Наступним перевіримо функціонал встановлення жорсткого швидкісного ліміту, перевірятимемо ми знов за допомогою вбудованої функції симулятора вказаної раніше (рис. 3.7). У якості обраних показників ми встановимо швидкісний ліміт у 50 км/год, та поріг спрацювання у 10 км/год.

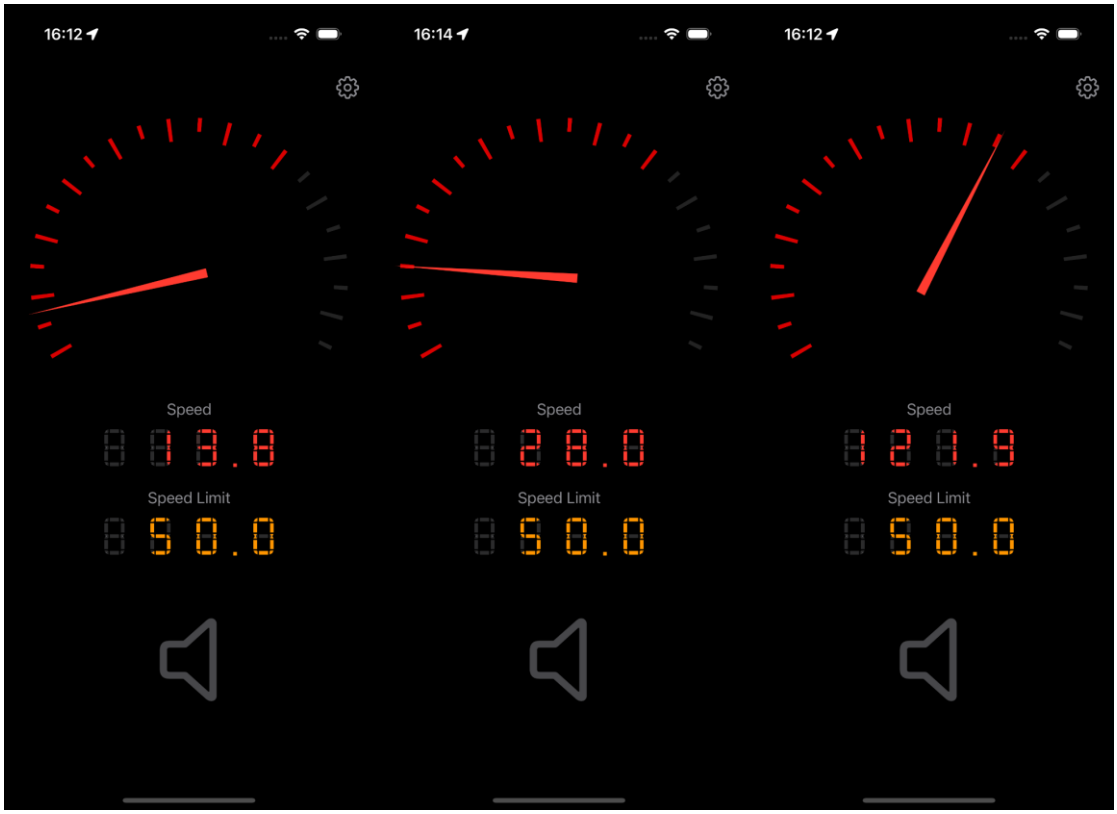

Рисунок 3.7 – Результат тестування швидкості та ліміту швидкістю за допомогою вбудованих функцій симулятору при встановленні жорсткого швидкісного ліміту

Як ми бачимо у результатах тестування встановлення жорсткого швидкісного ліміту працює.

Також було проведене тестування у реальних умовах. Для тестування було обрано авто Renault ZOE оскільки це авто має достовірний цифровий спідометр який надає точні показники і не залишає місця для інтерпретації швидкості та додаток Waze для порівняння із спідометром на основі системи GPS. В результаті тестування було визначено, що швидкість показана додатком розробленим у ході виконання дослідження співпадає з іншими спідометрами на системи GPS однак відрізняється від показуваної швидкості автомобіля приблизно на 5-10 км/год. З урахуванням цих даних не рекомендовано встановлювати поріг сповіщень з точністю до кілометра, оскільки є місце для похибки у GPS системі.

Тестування програмного забезпечення для сповіщення про обмеження швидкості показало його правильну роботу відповідно до заявлених функцій. Усі важливі частини, а саме: розпізнавання швидкості, визначення обмеження швидкості, відтворення звукових сповіщень, збереження та застосування користувацьких налаштувань, функціонують бездоганно. Програма виявляє стабільну поведінку як в середовищі емуляції, так і під час тестів у реальних умовах. Виявлені похибки, зокрема різниця у показниках швидкості між програмою та панеллю приладів автомобіля, є характерними для GPS-систем і беруться до уваги при налаштуванні рекомендованого порогу для звукових сигналів. Здобуті результати дозволяють стверджувати про придатність програмного забезпечення до практичного використання та його ефективність в умовах реального дорожнього руху.

3.4 Висновок до розділу 3

У третьому розділі детально описується процес втілення програмного забезпечення iOS-додатку для сповіщення водія про перевищення ліміту швидкості. Обґрунтовано вибір мови програмування Swift як основного інструменту розробки, беручи до уваги її переваги над альтернативними підходами. Зокрема, було встановлено, що Swift пропонує високу продуктивність, безпеку типів, зручність взаємодії з системними фреймворками Apple, а також сучасний підхід до створення інтерфейсів за допомогою SwiftUI.

Було реалізовано архітектуру застосунку, що базується на окремих модулях, кожен з яких відповідає за певну функцію: користувацький інтерфейс, роботу з даними геолокації, обробку інформації про обмеження швидкості та відтворення звукових сповіщень. Така модульна структура полегшує обслуговування, тестування та розширення проекту в майбутньому. Застосування CoreLocation, AVFoundation та OpenStreetMap API дало

можливість взаємодіяти в реальному часі з GPS-модулем пристрою, швидко визначати поточну швидкість користувача та зіставляти її з дозволеним обмеженням.

Центральною особливістю втілення є застосування реактивного підходу, що забезпечує динамічне оновлення інтерфейсу залежно від змін даних. Індикатор швидкості зроблено у вигляді семисегментного цифрового дисплея, що робить інформацію легкою для сприйняття під час руху. До того ж, програма дає змогу гнучко налаштовувати параметри, враховуючи звукові сигнали, чітке обмеження швидкості та поріг перевищення, що покращує комфорт використання в різних дорожніх умовах.

Внаслідок розробки створено стабільний прототип мобільного додатку, що поєднує інтуїтивно зрозумілий інтерфейс з необхідним функціоналом для своєчасного сповіщення водія про перевищення швидкості. Це, своєю чергою, сприяє підвищенню безпеки дорожнього руху та дозволяє користувачам краще контролювати свій стиль водіння.

ВИСНОВКИ

Поставлені перед бакалаврською кваліфікаційною роботою задачі виконано в повному обсязі [21], а саме:

1. Проведено аналіз предметної області моніторингу швидкості транспортних засобів і швидкісних обмежень на автомобільних шляхах.
2. Вивчено існуючі методи, алгоритми та програмні інструменти для відстеження швидкості руху та контролю за її перевищенням.
3. Розроблено архітектуру iOS додатку для сповіщення про наближення до швидкісного ліміту.
4. Визначено взаємодію програмних модулів системи, включаючи геолокаційний менеджер, модуль звукових сповіщень, систему налаштувань та інтерфейс користувача.
5. Здійснено програмну реалізацію додатку за допомогою мови програмування Swift із використанням SwiftUI.
6. Проведено тестування програмного модуля в середовищі симулятора та в реальних умовах, яке підтвердило правильність функціонування системи.
7. Підготовлено рекомендації щодо коректного використання додатку, з урахуванням можливих похибок GPS-системи.

У першому розділі визначається важливість проблеми перевищення швидкості як ключового фактора аварійності на дорогах. Детально проаналізовано наявні пристрої та технології для вимірювання швидкості та сповіщення про її перевищення. Розглянуто плюси та мінуси сучасних мобільних додатків, що виконують схожі функції.

У другому розділі представлено архітектуру майбутнього додатку, з акцентом на гнучкості, можливості розширення та зручності користування. Створено структурні та UML-діаграми системи, а також описано механізми

взаємодії між програмними модулями. Логіка роботи додатку формалізована у вигляді алгоритмів.

У третьому розділі реалізовано програмне забезпечення додатку з використанням Swift та SwiftUI, реалізовано основні функціональні блоки: показ поточної швидкості, розпізнавання обмеження швидкості на ділянці, формування звукових сповіщень, налаштування порогового значення для спрацювання та забезпечення жорсткого обмеження швидкості. Результати тестування в симуляторі та на реальних дорогах продемонстрували ефективність та правильність роботи системи.

Мета дослідження розширення функціональних можливостей додатків для оповіщення про наближення до швидкісного ліміту досягнута за рахунок того, що розроблений додаток отримує швидкісні ліміти за геопозицією користувача у реальному часі, містить цифровий та стрілочний спідометри, є простим у використанні та має звукові сповіщення про перевищення швидкості. Також він успішно функціонує в умовах реального дорожнього руху, має інтуїтивно зрозумілий інтерфейс та високу адаптивність. Отримані результати можуть слугувати базою для подальшого вдосконалення функціоналу або інтеграції додатку в комплексні системи допомоги водіям.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кириленко В.В. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОПОМОГИ ВОДІЄВІ в Матеріалах конференції «LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025)», Вінниця, 2025. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/24090/19873
2. Правила дорожнього руху 2025 31. Технічний стан транспортних засобів та їх обладнання [Електронний ресурс] – Режим доступу до ресурсу: https://vodiy.ua/pdr/31/
3. В Україні на чверть зросла кількість ДТП та піднявся рівень смертності [Електронний ресурс] – Режим доступу до ресурсу: https://suspilne.media/670834-v-ukraini-na-cvert-zrosla-kilkist-dtp-ta-pidnavsa-riven-smertnosti/
4. Impact Speed and a Pedestrian's Risk of Severe Injury or Death [Електронний ресурс] – Режим доступу до ресурсу: https://aaafoundation.org/impact-speed-pedestrians-risk-severe-injury-death/
5. Вплив метеорологічних факторів на умови руху [Електронний ресурс] – Режим доступу до ресурсу: http://obr.cc.ua/page10.html
6. Як працює система контролю швидкості в країнах Європи – дослідження [Електронний ресурс] – Режим доступу до ресурсу: https://www.village.com.ua/village/life/edu-news/346189-yak-pratsyue-sistema-kontrolyu-shvidkosti-v-krayinah-evropi-doslidzhennya
7. Спідометр. Види і пристрій. Похибка і особливості [Електронний ресурс] – Режим доступу до ресурсу: https://uk.avtotachki.com/ spidometr-vidy-i-ustrojstvo-pogreshnost-i-osobennosti/#i-3
8. What is a Heads-Up Display? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ansys.com/simulation-topics/what-is-heads-up-display

9. Радар-детектори: особливості, види і принцип роботи [Електронний ресурс] – Режим доступу до ресурсу: https://130.com.ua/uk/radar-detektory-osobennosti-vidy-rabota/
10. Shimizu, Sōichi. Why did the speed chime disappear?. *MOTA* [Електронний ресурс] – Режим доступу до ресурсу: https://autoc-one.jp/word/408033/
11. Swift and Objective-C: An In-Depth Comparison of iOS Programming Languages [Електронний ресурс] – Режим доступу до ресурсу: https://shakuro.com/blog/swift-vs-objective-c
12. What is Objective-C? [Електронний ресурс] – Режим доступу до ресурсу: https://artoonsolutions.com/glossary/objective-c/
13. Swift Pros and Cons[Електронний ресурс] – Режим доступу до ресурсу: https://www.netguru.com/blog/building-ios-app-with-swift
14. What is Objective-C? [Електронний ресурс] – Режим доступу до ресурсу: https://www.devopsschool.com/blog/what-is-objective-c-and-use- cases-of-objective-c/
15. The Swift Programming Language (6.2) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.swift.org/swift-book/documentation /the-swift-programming-language/
16. Driver Characteristics and Speeding Behaviour [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/228660490_Driver_Characteristics_and_Speeding_Behaviour
17. SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swiftui/
18. UserDefaults [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/foundation/userdefaults
19. Core Location [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/corelocation
20. AVFoundation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/av-foundation/
21. А. А. Яровий, О. В. Сілагін, І. В. Богач. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» Вінниця : ВНТУ, 2023. 54 с.

ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка iOS додатку для оповіщень про наближення до швидкісного ліміту

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,34 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

- У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Сілагін Є.О.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Кириленко В.В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (ОБОВ'ЯЗКОВИЙ)
ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМНОГО КОДУ

```
import SwiftUI
struct SpeedometerView: View {
    static var fadeInOut: AnyTransition {
        .asymmetric(
            insertion: .opacity,
            removal: .opacity
        )
    }

    @StateObject var locationManager = LocationManager()
    @AppStorage("soundName") var soundName: String = "trueno"

    @AppStorage("HardSpeedLimitOn") private var hardSpeedLimitOn: Bool = false
    @AppStorage("HardSpeedLimitValue") private var hardSpeedLimitValue: String
= "50"
    @AppStorage("SpeedLimitMargin") private var speedLimitMargin: String = "10"

    @State var isFastAsFuck: Bool = false
    @State var isChimeOn: Bool = false

    @State var currentDigit: Int = 0
    @State var segmentedSpeed: [Int?] = [nil, nil, 0, 0]
    @State var segmentedSpeedLimit: [Int?] = [nil, nil, 0, 0]
    @State var speed: Double = 0

    @State var showUnauthoriedAlert = false
    @State var muteSound = false

    let soundPlayer = SoundPlayer()

    var body: some View {
        NavigationStack {
            VStack {
                HStack {
                    Spacer()

                    NavigationLink {
                        SettingsView(soundPlayer: soundPlayer)
                    } label: {
                        Image(systemName: "gearshape")
                            .resizable()
                            .frame(width: 25, height: 25)
                            .foregroundStyle(.gray)
                    }
                }

                Speedometer()

                VStack {
```

```
                    Text("Speed")
                        .foregroundStyle(.gray)

                    HStack(alignment: .bottom) {
                        ForEach(0 ..< 4) { i in
                            if i == 3 {
                                Rectangle()
                                    .foregroundStyle(.red)
                                    .frame(width: 4, height: 4)
                            }

                            if let symbol = segmentedSpeed[i] {
                                SegmentedDisplay(segments:
segmentsForDigit(symbol), thickness: 4, length: 10)
                                    .padding()
                            } else {
                                SegmentedDisplay(thickness: 4, length: 10)
                                    .padding()
                            }
                        }
                    }

                    Text("Speed Limit")
                        .foregroundStyle(.gray)
                        .padding(.top, 15)

                    HStack(alignment: .bottom) {
                        ForEach(0 ..< 4) { i in
                            if i == 3 {
                                Rectangle()
                                    .foregroundStyle(.orange)
                                    .frame(width: 4, height: 4)
                            }

                            if let symbol = segmentedSpeedLimit[i] {
                                SegmentedDisplay(segments:
segmentsForDigit(symbol), activeColor: Color.orange, thickness: 4, length: 10)
                                    .padding()
                            } else {
                                SegmentedDisplay(thickness: 4, length: 10)
                                    .padding()
                            }
                        }
                    }
                }
                .padding(.top, -60)

                Spacer()

                Image(systemName: muteSound ? "speaker.slash" : "speaker")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 100, height: 100)
                    .padding()
                    .foregroundStyle(.gray)
```

```
                    .scaleEffect(muteSound ? 1.1 : 0.9)
                    .opacity(muteSound ? 0.3 : 0.5)
                    .onTapGesture {
                        muteSound.toggle()
                    }

                Spacer()

            }
            .onAppear() {
                locationManager.requestAuthorization()
                if hardSpeedLimitOn {
                    segmentedSpeedLimit                                    =
convertTo4DigitArray(Double(hardSpeedLimitValue) ?? 50.0)
                }
            }
            .onChange(of: locationManager.speedLimit) { oldValue, newValue in
                withAnimation() {
                    if !hardSpeedLimitOn {
                        segmentedSpeedLimit                                =
convertTo4DigitArray(Double(newValue))
                    } else {
                        segmentedSpeed                                     =
convertTo4DigitArray(Double(hardSpeedLimitValue) ?? 50.0)
                    }
                }
            }
            .onChange(of: hardSpeedLimitOn) { oldValue, newValue in
                if newValue {
                    segmentedSpeedLimit                                    =
convertTo4DigitArray(Double(hardSpeedLimitValue) ?? 50.0)
                }
            }
            .onChange(of: hardSpeedLimitValue) { oldValue, newValue in
                if hardSpeedLimitOn {
                    segmentedSpeedLimit                                    =
convertTo4DigitArray(Double(hardSpeedLimitValue) ?? 50.0)
                }
            }
            .onChange(of: muteSound) { oldValue, newValue in
                soundPlayer.isMuted = newValue
            }
            .onChange(of: locationManager.speedInKmh) { oldValue, newValue in
                withAnimation() {
                    segmentedSpeed = convertTo4DigitArray(newValue)
                }

                if    Int(newValue)    -    1    >    (hardSpeedLimitOn    ?
(Int(hardSpeedLimitValue)    ??    50)    :    locationManager.speedLimit)    +
Int(speedLimitMargin)! && !isChimeOn {
                    soundPlayer.playLoopingSound(named: [soundName])
                    isChimeOn = true
                } else {
                    soundPlayer.stop()
                    isChimeOn = false
```

```
                }

                if newValue > 130 {
                    withAnimation() {
                        isFastAsFuck = true
                    }
                } else {
                    withAnimation() {
                        isFastAsFuck = false
                    }
                }
            }
            .padding()
            .background {
                Color.black
                    .ignoresSafeArea()
            }
            .alert(isPresented: $showUnauthoriedAlert) {
                Alert(
                    title: Text("Enable Location Access"),
                    message: Text("Please allow access to your location to use
the app’s features"),
                    primaryButton: .cancel(Text("Cancel")),
                    secondaryButton: .default(Text("Settings")) {
                        if        let        url        =        URL(string:
UIApplication.openSettingsURLString) {
                            UIApplication.shared.open(url)
                        }
                    }
                )
            }
        }
    }
}

class LocationManager: NSObject, ObservableObject, CLLocationManagerDelegate {
    private let locationManager = CLLocationManager()
    private let speedLimitFetcher = SpeedLimitFetcher()
    @Published var speedInKmh: Double = 0.0
    @Published var speedInMph: Double = 0.0
    @Published var speedLimit: Int = 50
    @Published var authorizationStatus: CLAuthorizationStatus = .notDetermined
    @Published var showUnauthorizedAlert: Bool = false
    private var lastFetchTime: Date? = nil
    private let fetchCooldown: TimeInterval = 60
    private let minDistanceChange: Double = 100
    private var lastFetchCoordinate: CLLocationCoordinate2D?
    override init() {
        super.init()
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.activityType = .automotiveNavigation
        locationManager.allowsBackgroundLocationUpdates = true
        locationManager.pausesLocationUpdatesAutomatically = false
    }
```

```
    func requestAuthorization() {
        let status = locationManager.authorizationStatus
        self.authorizationStatus = status
        if status == .notDetermined {
            locationManager.requestAlwaysAuthorization() //  Use "Always" for
background
        } else if status == .denied || status == .restricted {
            showUnauthorizedAlert = true
        }
    }
    func startUpdatingLocation() {
        locationManager.startUpdatingLocation()
    }
    func stopUpdatingLocation() {
        locationManager.stopUpdatingLocation()
    }
    // MARK: - CLLocationManagerDelegate
    func locationManagerDidChangeAuthorization(_ manager: CLLocationManager) {
        let status = manager.authorizationStatus
        self.authorizationStatus = status
        if status == .authorizedAlways {
            startUpdatingLocation()
        } else if status == .denied || status == .restricted {
            showUnauthorizedAlert = true
        }
    }
    func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
        guard let location = locations.last else { return }
        let speedInMetersPerSecond = location.speed
        withAnimation {
            if speedInMetersPerSecond >= 0 {
                speedInKmh = speedInMetersPerSecond * 3.6
                speedInMph = speedInMetersPerSecond * 2.23694
            } else {
                speedInKmh = 0
                speedInMph = 0
            }
        }
        // Throttle speed limit fetching
        let now = Date()
        let shouldFetch: Bool
        if let lastTime = lastFetchTime,
           let lastCoord = lastFetchCoordinate {
            let timePassed = now.timeIntervalSince(lastTime)
            let distance = location.distance(from: CLLocation(latitude:
lastCoord.latitude, longitude: lastCoord.longitude))
            shouldFetch = timePassed > fetchCooldown || distance >
minDistanceChange
        } else {
            shouldFetch = true
        }
        if shouldFetch {
            lastFetchTime = now
            lastFetchCoordinate = location.coordinate
```

```
            fetchSpeedLimit(for: location.coordinate)
        }
    }
    func locationManager(_ manager: CLLocationManager, didFailWithError error:
Error) {
        print("LocationManager failed with error:
\(error.localizedDescription)")
    }
    // MARK: - Speed Limit
    private func fetchSpeedLimit(for coordinate: CLLocationCoordinate2D) {
        speedLimitFetcher.fetchSpeedLimit(for: coordinate) { [weak self] limit
in
            self?.speedLimit = limit
        }
    }
}
class SoundPlayer: NSObject, AVAudioPlayerDelegate {
    var audioPlayer: AVAudioPlayer?
    var soundQueue: [URL] = []

    private let volumeKey = "SoundPlayerVolume"

    // Volume property (range: 0.0 to 1.0), persisted
    var volume: Float {
        get {
            let stored = UserDefaults.standard.float(forKey: volumeKey)
            return stored == 0 ? 1.0 : stored
        }
        set {
            let clampedVolume = max(0.0, min(newValue, 1.0))
            UserDefaults.standard.set(clampedVolume, forKey: volumeKey)
            if !isMuted {
                audioPlayer?.volume = clampedVolume
            }
        }
    }

    // In-memory mute flag
    var isMuted: Bool = false {
        didSet {
            audioPlayer?.volume = isMuted ? 0.0 : volume
        }
    }

    override init() {
        super.init()
        configureAudioSession()
    }

    private func configureAudioSession() {
        do {
            try AVAudioSession.sharedInstance().setCategory(
                .playback,
                mode: .default,
                options: [.mixWithOthers]
```

```
            )
            try AVAudioSession.sharedInstance().setActive(true)
        } catch {
            print("Failed to set up audio session: \(error.localizedDescription)")
        }
    }

    func playLoopingSound(named sounds: [String]) {
        soundQueue = sounds.compactMap { Bundle.main.url(forResource: $0, withExtension: "mp3") }
        playNextSound()
    }

    private func playNextSound() {
        guard !soundQueue.isEmpty else { return }

        let soundURL = soundQueue[0]
        do {
            audioPlayer = try AVAudioPlayer(contentsOf: soundURL)
            audioPlayer?.delegate = self
            audioPlayer?.volume = isMuted ? 0.0 : volume
            audioPlayer?.prepareToPlay()
            audioPlayer?.play()
            print("ding")
        } catch {
            print("Error playing sound: \(error.localizedDescription)")
        }
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool) {
        // Optiona
    }

    func stop() {
        audioPlayer?.stop()
        audioPlayer = nil
    }
}
```

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА IOS ДОДАТКУ ДЛЯ ОПОВІЩЕНЬ ПРО НАБЛИЖЕННЯ ДО ШВИДКІСНОГО ЛІМІТУ

Виконав: студент 4-го курсу, групи 2КН-21б
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Кириленко В.В.
(прізвище та ініціали)

Керівник: асистент каф. КН
Сілагін Є.О.
(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Рисунок В.1 – Загальна архітектура додатку для сповіщення про перевищення швидкісного ліміту

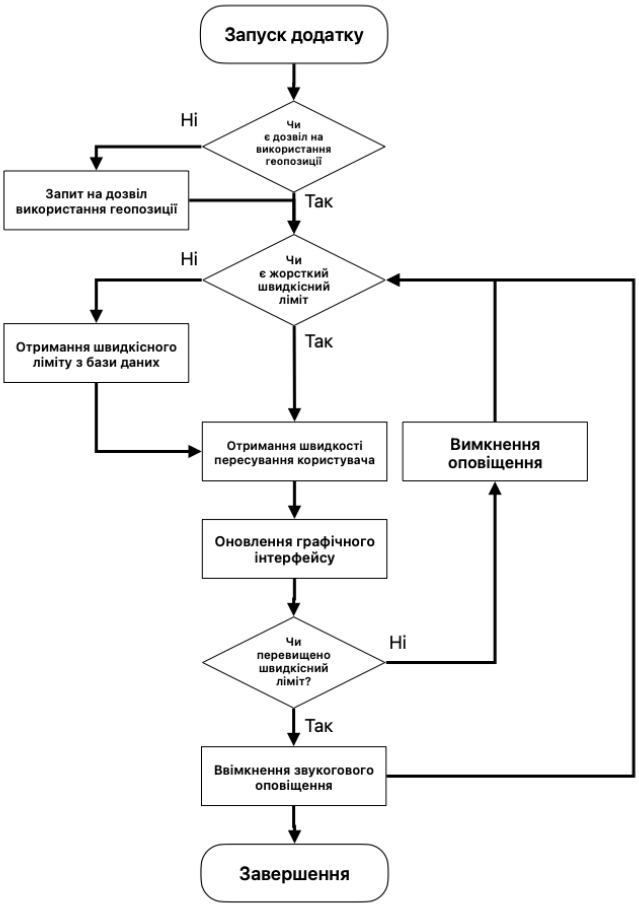

Рисунок В.2 – Загальний алгоритм роботи програмного модуля для сповіщення про перевищення швидкісного ліміту

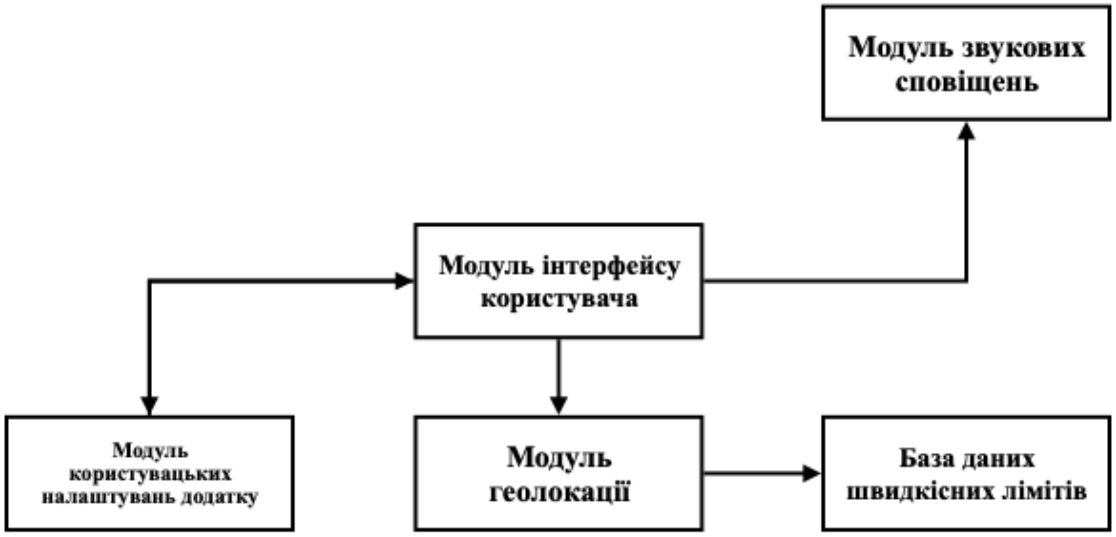

Рисунок В.3 – Структурна схема програмних модулів додатку для оповіщення про наближення до швидкісного ліміту

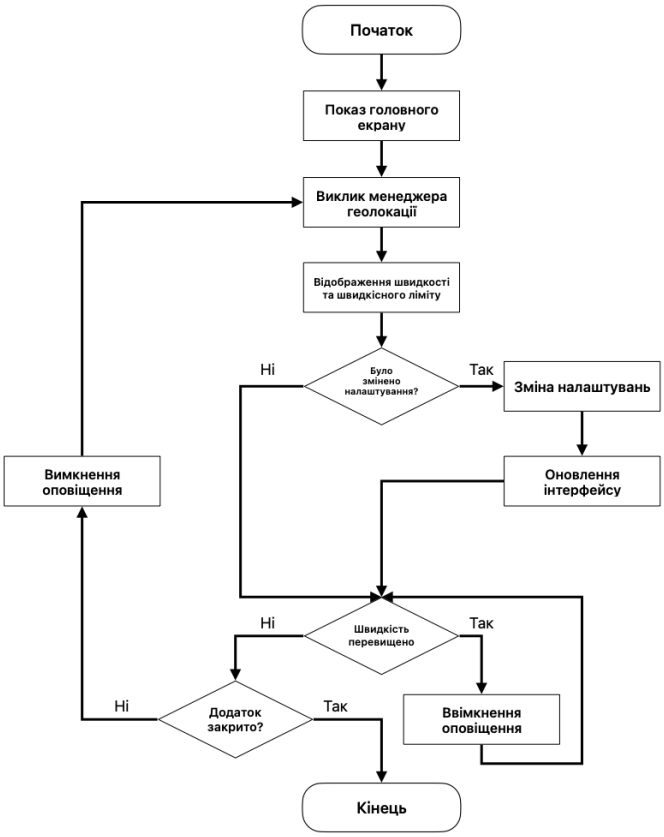

Рисунок В.4 – Алгоритм роботи модуля графічного інтерфейсу

Рисунок В.5 – Алгоритм роботи модуля звукового сповіщення

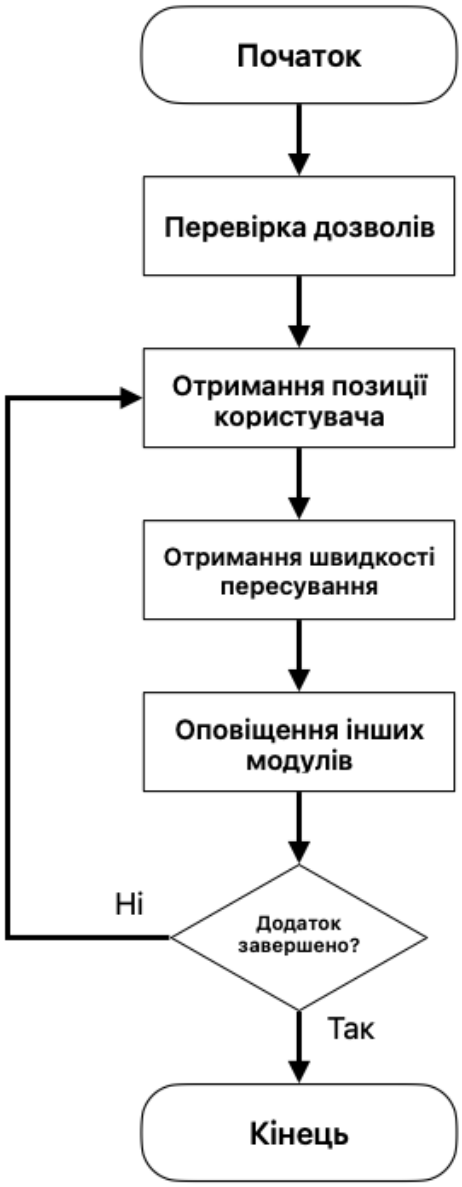

Рисунок В.6 — Алгоритм роботи модуля геолокації

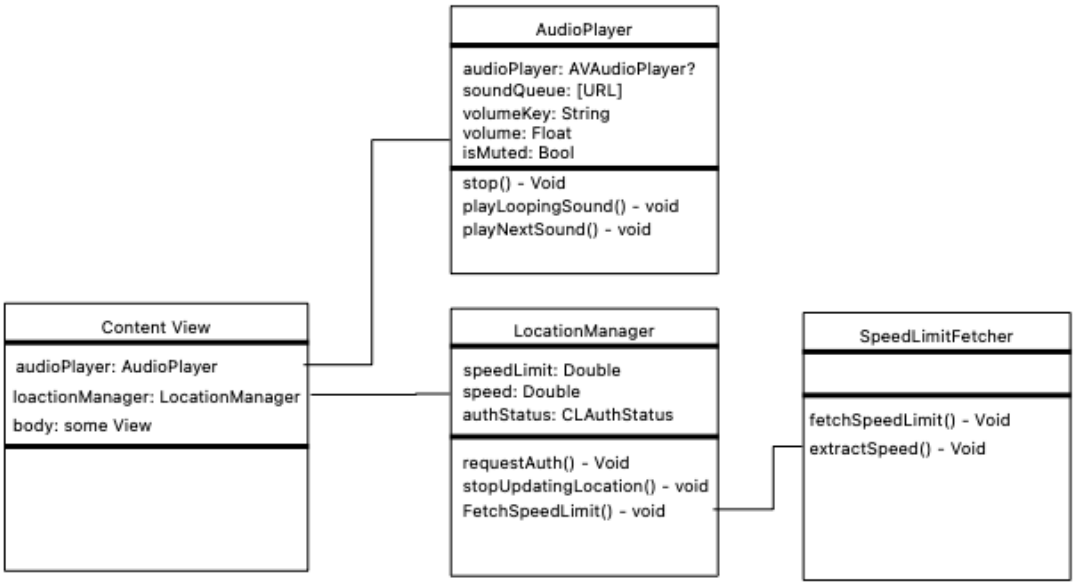

Рисунок В.7 — UML-діаграма класів системи сповіщення про наближення до швидкісного ліміту

Рисунок В.8 – Загальний вигляд інтерфейсу головного вікна додатку

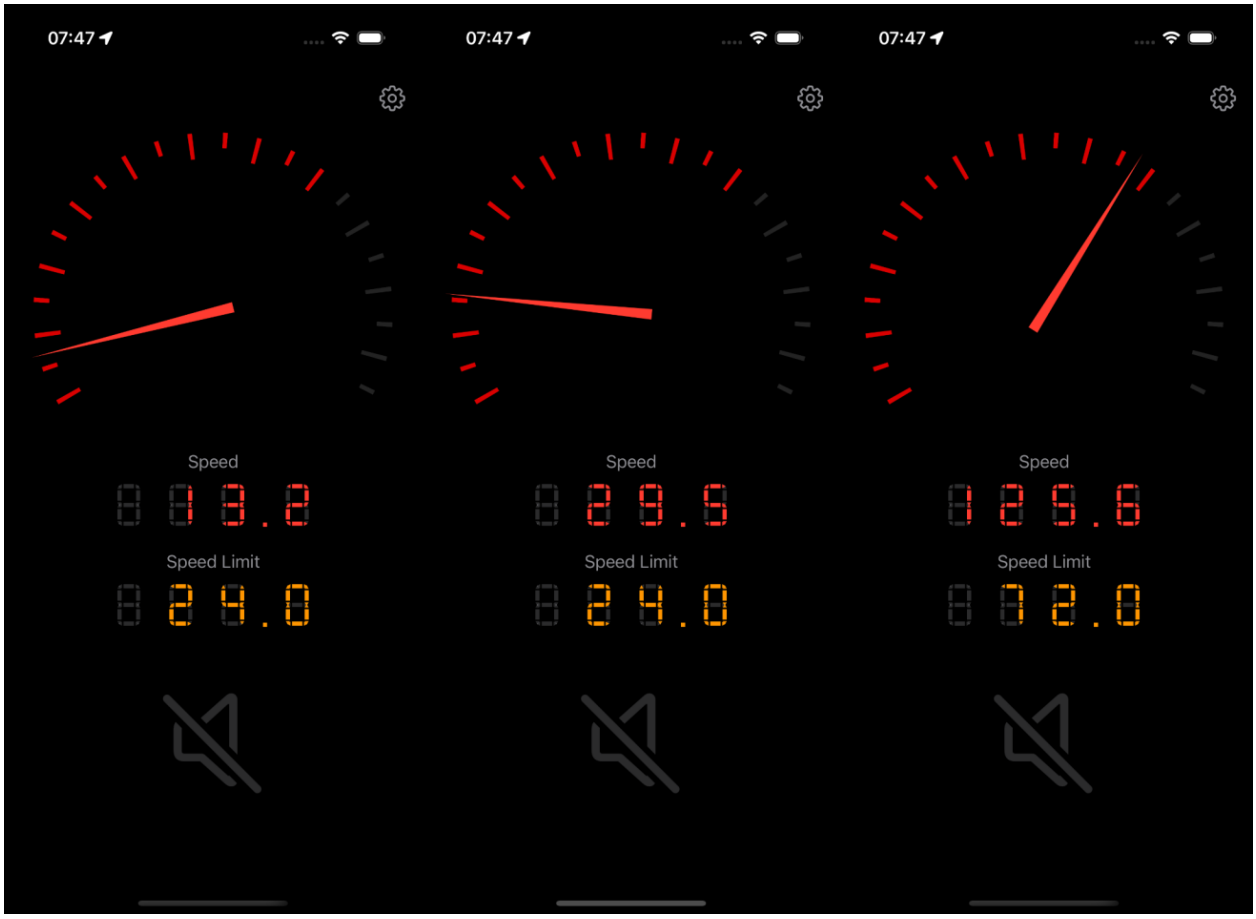

Рисунок В.9 – Результат тестування швидкості та ліміту швидкістю за допомогою вбудованих функцій симулятору

Рисунок В.10 – Знімок екрану з включеними сповіщеннями і перевищенням швидкісного ліміту

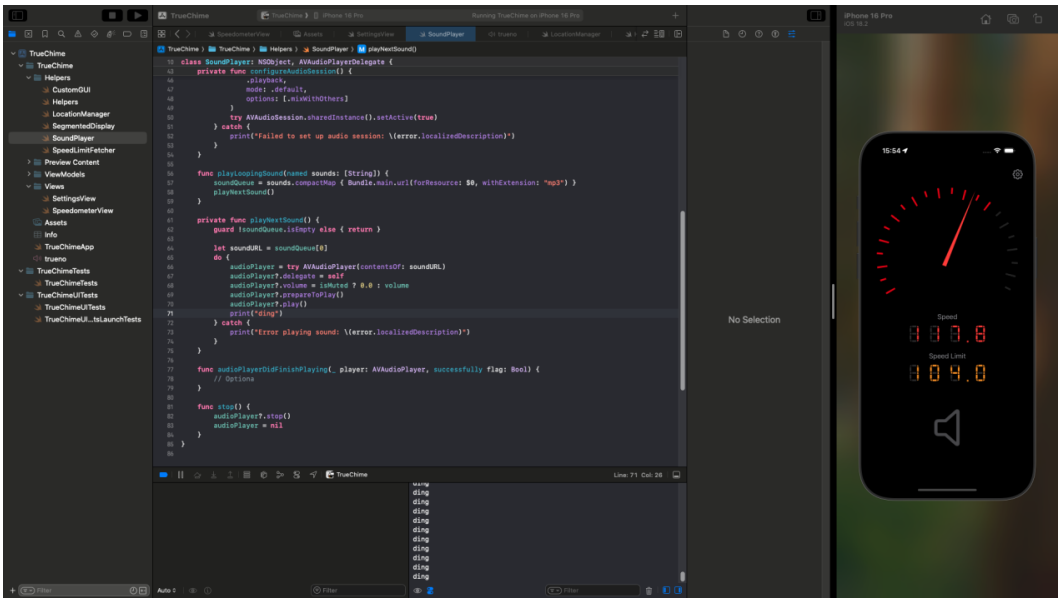

Рисунок В.11 – Результат функціонування додатку з тимчасовими змінами для відображення ввімкення звукового сигналу

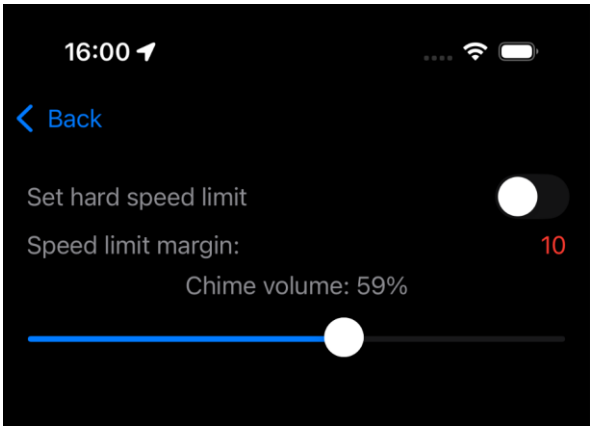

Рисунок В.12 – Загальний вигляд інтерфейсу вікна налаштувань додатку з порогом спрацювання дзвіночка у 10 км/год

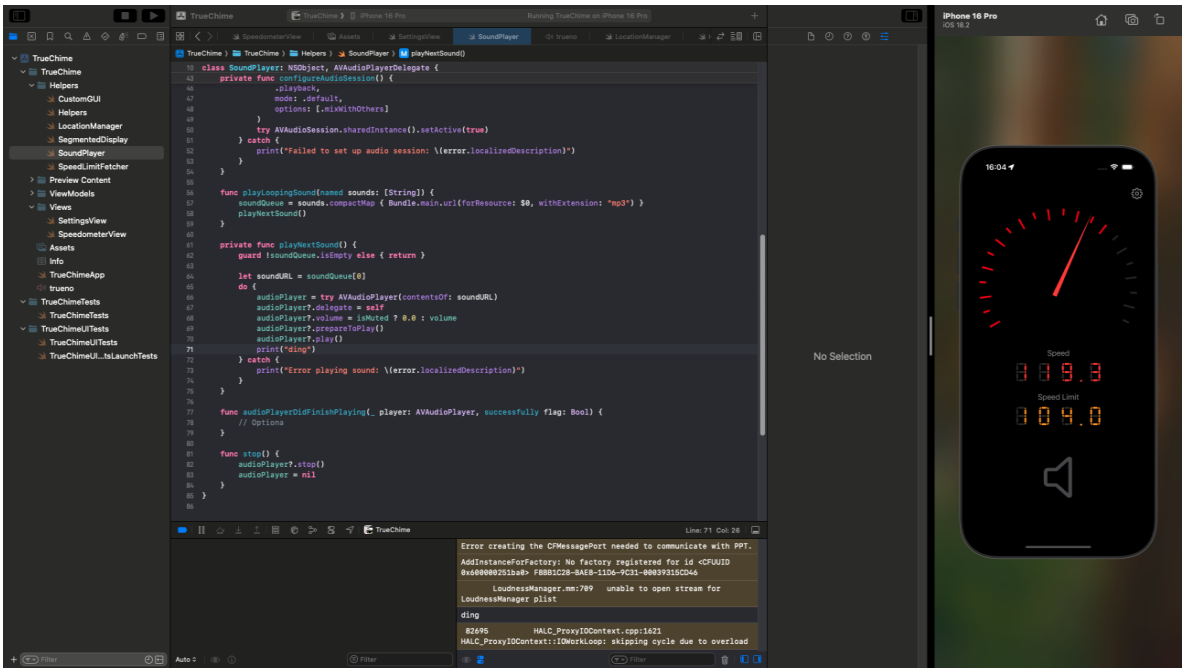

Рисунок В.13 – Результат роботи програми при встановленому порогу сповіщень у 25 км/год

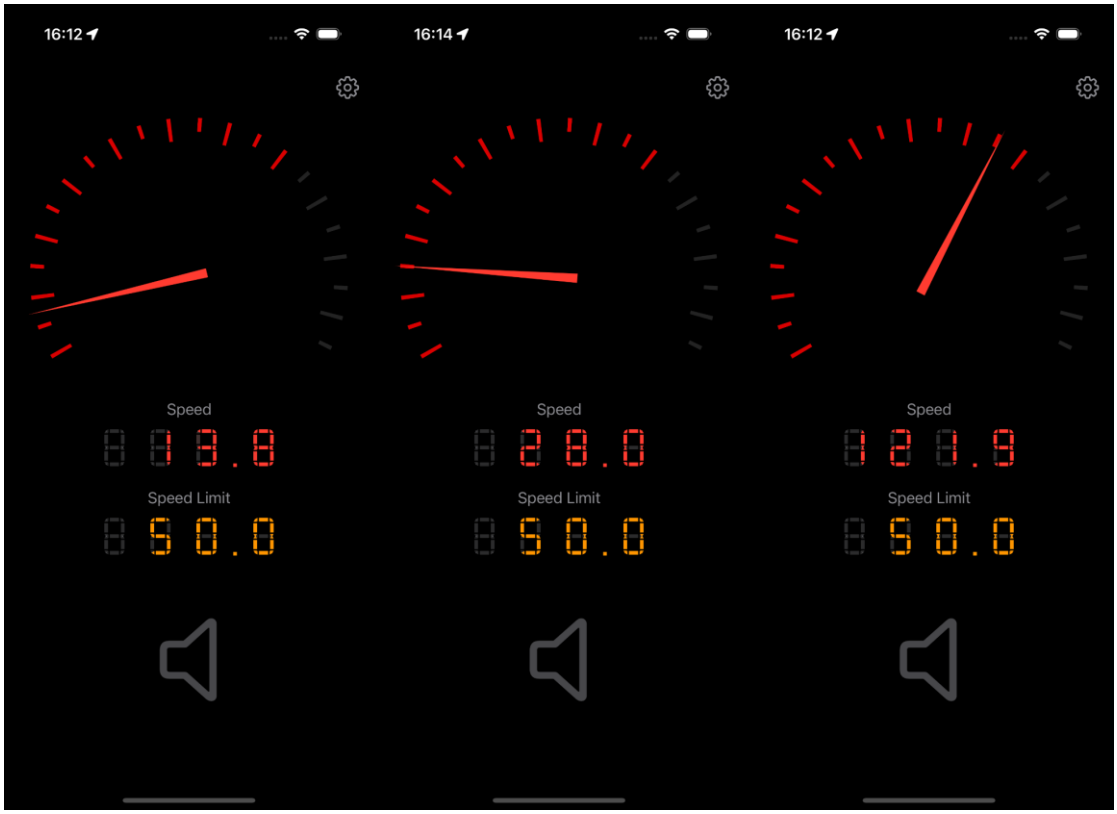

Рисунок В.14 – Результат тестування швидкості та ліміту швидкістю за допомогою вбудованих функцій симулятору при встановленні жорсткого швидкісного ліміту

ДОДАТОК Г (ДОВІДКОВИЙ)
ІНСТРУКЦІЯ КОРИСТУВАЧА

1) Після завантаження додатку на пристрій, на головному екрані з’явиться іконка додатку зображена на рисунку Г.1

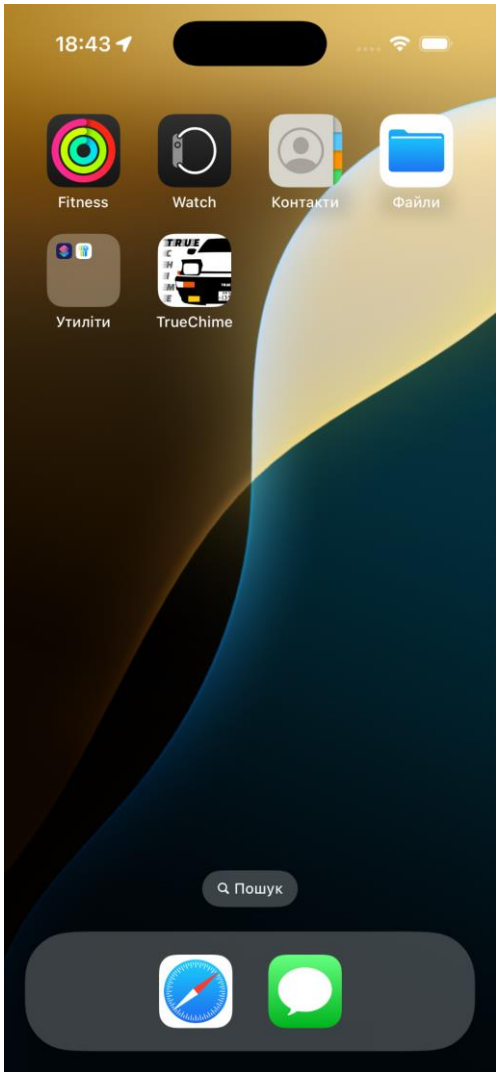

Рисунок Г.1 – Головний екран пристрою з встановленим додатком TrueChime

2) Після натиску на іконку додатку після процесу завантаження відкриється головний екран додатку зображений на рисунку Г.2.

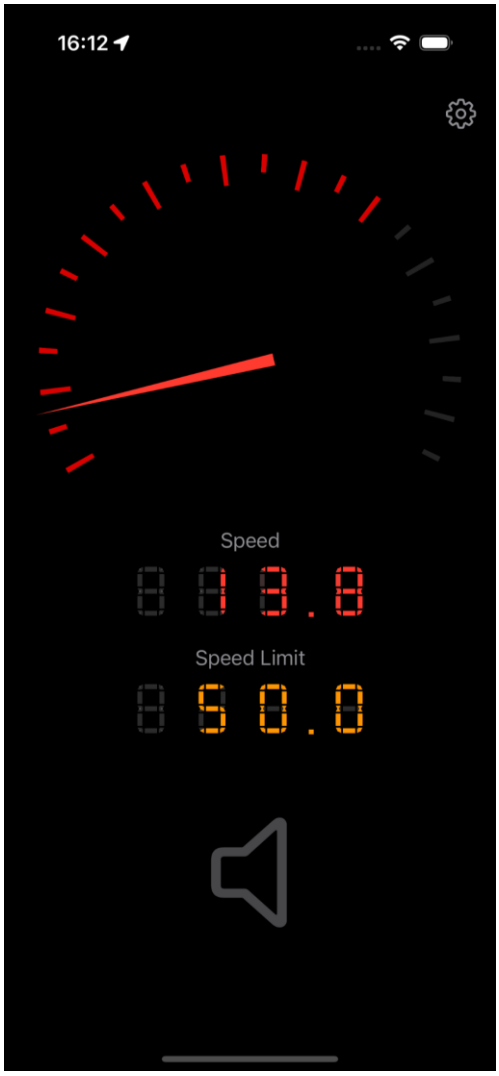

Рисунок Г.2 – Головний екран застосунку

3) Також при першому запуску з’явиться алерт який запросить дозвіл на використання даних про геолокацію пристрою зображений на рисунку Г.3. Для отримання доступу до повного функціоналу додатку потрібно обрати: “Дозволяти за використання”

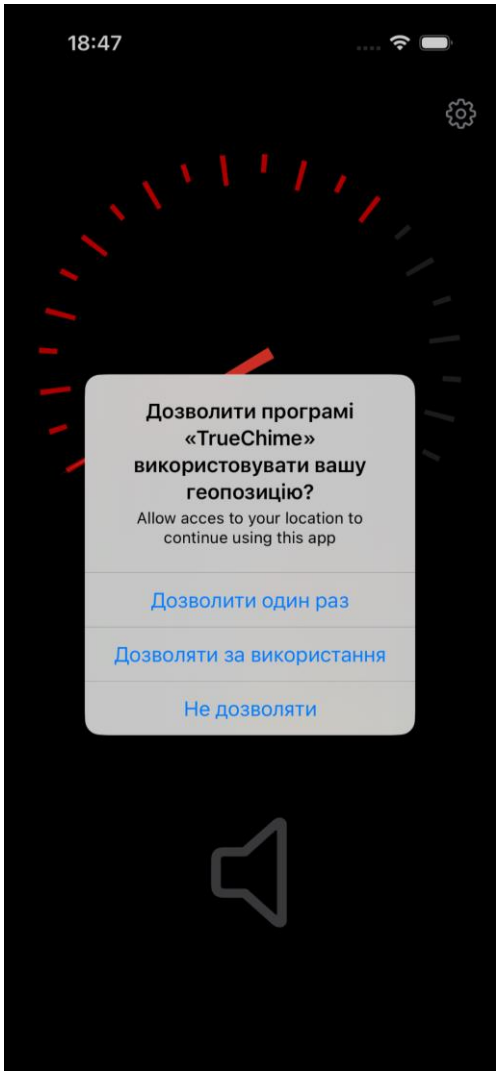

Рисунок Г.3 – Зовнішній вигляд інтерфейсу додатку

4) На цьому етапі налаштування додатку завершено. Можна використовувати функції додатку, такі як тимчасове вимкнення сповіщень зображене на рисунку Г.4.

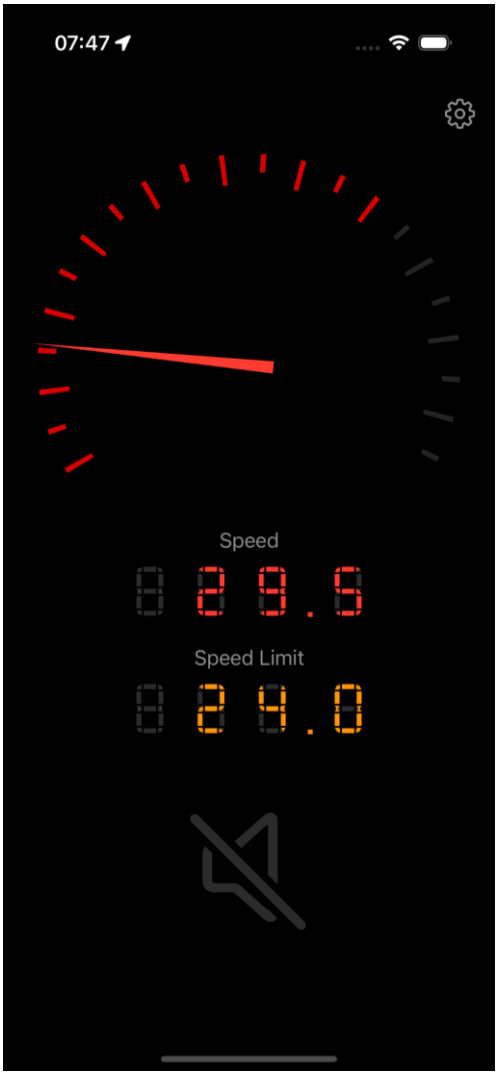

Рисунок Г.4 – Зовнішній вигляд інтерфейсу додатку з вимкненими оповіщеннями

5) Інші налаштування знаходяться у вікні налаштувань, яке відкривається натиском на іконку у вигляді шестерні що знаходиться у верхньому правому куту головного екрану додатку. Екран налаштувань зображено на рисунку Г.5

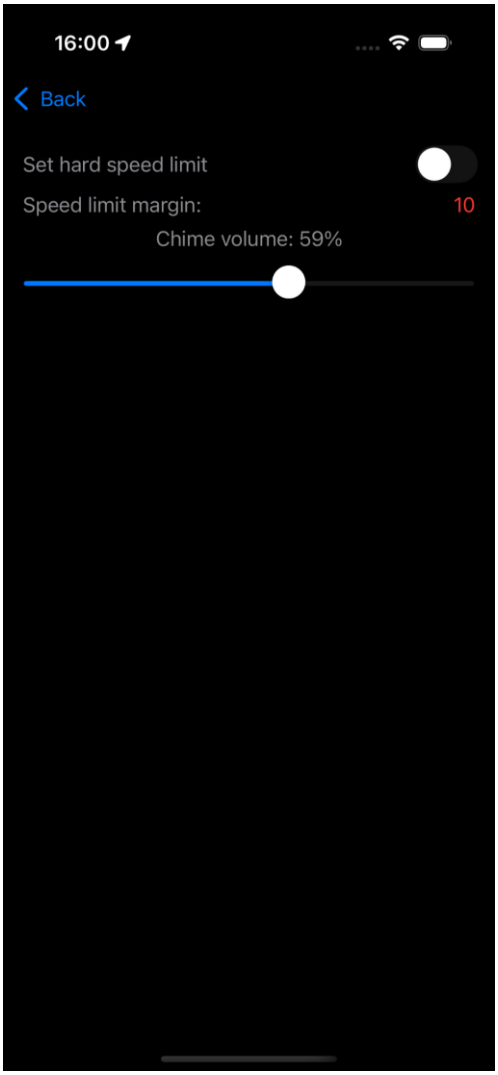

Рисунок Г.5 – Зовнішній вигляд екрану налаштувань

6) На цьому екрані можна встановити гучність звуку за допомогою повзунка під текстом “Chime volume”, встановити поріг спрацювання дзвіночка натиснувши на червоний текст справа від напису “Speed limit margin” або ввімкнути фіксований ліміт швидкості за допомогою чекбоксу справа від напису “Set hard speed limit”
7) Для завершення роботи з додатком достатньо вийти з нього та вимкнути програму. Всі користувацькі налаштування автоматично зберігаються.